2003-09-19

A Five-Zone Model for Direct Injection Diesel Combustion

Rich Asay
Brigham Young University - Provo

A FIVE-ZONE MODEL FOR DIRECT INJECTION DIESEL COMBUSTION

by

Richard J. Asay

A thesis submitted to the faculty of

Brigham Young University

in partial fulfillment of the requirements for the degree of

Master of Science

Department of Mechanical Engineering

Brigham Young University

August 2003

BRIGHAM YOUNG UNIVERSITY

GRADUATE COMMITTEE APPROVAL

of a thesis submitted by

Richard J. Asay

This thesis has been read by each member of the following graduate committee and by majority vote has been found to be satisfactory

Date	Dale R. Tree, Chair
Date	R. Daniel Maynes
Date	Mardson Q. McQuay

BRIGHAM YOUNG UNIVERSITY

As chair of the candidates graduate committee, I have read the thesis of Richard J. Asay in its final form and have found that (1) its format, citations, and bibliographical style are consistent and acceptable and fulfill university and department style requirements; (2) its illustrative materials including figures, tables, and charts are in place; and (3) the final manuscript is satisfactory to the graduate committee and is ready for submission to the university library.

Date

Dale R. Tree
Chair, Graduate Committee

Accepted for the Department

Brent L. Adams
Graduate Coordinator

Accepted for the College

Douglas M. Chabries
Dean, College of Engineering and Technology

ABSTRACT

A FIVE-ZONE MODEL FOR DIRECT INJECTION
DIESEL COMBUSTION

Richard J. Asay

Department of Mechanical Engineering

Master of Science

Recent imaging studies have provided a new conceptual model of the internal structure of direct injection diesel fuel jets as well as empirical correlations predicting jet development and structure. This information was used to create a diesel cycle simulation model using C language including compression, fuel injection and combustion, and expansion processes. Empirical relationships were used to create a new mixing-limited zero-dimensional model of the diesel combustion process. During fuel injection five zones were created to model the reacting fuel jet: 1) liquid phase fuel 2) vapor phase fuel 3) rich premixed products 4) diffusion flame sheath 5) surrounding bulk gas. Temperature and composition in each zone is calculated. Composition in combusting zones was calculated using an equilibrium model that includes 21 species. Sub models for ignition delay, premixed burn duration, heat release rate, and heat transfer were also included.

Apparent heat release rate results of the model were compared with data from a constant volume combustion vessel and two single-cylinder direct injection diesel engines. The modeled heat release results included all basic features of diesel combustion. Expected trends were seen in the ignition delay and premixed burn model studies, but the model is not predictive. The rise in heat release rate due to the diffusion burn is over-predicted in all cases. The shape of the heat release rate for the constant volume chamber is well characterized by the model, as is the peak heat release rate. The shape produced for the diffusion burn in the engine cases is not correct. The injector in the combustion vessel has a single nozzle and greater distance to the wall reducing or eliminating wall effects and jet interaction effects. Interactions between jets and the use of a spray penetration correlation developed for non-reacting jets contribute to inaccuracies in the model.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1. INTRODUCTION

Combustion processes are important in most practical power generation devices, among which diesel engines are the most efficient. Understanding the details of the combustion processes in these applications is essential in predicting performance and pollutant formation. The widespread use of diesel engines and the increasingly stringent regulations on emissions make the diesel combustion process a subject of considerable research importance.

Diesel combustion is extremely complex. At present, it is not possible to construct models that predict engine operation from the basic governing equations alone [1]. Computer models have been written that attempt to describe and predict the combustion process. These codes vary in complexity from simple correlations and zero-dimensional models with no spatial resolution to multi-dimensional finite difference models. While multi-dimensional models attempt to incorporate more fundamental equations and relationships, zero-dimensional models tend to rely more heavily on correlations and empirical data. Recently, many of the correlations and empirical relationships have been developed from imaging studies that have been performed to determine the structure of the diesel flame. These studies have used lasers and other methods to see inside the reacting fuel spray. Recently, in-cylinder and combustion bomb studies have produced new information regarding spray penetration, ignition delay,

NO_x and soot formation zones and flame lift-off. The imaging studies give a temporal as well as a spatial picture of the details of the combustion process.

The objective of this thesis is to produce a new zero dimensional, multi-zone model of diesel combustion that incorporates this new information. The proposed model will provide temperature, size, and species predictions for five zones: the liquid fuel, premixed reaction, rich products core, diffusion flame sheath and surrounding bulk gas. Sub models will include: 1) a spray geometry model, 2) an ignition delay model, 3) a heat release rate model, 4) a product equilibrium model and 5) a simple heat transfer model. This model will then be integrated to produce a cycle simulation model and used to predict engine performance parameters and pollutant formation. Predicted heat release rate and pressure will be compared to experimental data from a constant volume combustion vessel and two diesel engines.

2. BACKGROUND

2.1. THE DIESEL COMBUSTION PROCESS

The diesel cycle consists of five processes: intake, compression, fuel heat release, expansion, and exhaust. Modeled heat release is determined by modeling fuel injection, ignition delay, and combustion. A cycle simulation could model all of these processes. This work will focus on the closed portion of the cycle and exclude the intake and exhaust processes. The compression and expansion are adequately modeled using a first law energy balance of the cylinder and an appropriate heat transfer model. The greatest challenge lies in modeling the injection, ignition, and combustion process. The spray geometry, ignition delay, heat release rate, heat transfer and product equilibrium predictions must all be modeled in order to provide information necessary to produce basic performance and emissions predictions. In order to provide the background for modeling these processes, old and new conceptual models of diesel combustion will first be reviewed in addition to a review of current zero-dimensional models in the literature. A new zero-dimensional model will then be proposed based on the new conceptual combustion model.

2.2. THE OLD CONCEPTUAL MODEL OF DIESEL COMBUSTION

Because of the difficulty of studying diesel combustion in the engine, early conceptual models were based on models of other types of spray combustion. Generally, the models assumed a spray with a liquid fuel core or atomized spray surrounded by a

flame. The fuel was thought to burn at a mixture close to stoichiometric conditions but it was uncertain as to whether combustion took place in a flame sheath around the entire spray or around individual fuel droplets in an atomized spray. The size distribution and penetration length of liquid droplets in non-reacting sprays were measured but were essentially unknown in reacting sprays.

This former model was based on spray and combustion fundamentals and not on direct observation. There was uncertainty about the details of combustion. Engineers realized the uncertainty of this conceptual model but measurements were difficult to obtain. A more complete discussion of the old model can be found in Dec [2].

2.3. NEW CONCEPTUAL MODEL

Taking advantage of optical diagnostics and improved experimental facilities, laser and other imaging studies have been used during the past decade to improve our understanding of the structure of the diesel flame. Dec [2] and Flynn et al. [3] have compiled the results of many of these studies into a complete picture of the structure of the diesel spray. The conceptual model describes how the spray develops and combustion begins. It also describes the quasi-steady portion of combustion. A review of the conceptual model introduced by Dec [2] is given below.

2.3.1. TEMPORAL MODEL

As liquid fuel is injected into the cylinder, high temperature air is entrained into the fuel forming a cone shaped spray. This high temperature air evaporates and mixes with the fuel. Studies done on this portion of the spray have determined that there is a maximum penetration distance for the liquid fuel after which all the fuel will vaporize

[4,5]. This distance is called the liquid length. This length varies with fuel properties, combustion chamber conditions, and injector geometry, but is basically the result of an energy balance between the energy of the air entrained and the energy required to vaporize the fuel. Higher air density and temperature in the cylinder cause more air to be entrained and reduce the liquid length. Surprisingly, the liquid length is not correlated with droplet size or injection pressure except to the extent that these parameters cause more or less air to be entrained.

Beyond the liquid length, the fuel vapor and air continue to mix and penetration continues into the combustion chamber. With the high temperatures in the cylinder, autoignition begins when the fuel-air equivalence ratio and the temperature in the spray reach combustible limits. Ignitable mixtures would of necessity exist beyond the liquid length or on the sides of the spray where the fuel has evaporated and has the opportunity to heat above the boiling point. Combustion of this premixed, vaporized-fuel and air mixture is thought to occur volumetrically in a premixed reaction rather than a propagating premixed flame. The initial premixed burn involves the combustion of these rich premixed vapors and consumes all the oxygen previously entrained in the vapor portion of the spray. At the end of the initial premixed burn, a diffusion flame surrounds the rich products of the premixed reaction. This diffusion flame appears to form where the equivalence ratio of the mixture is close to one. Once the diffusion flame is formed and the initial premixed burn ends, the combustion process becomes quasi-steady. A schematic diagram of a quasi-steady diesel combustion flame jet is shown in Figure 2.1.

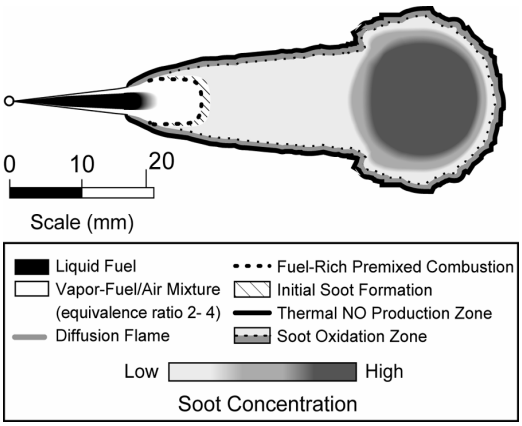


Figure 2.1. Quasi-steady diesel combustion plume as presented by Dec (1997) [2]. Used by permission

2.3.2. QUASI-STEADY MODEL

During this phase the liquid length remains a feature of the spray. The liquid length shortens a small amount due to combustion that increases the energy of the air entrained into the jet. Beyond the liquid length the fuel is all in the vapor phase mixed with hot entrained air. This mixture increases in temperature until reacting in the fuel rich premixed combustion zone. For normal diesel operation the equivalence ratio of this mixture upon reacting is thought to be between 2 and 4 (Dec [2]) and thus is termed a rich premixed burn. The reaction consumes the available oxygen leaving combustible materials such as CO and unburned hydrocarbon. The rich premixed burn produces products at intermediate temperatures, 1600 – 1900 K, but does not reach the stoichiometric flame temperature because a large portion of chemical energy remains in the CO and unoxidized products due to a lack of oxygen.

A diffusion flame forms an envelope not only around the rich premixed products but also locates around the liquid spray forming a lifted diffusion flame. The distance from the nozzle to the edge of the diffusion flame is called the *lift-off length*. The lift-off length is extremely important in determining th e characteristics of the flame because it determines how much air is entrained into the rich premixed reaction zone. It is in the rich premixed reaction zone where soot is initially formed

The temperature of the flame sheath is near the stoichiometric adiabatic flame temperature. The lift-off length is typically shorter than the liquid length for the current operating conditions of most modern diesel engines. Longer lift-off lengths allow more air entrainment increasing oxygen in the rich premixed reaction zone and thereby decrease soot formation.

The products of this premixed reaction continue to move axially away from the nozzle and spread radially outward. Because the liquid length and premixed burn location remain fixed until the end of combustion, this portion of the combustion process is termed quasi-steady although the leading edge continues to move forward and expand radially.

At the end of injection, the liquid portion of the spray slows and stops flowing out of the nozzle. The last bit of liquid moves away from the nozzle but the liquid length remains relatively constant leading to an end of the liquid phase. The slowing of the spray allows the flame to move closer to the nozzle, shortening the lift-off length. Rapidly, the diffusion flame encircles the entire jet. At this point a significant portion of the fuel energy remains unreacted but the conceptual model is no longer well understood. Images of combustion in the cylinder following the quasi-steady phase show pockets of

rich premixed products (containing soot) surrounded by a diffusion flame (a thin layer of OH). It appears that the rich products within the jet are carried toward the wall where the jet splits and forms large-scale turbulent structures that entrain air and continue to burn as a diffusion flame.

2.4. DIESEL COMBUSTION MODELS

There are two categories of models used to describe diesel combustion, comprehensive three-dimensional models and phenomenological models or zero dimensional models. Three-dimensional models use computational fluid dynamic (CFD) codes to describe the flow field and temperature conditions in the combustion chamber. Sub models are included for spray breakup, ignition, heat transfer, and pollutant formation. Phenomenological models separate the combustion chamber into different zones and can be categorized by the number of these zones. There are one, two, three, and multi-zone models. These models are often called zero-dimensional because they provide no spatial resolution. More detailed explanations of these model types can be found in [1], and [6-9].

2.4.1. THREE-DIMENSIOINAL MODELS

CFD has been used extensively for modeling engines. In addition to modeling the three-dimensional flow, engine CFD codes require a moving boundary condition for piston movement. With a complex flow model it is normally necessary to model combustion using the same type of correlations that are used in zero dimensional models. This diminishes the benefits of 3-D models. Due to the complexity of these models, they demand large amounts of computer power. The time required for the calculations is also

limiting. This is especially important when many iterations are required to determine constants or to determine the effects of changing parameters. This type of model will not be considered for this thesis. Examples of these models can be found in [10,11].

2.4.2. ONE-ZONE MODELS

Single zone models are advantageous because of their simplicity and are widely used along with empirical data within the engine industry to make design decisions. This method assumes that the entire volume of the combustion chamber is a homogeneous mixture of air and combustion products. It then assumes that the fuel is burned immediately on injection into the combustion chamber [7]. Examples of this type of model can be found in [1,12,13]. Often, the measured pressure rise in an engine is used to tune the model or is used to provide a rate of heat release. Zero dimensional models are therefore usually not entirely predictive but are used to extend the value of the measured data.

The disadvantage of single zone models is that they cannot fully describe the complex phenomena that comprise the compression-ignition engine combustion process and substantial empirical input must be used [1].

2.4.3. TWO-ZONE MODELS

A two-zone model separates the combustion chamber into a fuel/air mixture zone (zone 1) and a zone of air and combustion products (zone 2) as illustrated in Figure 2.2. This two-zone structure is used extensively in spark ignition engine models [7].

However, Salem et al. [14] used this type of model for a diesel engine where the spray was the air-fuel mixture zone. The fuel is assumed to burn at the edge of the spray.

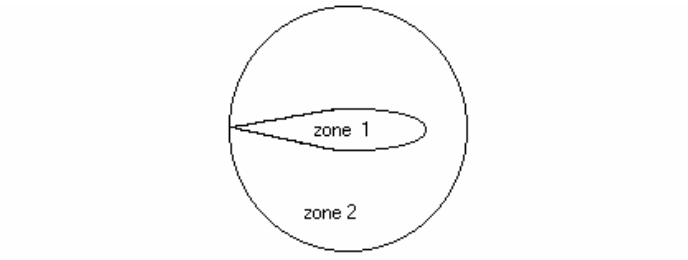


Figure 2.2. Two zone model

2.4.4. THREE-ZONE MODELS

Foster [7] describes a model by Gao et al that uses three zones to describe combustion in an ethanol fumigated diesel engine (Fig. 2.3). In this model two of the zones contain the air-fuel mixture. Zone 1 contains air and ethanol, and zone 2 contains diesel fuel, air, and ethanol. The third zone contains the products of combustion. The diesel fuel is assumed to burn first, resulting in the destruction of one of the air-fuel zones. The air-ethanol zone is then assumed to burn.

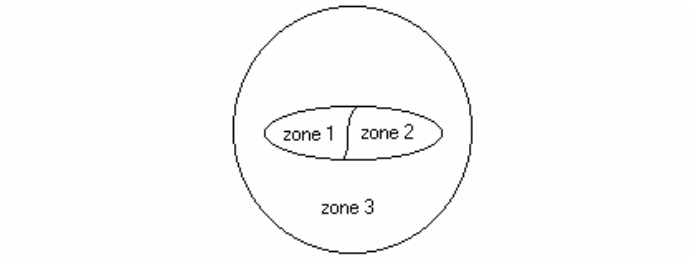


Figure 2.3. Three zone model

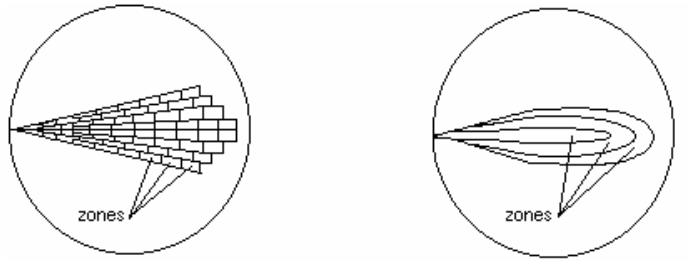


Figure 2.4. Multi-zone models

2.4.5. MULTI-ZONE MODELS

Multi-zone models separate the fuel spray into a large, finite number of zones. Two examples of these types of models are shown in Figure 2.4. In one, the zones are small packets of fuel that move through the combustion chamber. The state of the mass in each packet is tracked as it moves. This method has been used by several researchers [6, 15-21]. A second type of multi-zone model [22-24] follows the development of equivalence ratio zones surrounding a central liquid core. The fuel is then assumed to burn when the equivalence ratio is equal to one.

2.5. SUMMARY AND NEED FOR A NEW MODEL

The new conceptual model suggests several features of diesel combustion that are not consistent with previous models. In the new conceptual model, the concept of a liquid length has been introduced which is not described by any of the previous models. It is also now known that there is not a solid liquid core but an atomized spray forming a jet of fuel and air. The liquid length can be correlated with in-cylinder temperature and pressure and fuel properties. It is now known that droplets do not penetrate beyond the liquid length. This is different from the previous view of a liquid core penetrating deep

into the spray. The new conceptual model involves a two-step combustion process that is spatially separated and not a single diffusion flame combustion process. The new conceptual model has shown that soot forms within the rich premixed product zone and burns out as it passes through the flame sheath, NO is not formed inside of the flame but only on the lean side of the flame sheath. A sub-model that describes the temperature and stoichiometry of these two zones where soot and NO are formed does not exist in current models. Describing the temperature of these zones with a zero dimensional model will allow the computational power to be used on comprehensive kinetic mechanisms describing NO and soot formation.

The conceptual model proposed by Dec [2] has been used to create a new multi-zone, zero-dimensional, computer model written in C language. The computer program simulates the initial spray formation and ignition process as well as the processes described in Dec's conceptual model. The quasi-steady model is separated into five zones (Fig. 5).

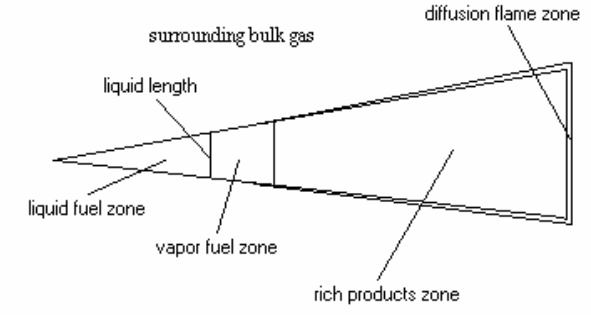


Figure 2.5. Five zone spray combustion model

The five zones are:

1. Liquid phase fuel
2. Vapor phase fuel
3. Rich products
4. Diffusion flame
5. Surrounding bulk gas

This model includes sub models for spray geometry, ignition delay, heat release rate, product concentrations, and heat transfer. This model has been integrated into a cycle simulation program.

2.6. OBJECTIVES

The objective of this thesis is to complete a five-zone diesel combustion model including sub models for: 1) spray geometry, 2) ignition delay, 3) heat release rate, 4) equilibrium product concentrations, and 5) heat transfer. Prediction of the product concentrations will require the development of a combustion equilibrium code. The combustion model will be integrated into a cycle simulation model. The results of the combustion model and cycle simulation code will be compared with experimental results obtained from a constant volume combustion vessel and from the results of two diesel engines.

2.7. CONTRIBUTION

Numerous zero dimensional models have been written based on the old conceptual model to predict diesel combustion phenomena. The recent combustion imaging studies of Dec [2] and the spray penetration studies by Siebers [4,27] provide

new information, which has not yet been incorporated into modeling efforts. This work will produce a new model of diesel combustion, which incorporates this new information. By placing the new conceptual information in the quantitative framework of a model, the information will become more useful for engine design and development.

2.8. DELIMITATION

Due to the complexity of the problem, the following simplifications will be made in this initial effort to produce a new simulation program. The combustion chamber will be assumed quiescent. Jet impingement on the cylinder wall will be neglected and the combustion process will be assumed to continue in a quasi-steady manner. Only simple, existing models for heat transfer and ignition delay will be used.

3. SPRAY CORRELATION AND EQUILIBRIUM MODEL

Before describing the cycle simulation, two models used throughout the simulation will be explained. First the spray model will be explained, followed by the equilibrium model.

3.1. SPRAY MODEL

The spray model is used to calculate the geometry of the fuel spray and defines the amount of cylinder gas entrained into the spray. The spray model predicts spray penetration length, liquid length, dispersion angle, and local equivalence ratio axially along the length of the jet. The jet geometry is used in the cycle simulation to define zones that have distinct phase, species, or temperature characteristics. The following is a brief description of the spray penetration and liquid length models and the equations involved. For more information consult Refs. [25,27].

3.1.1. SPRAY PENETRATION MODEL

The spray penetration model was developed at Sandia National Laboratories [25] and is given in Equation 3.1 below. The penetration correlation was developed using mass and momentum balances for a non-vaporizing spray of uniform velocity, temperature, and composition. The correlation was compared with penetration data from both vaporizing and non-vaporizing sprays. The equation is shown to underpredict the

penetration of a non-vaporizing spray by at most 18% and over predict the penetration of an evaporating spray by at most 21%.

$$\tilde{t} = \frac{\tilde{S}}{2} + \frac{\tilde{S}}{4}\sqrt{1+16\tilde{S}^2} + \frac{1}{16}\ln\left(4\tilde{S} + \sqrt{1+16\tilde{S}^2}\right) \tag{3.1}$$

In the spray penetration equation $\tilde{t}$ is the dimensionless time and $\tilde{S}$ is the dimensionless distance of the spray penetration defined by

$$\tilde{t} = \frac{t}{t^+} \tag{3.2}$$

$$\tilde{S} = \frac{S}{S^+} \tag{3.3}$$

where t and S are the actual time and spray penetration distance and t^+ and S^+ are defined as

$$t^+ = d_f \tilde{?}^{1/2} \frac{1}{U_f \tan(a/2)} \tag{3.4}$$

$$S^+ = d_f \tilde{?}^{1/2} \frac{1}{\tan(a/2)} \tag{3.5}$$

In these equations d_f is the effective spray nozzle orifice diameter calculated by multiplying the actual orifice diameter by the area contraction coefficient to the ½ power.

$$d_f = Ca^{1/2} d_o \tag{3.6}$$

$\tilde{?}$ is the ratio of fuel density to air density.

$$\tilde{?} = \frac{?_f}{?_a} \tag{3.7}$$

U_f is the fuel injection velocity calculated as

$$U_f = C_v \sqrt{2\frac{(P_f - P_a)}{?_f}} \tag{3.8}$$

where C_v is the velocity contraction coefficient (C_d/C_a), P_f is the fuel injection pressure, P_a is the air pressure and ρ_f is the fuel density.

Because of the uniform velocity profile assumption stated earlier, a real spray with the same momentum as the modeled spray will have a larger cone angle. The relationship between the measured (θ) and modeled (α) spray angle, given by Equation 3.9, was found experimentally.

$$\tan(a) = .66\tan(?) \tag{3.9}$$

A correlation given in [26] was used to predict the measured spray angle θ as seen in Equation 3.10.

$$\tan\left(\frac{?}{2}\right) = 0.26\left[\left(\frac{?_a}{?_f}\right)^{0.19} - .0043\sqrt{\frac{?_f}{?_a}}\right] \tag{3.10}$$

Given the shape of the spray (a cone), the volume filled with entrained gasses can be calculated. Then, assuming the entrained gases to be air, the equivalence ratio at any position can be calculated using Equation 3.10.

$$f(\tilde{S}) = \frac{2A/F_s}{\sqrt{1+16\tilde{S}^2}-1} \tag{3.10}$$

In this equation A/F_s is the stoichiometric air/fuel ratio.

Equation 3.1 cannot be explicitly solved for $\tilde{S}$, and therefore an iterative solution was used to calculate the penetration length. These equations can be used to determine either the penetration length at any given time after the start of injection, or the time

required to reach a certain penetration length. Both are used in the cycle simulation program along with the equivalence ratio calculation.

3.1.2. LIQUID LENGTH MODEL

The liquid length model predicts the maximum distance liquid fuel penetrates into the cylinder [27]. An energy balance determines the energy from in-cylinder gasses required to vaporize the injected fuel. When the energy entrained in the spray is sufficient to vaporize the fuel, the spray is considered to have reached maximum liquid fuel penetration called the liquid length. The correlation developed at Sandia [27] was used as given by Equation 3.11.

$$\frac{LL}{d_o} = k\,\tilde{?}^{\alpha}\,B^{ß} \qquad (3.11)$$

Where: LL is the liquid fuel penetration length,

d_o is the orifice diameter

$\tilde{r}$ is the same density ratio as defined by Equation 3.7

k = 10.5, α = 0.58, and β= 0.59 are empirical constants, Higgins (Ref [27])

B is termed the "specific energy ratio" and is given by Equation 3.12.

$$B = \frac{C_{p,liq}\left(T_b - T_f\right) + h_{vap}}{C_{p,air}\left(T_{air} - T_b\right)} \qquad (3.12)$$

In this equation $C_{p,liq}$, and $C_{p,air}$ are the constant pressure specific heats for liquid fuel and air respectively. T_b is the boiling temperature of the fuel, T_{fuel} is the fuel temperature in the injector, and T_{air} is the temperature of the gasses in the cylinder. The heat of vaporization of the fuel is h_{vap}. The parameter B, then, is the ratio of energy required to vaporize the fuel over the energy provided by the outside air.

3.2. EQUILIBRIUM MODEL

The equilibrium code began as an extension of the program by Olikira and Borman Ref [28], however, Olikara and Borman included only CO and CO_2 as possible species where carbon could be distributed in the products which caused the solution to fail at equivalence ratios above approximately 2. The new equilibrium code is expanded to include 21 species instead of only the 12 considered by Olikira and Borman. These species allow the program to solve for equivalence ratios of up to at least 8. In the following sections the species included will first be discussed, followed by the equilibrium reactions used and the equilibrium constants. The solution method will then be explained including the numerical methods employed to find a solution to this complicated problem.

Several equilibrium codes have been developed and used among researchers in the combustion community and some are available for open use. These include STANJAN, a code by Olikira and Borman [28], and the NASA-Lewis code. A brief explanation of the reason behind the development of this code will be briefly given.

STANJAN is a code written in FORTRAN at Stanford University. STANJAN is distributed freely for research purposes as a compiled code, but the compiled version requires a user interface, which cannot b e called as a subroutine by another program as required for this application. Source code for STANJAN can be obtained by joining the STANJAN research group.

The source code for Olikara and Borman was already available in FORTRAN, was written specifically for engine combustion with a reduced set of species and reactions, and was shorter and easier to understand.

The NASA-LEWIS code, written by Gordon and McBride, is an extensive code capable of solving equilibrium problems for several phases and over 100 species. This code was considered to be more comprehensive than necessary and too computationally intensive. It was also anticipated that NASA-Lewis would be more complex to learn and incorporate.

All of the available codes were written in FORTRAN but it was desired to write the cycle simulation code in C. All of the codes would therefore require either a translation into C or a method for interfacing compiled FORTRAN subroutines with the cycle simulation code written in C.

Initially it was decided to translate the relatively short and simple FORTRAN code of Olikara and Borman into C. This was completed in a short time but as mentioned above, the unanticipated need of solving equilibrium under rich conditions above $\phi = 2$ required an expanded set of species and equations and a new method for solving the equations. As a result, this research task became more time consuming than expected.

3.2.1. SPECIES

The NASA-Lewis equilibrium code was used to determine which species were most important to consider. It was executed at various high equivalence ratio conditions. The species with significant mole fractions were added to the equilibrium model. The NASA-Lewis code predicted a significant amount of solid carbon above an equivalence ratio of 3.5. Due to increased complexity of calculation solid carbon was neglected. The species included are shown in Table 3.1.

Table 3.1. Species and assigned species number as occurs in the equilibrium code

1.	H	7.	NO	13.	CH3	19.	HCN
2.	O	8.	O2	14.	CH4	20.	HCO
3.	N	9.	H2O	15.	C2H2	21.	C2H
4.	H2	10.	CO2	16.	C2H4		
5.	OH	11.	N2	17.	NH2		
6.	CO	12.	Ar	18.	NH3		

With the species determined the global reaction equation is shown in equation 3.13.

$$C_nH_mO_lN_k + \frac{n+m/4-l/2}{f}\left(O_2 + 3.7274N_2 + .0444Ar\right) \Rightarrow \tag{3.13}$$
$$N_1H + N_2O + N_3N + N_4H_2 + N_5OH + N_6CO + N_7NO + N_8O_2 + N_9H_2O + N_{10}CO_2 + N_{11}N_2 +$$
$$N_{12}Ar + N_{13}CH_3 + N_{14}CH_4 + N_{15}C_2H_2 + N_{16}C_2H_4 + N_{17}NH_2 + N_{18}NH_3 + N_{19}HCN + N_{20}HCO + N_{21}C_2H$$

In this equation N_i are the number of moles of each species. The fuel composition (n, m, l and k) and equivalence ratio (ϕ) are known and the equation is written assuming 1 mole of fuel. This leaves the moles of each of the 21 species as unknowns. In order to solve this problem there must be 21 equations. The first 5 equations are the carbon, hydrogen, oxygen, nitrogen, and argon balances. The rest of the equations come from the 16 equilibrium reactions shown in Table 3.2.

The order to these reactions was chosen with the solution method in mind. The form of these reactions will be discussed further with the solution method. These reactions can be used to write the remaining equations to solve the system.

Table 3.2. The 16 equilibrium reactions used in this model.

$H_2 \Leftrightarrow 2H$	(R1)	$CH_3+H \Leftrightarrow CH_4$	(R9)
$O_2 \Leftrightarrow 2O$	(R2)	$2CO+H_2 \Leftrightarrow C_2H_2+O_2$	(R10)
$N_2 \Leftrightarrow 2N$	(R3)	$C_2H_2+H_2 \Leftrightarrow C_2H_4$	(R11)
$H_2+O_2 \Leftrightarrow 2OH$	(R4)	$N+H_2 \Leftrightarrow NH_2$	(R12)
$O_2+N_2 \Leftrightarrow 2NO$	(R5)	$NH_2+H \Leftrightarrow NH_3$	(R13)
$2H_2+O_2 \Leftrightarrow 2H_2O$	(R6)	$CH_3+N \Leftrightarrow HCN+H_2$	(R14)
$2CO+O_2 \Leftrightarrow 2CO_2$	(R7)	$CO+H \Leftrightarrow HCO$	(R15)
$CO+2H_2 \Leftrightarrow CH_3+OH$	(R8)	$2CO+H \Leftrightarrow C_2H+O_2$	(R16)

3.2.2. EQUILIBRIUM CONSTANTS

For an arbitrary equilibrium reaction

$$aA + bB \Leftrightarrow cC + dD \tag{3.14}$$

the reaction constant, K_p, can be written as

$$K_p = \frac{(X_C)^c (X_D)^d}{(X_A)^a (X_B)^b} \left(\frac{P}{P_0} \right)^{(c+d-a-b)} \tag{3.15}$$

where X_i are the mole fractions, P and P_0 are the reaction and atmospheric pressure in atmospheres respectively. Because values of K_p are reported in the literature for $P_0 = 1$ atm, it can be removed from the equation. Realizing that $X_i = N_i / \Sigma N_i$ the K_p equation can be rewritten as

$$K_p = \frac{(N_C)^c (N_D)^d}{(N_A)^a (N_B)^b} \left(\frac{P}{\sum_1^i N_i} \right)^{(c+d-a-b)} \tag{3.16}$$

This form was used for the 16 equilibrium reactions providing 21 equations for the 21 unknowns in the global reaction. The set of 21 equations produced are non-linear and can be quite difficult to solve.

The value of K_p is a function of temperature. Curve fits for K_p are included in Reference [28] for the 7 reactions considered in that work. The fits for the remaining reactions were calculated using data from the JANAF thermochemical tables [30] and are included in the code. The same form of equation was used for all of the fits.

3.2.3. NEWTON-RAPHSON FOR NONLINEA R SYSTEMS

The Newton-Raphson method was used to find the solution to the equation set. An algorithm from *Numerical Recipes in C* [31] was modified for this equation set. The algorithm requires that the set of equations be written in the form $F_i(x_i) = 0$. For the species balances this is easily accomplished by moving everything to one side of the equation. For the equilibrium reactions one of the two forms shown in Equations 3.17 and 3.18 were used.

$$0 = (N_C)^c (N_D)^d \left(\frac{P}{\sum_1^i N_i} \right)^{(c+d-a-b)} - K_p (N_A)^a (N_B)^b \tag{3.17}$$

$$0 = (N_C)^c (N_D)^d - K_p (N_A)^a (N_B)^b \left(\frac{\sum_1^i N_i}{P} \right)^{(a+b-c-d)} \tag{3.18}$$

The reason for using these two forms will be discussed when describing the Jacobian matrix. The Newton-Raphson method solves for the array of values **x**, which make the matrix functions **F** equal to zero as shown in Equation 3.19. This is done by

first writing the matrix of functions $\mathbf{F}$ as a truncated Taylor's series expansion as shown in Equation 3.20.

$$\mathbf{F}(\mathbf{x}) = \mathbf{0} \tag{3.19}$$

$$\mathbf{F}(\mathbf{x}) = \mathbf{0} = \mathbf{F}(\mathbf{x}_0) + \mathbf{F}'(\mathbf{x}_0)(\mathbf{x} - \mathbf{x}_0) \tag{3.20}$$

$\mathbf{F}'(\mathbf{x}_0)$ represents the partial derivative of each of the 21 functions evaluated at the initially guessed values of N_i. These partials form a Jacobian matrix $\mathbf{J}(\mathbf{x_0})$. Rearranging the expansion and substituting $\delta\mathbf{x} = (\mathbf{x}\text{-}\mathbf{x}_0)$ gives the typical $\mathbf{A} \cdot \mathbf{x} = \mathbf{B}$ format in Equation 3.21.

$$\mathbf{J}\, d\mathbf{x} = -\mathbf{F}(\mathbf{x}_0) \tag{3.21}$$

This equation is solved using LU decomposition. From an initial guess of values for $\mathbf{x_0}$, a new set of values $\mathbf{x}$ are found using Equation 3.22 which should be closer to producing the desired result of $\mathbf{F}(\mathbf{x}) = 0$.

$$\mathbf{x}_{new} = \mathbf{x}_{old} + d\mathbf{x} \tag{3.22}$$

If $\mathbf{F}(\mathbf{x})$ does not equal zero, the new values of $\mathbf{x}$ replace the original guesses and the process is repeated.

An example of the Jacobian matrix used in the solution is shown in Equation 3.23. The matrix should be diagonally dominant and must have nonzero values along the diagonal. The equations were ordered to provide for these criteria. In some cases this was facilitated by the order in which the function was written and thus, both of the arrangements shown in Equations 3.17 and 3.18 above were used.

$$\mathbf{J} = \begin{vmatrix} \frac{\partial F_1}{\partial N_1} & \frac{\partial F_1}{\partial N_2} \cdots \frac{\partial F_1}{\partial N_{21}} \\ \frac{\partial F_2}{\partial N_1} & \\ \vdots & \\ \frac{\partial F_{21}}{\partial N_1} & \frac{\partial F_{21}}{\partial N_2} \cdots \frac{\partial F_{21}}{\partial N_{21}} \end{vmatrix} \tag{3.23}$$

3.2.4. CONVERGENCE METHODS

Because equilibrium problems can be difficult to solve, additional methods were used to make convergence more robust. To begin with, an initial guess near the correct solution helps to avoid a solution at a local minima and helps to reduce computation time. Olikara and Borman in Reference [28] suggested beginning with the simplified system given by Equation 3.24.

$$C_n H_m O_l N_k + \frac{n + m/4 - l/2}{f}\left(O_2 + 3.7274N_2 + .0444Ar\right) \Rightarrow N_4 H_2 + N_6 CO + N_8 O_2 + N_9 H_2 O + N_{10} CO_2 + N_{11} N_2 + N_{12} Ar \tag{3.24}$$

Olikara and Borman demonstrate an iterative technique for solving the seven equations and seven unknown species in this reaction and then show how these results can be used in combination with the equilibrium constants through back substitutionto obtain the remaining unknowns.

This method works for equivalence ratios in the range of approximately 1-2. For this work a guess was made at an equivalence ratio of 1.5 for all cases. The program then steps through a series of equivalence ratios updating the initial guess at each solution until it reaches the desired equivalence ratio. At each step the program tests the solution

to determine if the resulting species are all positive. If they are not it changes the step size of equivalence ratio and recalculates a result.

Another feature implemented to improve convergence is called line searches and backtracking. This method is used within the Newton-Raphson algorithm. A function f is defined as:

$$f = \frac{1}{2}\mathbf{F}\,\mathbf{F} \tag{3.25}$$

where **F** is the set of functions to be solved. A full Newton step does not insure a decrease in f, but a small enough step in the same direction will decrease f. The initial guess is first used to calculate f. A full Newton step is taken and f is recalculated. If f is not reduced in a full Newton step then a smaller step is taken as indicated in Equation 3.26.

$$\mathbf{x}_{new} = \mathbf{x}_{old} + ?\,d\mathbf{x} \qquad 0 < ? \le 1 \tag{3.26}$$

The value of λ is determined by backtracking in the direction of the full Newton step until λ reduces f sufficiently. Once a value for λ is successful at reducing f, the algorithm is ready to proceed to the next iteration. This guarantees that the new solution is closer than the old. Using this method, solutions for all of the conditions encountered in this study were obtained.

4. CYCLE SIMULATION DESCRIPTION

4.1. INTRODUCTION

Because the diesel combustion process is so complex, it is difficult to avoid complexity in the model. A useful model should be able to predict all of the processes involved in the cycle. In order to accomplish this with a reasonable model simplifying assumptions must be made. This chapter will attempt to describe the cycle simulation program. The theoretical basis for the models will be discussed and the assumptions will be explained. An overview of the program will be first, followed by a detailed description of the models used to describe the engine cycle. A complete listing of the code can be found in Appendix A.

4.2. OVERVIEW

The cycle simulation program models the compression, combustion, and expansion processes in the engine. During these processes the properties in the cylinder are calculated each ¼ degree of crank angle rotation. The residual gas temperature is factored into the initial conditions but the gas exchange processes are not modeled. The compression and expansion processes are easily modeled. The temperature and pressure at each step are calculated using an isentropic process. The composition of the gasses are also determined during the expansion process. Heat transfer is accounted for throughout the closed portion of the cycle. The five-zone model introduced in Section 2.5 is used to

describe events during the fuel injection and combustion phase. The spray geometry used in the model is from the correlation discussed in Chapter 3. This model outputs spray geometry parameters such as the liquid length, the spray penetration length, and equivalence ratio. The flame lift-off length is also incorporated into the program. In addition to elements of the model described previously the ignition delay, lift-off length, heat release, heat transfer, and zonal temperatures are calculated in the model as will be described below

4.3. DETAILED DESCRIPTION

This description will discuss the equations and individual models that make-up the cycle simulation program. No attempt will be made to describe the mechanics of coding the information, but rather, the description will detail the models and ideas used in the program. Some cylinder geometry will come first followed by a discussion of the compression and expansion processes. The fuel injection and combustion phase will be described next.

4.3.1. CYLINDER GEOMETRY

The volume in the cylinder can be easily calculated at any crank angle using the slider crank formula in Equation 4.1.

$$V = V_c + \frac{p B^2}{4}(l + a - s) \tag{4.1}$$

In this equation V_c is the clearance volume, B is the bore, l is the connecting rod length, a is the crank radius, and s is given by

$$s = a\cos? + \left(l^2 - a^2 \sin^2 ?\right)^{1/2} \tag{4.2}$$

To calculate the heat transfer, the surface area in the cylinder is also calculated, using Equation 4.3.

$$A = A_h + A_p + p\,B\left(l + a - s\right) \tag{4.3}$$

A_h is the cylinder head area and A_p is the piston area. These are not the same due to the bowl shape in a typical diesel piston.

4.3.2. COMPRESSION AND EXPANSION STROKE

Defining top dead center during the gas exchange processes as 0 degrees, the compression stroke begins at 180 degrees. The initial conditions in the cylinder at this time are close to the conditions in the intake. The intake temperature and pressure are read in from an input file along with the displacement volume. The intake air composition is 21% oxygen, 78% nitrogen and 1% argon. The mass in the cylinder is calculated using the ideal gas law. The initial enthalpy and C_p are calculated using curve fits. Subtracting Pv from the enthalpy calculates the internal energy. This defines the state at the start of the compression process. The model then steps through the cycle in ¼ crank angle increments.

An isentropic process calculates the cylinder conditions throughout the compression and expansion processes. For the compression stroke the composition in the cylinder is air. During the expansion stroke the composition is air and ideal products of combustion. The mole fractions of product gasses are calculated based on the overall equivalence ratio. The difference in composition, which is used to obtain the thermodynamic property $k = C_p/C_v$, is the only distinction between the compression and expansion process.

An isentropic process, shown in Equations 4.4 and 4.5, is used to calculate the pressure and temperature at each cylinder volume as calculated from Equation 4.1.

$$T_{i+1} = T_i\left(\frac{V_i}{V_{i+1}}\right)^{k-1} \tag{4.4}$$

$$P_{i+1} = P_i\left(\frac{V_i}{V_{i+1}}\right)^{k} \tag{4.5}$$

In these equations k is the specific heat ratio calculated as

$$k = \frac{C_p}{C_p - R_u} \tag{4.6}$$

The internal energy at each step in the isentropic process can be calculated in two ways. The internal energy at the start of compression is known from the initial conditions. The internal energy at subsequent steps can be calculated for each new temperature. Alternatively the internal energy can be calculated by adding the $P\Delta V$ work done in each step to the internal energy of the previous step. Both methods were used to verify the energy calculation.

Woschni's correlation, found in ref. [1] was used to calculate the heat transfer coefficient. This correlation was used during the entire cycle – compression, fuel injection and combustion, and expansion. The correlation is shown in Equation 4.7 where B equals the cylinder bore, and T and P are the in-cylinder temperature and pressure. The average velocity of in-cylinder gas, w, is given by Equation 4.8.

$$h_c = 3.26\,B^{-0.2}\,P^{0.8}\,T^{-0.55}\,w^{0.8} \tag{4.7}$$

$$w = \left[C_1\bar{S}_p + C_2\frac{V_d\,T_r}{P_r\,V_r}(P - P_m)\right] \tag{4.8}$$

In equation 4.8, C_1 and C_2 are constants. C_1 is always equal to 2.28. C_2 is equal to 0 during the compression stroke and equal to .00324 during combustion and expansion. $\overline{S}_p$ is the mean piston speed and V_d is the displacement volume. T_r, P_r, and V_r are the temperature, pressure, and volume in the cylinder at a reference state. The reference state used is the cylinder condition at the start of injection. P is the cylinder pressure and P_m is the motored pressure at the same crank angle. P_m was calculated using an isentropic process for the combustion and expansion processes without including combustion effects.

The heat transfer coefficient can now be used to calculate the total heat transfer for each time step as shown in Equation. 4.9.

$$Q_{out} = h_c A(T - T_{wall})?t \tag{4.9}$$

Q_{out} is the energy removed from the cylinder due to heat transfer. The total area in the cylinder is A. T is the average temperature in the cylinder and T_{wall} is the temperature of the cylinder surface. T_{wall} was chosen to be 600 K based on published data. The time for ¼ crank angle step is Δt. The heat lost reduces the internal energy in the cylinder at the end of each time step where the temperatures and pressures are recalculated assuming an ideal gas and the calculated specific heat. The nature of this model requires it to be adjusted to individual engines. This is accomplished by multiplying the heat transfer coefficient by a constant to produce the proper total heat release.

The simulation begins with a cylinder full of air at the intake temperature and pressure but this does not account for residual gasses that are left in the clearance volume from the previous cycle. In order to improve the initial condition used, the residual mass

and temperature are calculated at the end of the cycle and used to produce an improved initial temperature and pressure by assuming that the residual mass in the clearance volume is mixed with the incoming mass at the intake temperature and pressure in the displacement volume. Using the new initial condition the cycle simulation is repeated. The process continues until there is no change in the initial condition.

The temperature of the residual mass is calculated assuming the mass in the cylinder at the end of expansion is allowed to expand isentropically to the exhaust pressure as described by Equation 4.10. The subscript 540 refers to the crank angle at the end of expansion and P_{ex} is the pressure in the exhaust manifold. Although residual gas temperature calculations are done, the initial composition of the gas in the cylinder remains that of air.

$$T_{res} = T_{540}\left(\frac{P_{ex}}{P_{540}}\right)^{\frac{k-1}{k}} \tag{4.10}$$

4.3.3. HEAT RELEASE MODEL

The fuel injection and combustion portion of the cycle is the most challenging to model. The energy balance in the cylinder is shown in Equation 4.11. It should be noted here that the reference state for enthalpy calculations is 298 K. In the following equations $h_T = h_T - h_{298}$ which represents only the sensible enthalpy.

$$\frac{dU}{dt} = \dot{m}_f\left(h_{Tinj} - h_{fg}\right) - P\frac{dV}{dt} - \dot{Q}_{HT} + \dot{Q}_{HR} \tag{4.11}$$

The boundary work term can be easily calculated using a b ackward difference integration. Pressure is known at the beginning of every step and the volume can be calculated at any time (i.e. crank angle) using Equation 4.1.

The mass flow rate of fuel $\dot{m}$ is calculated using the injection velocity, U_f, given in the spray model (Equation. 3.8), and injector parameters using Equation 4.12.

$$\dot{m}_f = C_a A_{inj} ?_f U_f \tag{4.12}$$

In this equation A_{inj} is the nozzle hole area in the injector, C_a is the area contraction coefficient, and ρ_f is the fuel density. The area contraction coefficient must be measured for a given fuel injector and orifice to produce an accurate prediction of the flow rate and consequently injection duration. The accuracy of the model in predicting the spray penetration, which is subsequently used for heat release, is largely dependent upon this area contraction coefficient. Often the published literature for an engine only supplies the orifice diameter, and the peak injection pressure is only estimated. When this is the case, uncertainty in the injection duration is high. Under well-characterized conditions, the injection duration, injection pressure history, and contraction coefficient are known and the problem is actually over specified. The model will be compared with data from both well-characterized and poorly characterized injectors to investigate the associated error.

After the start of injection and prior to the start of combustion, the apparen t heat released from the fuel is negative due to evaporative cooling. This is due to the heat of vaporization of the fuel, h_{fg}, in Equation 4.11. Evaporation reduces the sensible internal energy producing a dip in the apparent heat release determined fro m pressure measurement just before the start of combustion that is seen in the data. The enthalpy of vaporization continues throughout the injection event but is not obvious after combustion due to the large amounts of heat release from the fuel burning.

Heat transfer from the cylinder $\dot{Q}_{HT}$ is calculated using Equations 4.7 - 4.9 as in the compression and expansion strokes.

$\dot{Q}_{HR}$ in Equation 4.11 is the heat release rate from combustion. The basic principle of the heat release calculation is to assume that all of the oxygen entrained in the jet has reacted with the fuel to produce ideal stoichiometric combustion products and unburned fuel. At any period of time during injection, the spray correlation defines the boundary of the jet and the total mass of air entrained in the jet. The heat release is therefore mixing limited and proceeds only as fast as O_2 is entrained with the air into the jet. For the purposes of calculating the heat released from this reaction, the products are assumed to contain only CO_2, H_2O, N_2, and unburned fuel and they are assumed to be at 298 K. The total heat release at time t is calculated using Equation 4.13. Since the reference state is 298 K this only includes the heat of formation of each species. This is the heat of combustion or the lower heating value of the fuel Q_{LHV}. The heat release rate at any crank angle can then be calculated using Equation 4.14.

$$Q_{Tot,t} = H_{prod,298} - H_{react,298} \tag{4.13}$$

$$\dot{Q}_{HR} = Q_{Tot,t+\Delta t} - Q_{Tot,t} \tag{4.14}$$

It should be noted that the heat release calculation is uncoupled from the five-zone model and the temperatures in the zones. An attempt was made to couple the heat release but instability was produced in the code resulting in this simplification.

Unfortunately there are portions of the combustion process that are not limited by the air entrainment rate, which complicated the calculation of the heat release rate. These exceptions and the method selected for handling them are discussed below.

First, air entrained within the liquid length section of the jet cannot react with the fuel because the mixture is not at a combustible temperature. However, the fuel along the outer surface does burn back to the lift-off length, which is generally upstream of the liquid length. Also, fuel entrained beyond the flame length or length of the jet where the equivalence ratio is greater than 1 also cannot contribute to combustion because the fuel is already burned in this region. Second, the fuel jet initially penetrates well beyond the liquid length before auto ignition occurs and combustion begins. This ignition delay time must be determined before heat release can begin. This delay in combustion would produce a large spike in heat release if all of the fuel were allowed to burn instantaneously after the delay period. Therefore, a sub-model for determining the burn rate of the initial premixed reactants was developed which spreads the combustion out over the appropriate period of time.

Two factors affect ignition delay in engines, a mixing time and a kinetic time. The mixing time arises from the need of the fuel to vaporize before it can ignite. Assuming the jet develops as modeled, the time delay for mixing was determined by the time to reach the liquid length, t_{ll}. The kinetic delay time was calculated using an exponential function as shown in Equation 4.15.

$$t_k = \exp\left(-C_{ig} T_{2a}\right) \tag{4.15}$$

The constant C_{ig} was determined to be .012 by fitting data at the 1200-RPM, low fuel load, normal injection timing condition in the Sandia engine. The temperature used, T_{2a}, is the temperature of vaporized fuel at the end of the liquid length or the temperature at the boundary of the liquid length and zone 2 representing the evaporated fuel. The method for determining this temperature is discussed later with zone 2 calculations.

Ignition delay was set equal to the sum of the mixing delay and the kinetic delay as shown in Equation 4.16.

$$t_{id} = t_{ll} + t_k \tag{4.16}$$

Once ignition takes place heat release begins. Figure 4.1 shows the jet at an arbitrary time after ignition to illustrate the heat release calculations.

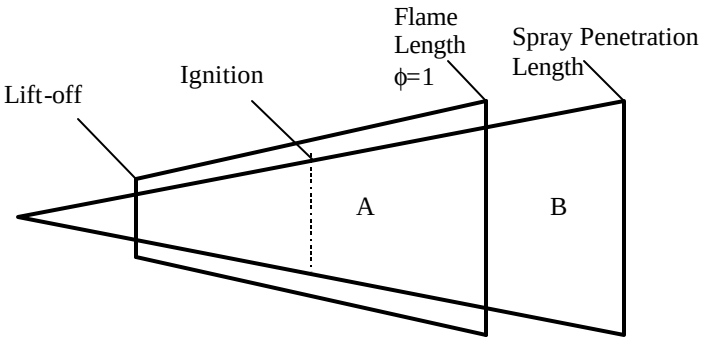


Figure 4.1. Spray Geometry used for Heat Release Calculations.

Region A is fuel rich and includes everything beyond the lift-off length up to the flame length. Region B includes everything beyond the flame length and is lean. The total heat release at any time becomes Equation 4.17.

$$Q_{Tot,t} = Q_A + Q_B \tag{4.17}$$

Q_A and Q_B are calculated using Equations 4.18 and 4.19.

$$Q_A = m_{air,A}\left(\frac{F}{A}\right)_s Q_{LHV} \tag{4.18}$$

$$Q_B = m_{fuel,B} Q_{LHV} \tag{4.19}$$

The mass of air in region A is given by the volume in the region determined from the spray model and the density using Equation 4.20. The density used is the density at the start of injection. The mass of fuel in region B is determined using the time calculations from the spray model and the fuel flow rate. In Equation 4.21 t_{sp} is the time to the spray penetration length and t_{fl} is the time to the flame length.

$$m_{air,A} = ?_{air,inj} V_A \tag{4.20}$$

$$m_{fuel,B} = (t_{sp} - t_{fl})\dot{m}_f \tag{4.21}$$

Once the heat released is calculated the heat release rate can be calculated using Equation 4.14.

After the start of combustion region A grows outward as the spray continues to penetrate into the cylinder until reaching the flame length. At this point region B forms. Region A also grows toward the injector initially until it reaches the lift-off length. The time required between the start of injection and the start of combustion when a quasi-steady reaction zone is set up is the premixed burn duration. There is no clear data on how fast the premixed burn will occur. An estimate was made using a global kinetic reaction rate for $C_{10}H_{22}$. Details of this can be found in Appendix B. The Appendix shows how a premixed burn time and a shape for the rate of heat release are determined for the premixed burn.

Region A will begin to disappear after the end of injection. The back end of the spray is assumed to move at the same velocity as predicted by the spray penetration model for the leading edge. Once the back end of the spray reaches the flame length all of the heat has been released from the fuel and the model begins the expansion stroke.

Using the description above to calculate the heat release rate, $\dot{Q}_{HR}$, the right-hand side of Equation 4.11 is known allowing integration to determine the internal energy "U" at each time step (1/4 crank angle). The internal energy in the cylinder is used to calculate the cylinder temperature using an iterative method. Since internal energy is enthalpy minus PV the enthalpy curve fits are used. The cylinder pressure is then calculated using the ideal gas law.

The heat release produced from this model can be compared with data taken from an engine. The goal now is to produce the 5-zone model to determine temperatures in the zones. Since pollutant formation is very dependent on temperature this information is useful in understanding where and how the pollutants form.

4.3.4. FIVE-ZONE MODEL

The spray geometry and the five zones seen in Figure 4.2 are calculated at each ¼ crank angle after the start of injection. Information provided by the spray model includes: the penetration length, spread angle, liquid length, and fuel air equivalence ratio as a function of spray length. The spray model assumes the spray evolves in a constant temperature, pressure, and density environment. This is obviously not the case in the engine but the only available spray correlation upon which the model was based was developed for non-reacting (no combustion) conditions. It is one of the objectives of this work to determine the accuracy of applying this non-reacting spray model to results from a reacting spray.

Figure 4.2 shows a quasi-steady jet where each of the 5 zones has developed. The lift-off length is set arbitrarily in the input file and read into the code. Since the diffusion flame surrounds the jet from the lift-off length outward there are combustion products

entrained in zones 1 and 2. However this does not mean that zones 1 and 2 no longer exist past the lift-off length. There is still unburned fuel and entrained air in these zones due to the rich mixture and the zones are still differentiated by the fuel being liquid or vapor phase.

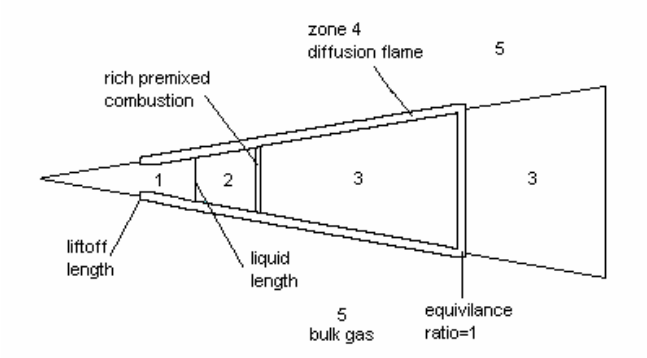


Figure 4.2. Five zone spray model

The size and shape of the zones shown in Figure 4.2 can all be determined from the spray model, ignition delay model, and assigned lift-off length. Using the geometry from the spray model the volume of each zone can be determined and the mass of fuel in each zone can be calculated from the correlation given in Equation 3.8. The density in the cylinder is multiplied by the volume of each zone to obtain the mass of air entrained in each zone. A description of how temperature and enthalpy in each zone is calculated is given in the following paragraphs.

At the start of injection zone 1 begins to form. Zone 1 is fully formed when the spray reaches the liquid length. After the start of combustion the lift-off length is allowed

to move from the border between zones 2 and 3 to the position given in the input file. This causes a shortening of the liquid length because high temperature products are entrained that help evaporate the fuel. The composition of this zone is liquid fuel and entrained air, beyond the lift-off length entrained products are also added. The temperature in this zone is defined to be the boiling temperature of the fuel since the temperature is approximately constant during the fuel vaporization process. The enthalpy is calculated using the boiling temperature and the fuel, air, and products composition.

Zone 2 begins to form once the liquid length is reached. It comprises the rest of the fuel jet until the start of combustion. At the start of combustion, or ignition delay time, zone 2 begins to react and continues until ¾ of the initial mass is consumed. This is accomplished using the exponential decay function in Equation 4.22 at each crank angle step until the minimum mass is reached.

$$m_{2i+1} = m_{2i} - \left(m_{2i} - m_{2min}\right)\exp\left(\frac{-E_A}{R_u\, T_{boil}}\right) \tag{4.22}$$

Here m_{2i+1} and m_{2i} are the new and old mass respectively within zone 2 for a given ¼ CA step. m_{2min} is the minimum mass in zone 2. E_A is the same activation energy used in the ignition delay model and T_{boil} is the boiling temperature of the fuel. Once the mass in zone 2 reaches the minimum, the position of the division between zone 2 and zone 3 remains constant until the end of injection. This zone is made up of vaporized fuel and entrained air. When the premixed burn begins and the flame sheath moves back to the lift-off length products are entrained instead of air.

Consuming ¾ of the initial mass in zone 2 and leaving ¼ as the minimum mass is arbitrary. There is no good data indicating where or even if a well-defined boundary exists between the premixed vapor and the products. If the zone 2 / zone 3 boundary is

moved closer to the nozzle, the equivalence ratio and flame temperatures are such that the equilibrium model has difficulty finding a solution also indicating that the fuel would not readily react. As additional information is made available the model can be improved in this area.

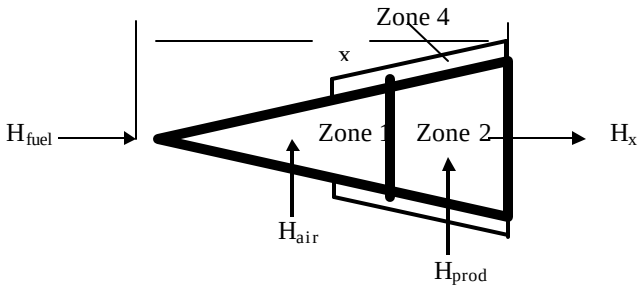


Figure 4.3. Zone 2 Energy Balance

The temperature variation across zone 2 can be calculated using an energy balance. Figure 4.3 illustrates the energy balance at a distance, x, from the nozzle. The entire spray from the nozzle to x is the control volume. Assuming the spray is steady to position x and there is no heat transfer the first law reduces to $H_{in}=H_{out}$. Enthalpy from the fuel, air and products of the flame sheath are entering and enthalpy is leaving at distance x from the nozzle. Since enthalpy entering equals enthalpy exiting the energy balance becomes Equation 4.23.

$$H_x = H_{fuel} + H_{air} + H_{prod} \tag{4.23}$$

The absolute sensible enthalpy is evaluated using $H = mC_p\Delta T$ and $\Delta T = T\text{-}298K$ with 298 K as a reference temperature. Substituting into Equation 4.23 and solving for temperature yields Equation 4.24.

$$T_x = \frac{m_{fuel}\left(-h_{fg} + C_{p,fuel}T_{boil}\right) + m_{air}C_{p,air}T_{air} + m_{prod}C_{p,prod}T_{prod}}{m_x C_{p,x}} \qquad (4.24)$$

Since the fuel is injected in liquid phase the heat of vaporization, h_{fg}, must be included. The temperature of the air in this equation is the zone 5 temperature, and T_{prod} is the adiabatic flame temperature calculated in zone 4. The mass of the unburned air and the burned air is calculated using the density at the start of injection and the air volume

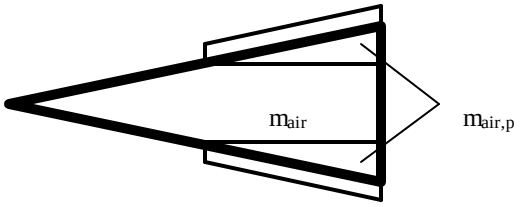


Figure 4.4. Masses Used in Zone-2 Temperature Calculations

shown in Figure 4.4. The mass of the products is the mass of the air and the mass of the fuel calculated using Equation 4.25.

$$m_{prod} = m_{air,p} + m_{air,p}\left(\frac{F}{A}\right)_s \qquad (4.25)$$

T_x is found at 1/4 mm increments throughout zone 2. The temperature at each x is multiplied by the mass in an axial cross section of the spray between steps. These are

added up and divided by the total mass in zone 2 to produce a mass averaged temperature. This temperature and the overall composition are used to calculate the total enthalpy in the zone.

At the start of combustion zone 3 begins to form. While zone 2 is shrinking, zone 3 grows toward the injector as well as outward with the spray tip. After zone 2 reaches its minimum, zone 3 continues to grow only at the spray tip as penetration continues. This zone is made up of products of combustion.

The product composition and the temperature in this zone are determined using the equilibrium model and an adiabatic flame calculation. This zone is divided into slices as shown in Figure 4.4 allowing an axial determination of the zonal temperature and product composition. The stoichiometry of each slice is a function of axial position as given by Equation 3.10 and is assumed uniform in the radial direction. The adiabatic flame temperature must be found by an iterative approach. The enthalpy of the reactants in each step is calculated from the stoichiometry. The temperature of a step is guessed until the equilibrium program calculates a matching enthalpy for the products of combustion. This gives a temperature for each slice. The overall temperature is calculated as a mass average and enthalpy in the zone is calculated using curve fits.

At some axial distance in the downstream direction of the jet, the correlation predicts an average equivalence ratio of one. The distance between this axial location and the injector nozzle is called the flame length. This is the axial position where the flame will be positioned due to the stoichiometry being the most favorable for reaction. Beyond the flame length the jet entrains air and therefore becomes more dilute of

products decreasing the temperature. Temperatures and enthalpies in zone 3 beyond the flame length are calculated as discussed for the rest of zone 3 above.

Zone 4 also develops at the start of combustion. It is a massless zone representing the diffusion flame sheath surrounding the spray. The flame envelope initially surrounds zones 2 and 3 during the initial premixed burn period. The flame envelope continues toward the injector and also surrounds a portion of zone 1 beyond the lift-off length but does not move all the way back to the nozzle tip. Zone 4 is characterized by the adiabatic flame temperature that is produced at an assumed equivalence ratio of 1 on the boundary between the fuel rich jet (zones 2 and 3) and the surrounding air (zone 5). This adiabatic flame temperature is calculated using the equilibrium code to determine the species, the enthalpy curve fits, and an iterative technique that finds the temperature at which the enthalpy of the products equals the enthalpy of the reactants.

Zone 5 is the surrounding bulk gas. Its composition is always air and its mass is calculated by subtracting the mass of air entrained in the spray from the total air mass in the cylinder.

The internal energy in zone 5 is calculated by subtracting the internal energy in the other three zones within the fuel jet from the total cylinder internal energy given in Equation 4.11. The internal energy in zone 5 is used to determine the temperature in zone 5, again using an iterative method with the enthalpy curve fits.

Following this method the state in the cylinder can be determined at any time during fuel injection. The composition and temperature in each zone of the model can also be calculated at any time.

At the end of injection, all of the zones begin to be consumed as the combustion process proceeds. The last parcel of fuel injected is assumed to penetrate according to the jet correlation in the same way that the previous fuel entered. Therefore the spray model is also used to determine the position of the back end of the fuel jet. As the back end of the jet moves across zone 1 the mass in the zone decreases until the zone is consumed when the end of the spray reaches the liquid length. Similarly, the end of the jet moves through zone 2 and zone 3 until the mass in each zone is zero.

Finally, diesel fuel injectors typically produce between four and eight fuel jets for each injector that proceed radially outward from a centrally located injector. We assume symmetry for the jets and their location in the cylinder and therefore model only a single jet and the wedge shaped section of the combustion chamber occupied by a single jet. The above discussion describes calculations done for one wedge and, consequently for one fuel jet.

In Chapter 6 results of the model will be compared with data taken from a constant volume combustion vessel and two diesel engines.

5. EXPERIMENTAL SETUP

Experimental data was used to evaluate the accuracy of the model and suggest areas for future improvement. Data from the Cummins single-cylinder research engine at BYU was used as well as data from the optically accessible engine and constant volume combustion vessel at Sandia National Labs in California. The experimental setup of these apparatus and their operating conditions will be explained in this chapter. The method for processing the data will also be explained. The inputs to the model required for a prediction of each set of experimental results will be discussed.

5.1. SANDIA BOMB

The diesel combustion bomb at Sandia is a constant volume chamber capable of producing typical top-dead-center cylinder conditions for modern diesel engines. The chamber is approximately a cube of 1.2-liter volume with each side measuring 108 mm. Each wall of the cube has a 105mm diameter opening where a window or other device may be located. One opening was filled with a metal plug that housed the injector. Another was filled with a plug that contained two spark plugs and a mixing fan. The remaining openings were filled with sapphire windows having a 102 mm diameter aperture that allowed almost complete optical access to the diesel spray.

There are ports in each of the eight corners of the cube measuring 19 mm in diameter. Three of these contain valves, two of which are for intake and one for exhaust.

One contains a thermocouple that provides information on the state in the chamber prior to an injection event. A piezoelectric crystal type pressure transducer provides a pressure trace for the combustion event. The rest of the openings were plugged for the experiments reported here.

The Sandia injector is a prototype built by Detroit Diesel. It is a common rail type injector that is activated with an electronically controlled solenoid. The tip of the injector has one opening that injects fuel directly into the center of the bomb. Fuel is supplied to the injector by an accumulator that is pressurized using a high-pressure pump. The pressure line into the injector is instrumented with another piezoelectric pressure transducer to acquire injection pressures. The needle is instrumented with a Hall effect sensor to detect needle lift giving start of injection and injection duration. Fuel injection is also detected optically using an extinction measurement from a HeNe laser.

Control of the bomb and the acquisition of the data are handled with two PC's that are networked together. More detailed information on the bomb can be found in Reference [4]. Parameters describing the Sandia bomb and the conditions used in this work are found in Tables 5.1 and 5.2 respectively.

In order to reach diesel operating temperatures and pressures in the constant volume combustion vessel a lean mixture of combustible gasses was burned. The spark plugs were used to ignite this mixture. The proportions of the mixture were such that after the reaction 21 % oxygen remained in the hot products simulating a compression process. The mixing fan insures a homogeneous mixture. After this simulated compression, the injector is activated at a prescribed time to produce a diesel injection into the chamber.

Table 5.1. Sandia bomb parameters

Side Length	108 mm
Side Opening Diameter	105 mm
Window Diameter	102 mm
Volume	1.2 liters
Injector Nozzle Size	246, 180, 100, 71 µm
Number of Nozzles	1

Table 5.2. Sandia bomb operating conditions

Run	Tin (K)	Pin (kPa)	Pinj (MPa)	Nozzle Dia µm
1	1000	4247	140	246
2	1000	4247	143	180
3	1000	4247	143	100
4	1000	4247	143	71

It is important to note that the constant volume combustion chamber has a longer distance between the injector nozzle and the opposite wall than is usually the case in a real engine. The jet can travel 102 mm in the combustion vessel before wall impingement while the range of travel in the two engines also described in the chapter are approximately 13 and 25 mm which are typical of the range found in commercially available small and large truck diesel engines respectively. The independent variable being changed in the combustion vessel experiments is the nozzle diameter. The two larger diameter jets have significant wall impingement while the two smaller diameter jets reach their flame lengths before the wall is reached. The ability of the combustion vessel to observe reacting jets without wall interaction is valuable in evaluating the model even though it is not realistic.

5.2. SANDIA ENGINE

The Sandia research engine is based on a Cummins N-series engine. The engines dimensions remain largely stock and are listed in Table 5.3. The piston has been modified to allow optical access, as has the cylinder wall. Because of this the compression ratio is lower than stock. The intake air is heated to 426 K and pressurized to 192 kPa to provide typical diesel conditions (1000 K and a density of 16.6 kg/m^3) after compression. The cylinder head also has a window replacing one of the exhaust valves. The injector used is a Cummins CELECT unit type injector. The injector is instrumented for needle lift providing injection-timing data. This engine is attached to a dynamometer that is used to keep the engine at a constant speed. In order to accommodate optical access by keeping the windows clean and to synchronize the laser with fired engine events, the engine is only fired once every 20 cycles. A more detailed description of this engine can be found in Dec [2] and Siebers [4].

Table 5.3. Sandia engine parameters.

Bore	.1397 m
Stroke	.1524 m
Compression Ratio	10.75
Connecting Rod Length	.3048 m
Piston Area	.024 m^2
Injector Nozzle Size	194 µm
Number of Nozzles	8

Ensemble averaged pressure data were received for this engine at several operating conditions from which heat release was calculated. Data are included for 1200 and 1680 RPM conditions as well as a normal and a retarded injection timing. There are

also two fuel loads for the 1200 RPM condition, while the 1680 RPM condition only includes the high fuel load. The data sets are ensemble averages of 20 cycles. Table 5.4 gives the speed, load, timing, intake pressure, and intake temperature for each of the conditions for which data was received from Sandia.

Table 5.4. Sandia engine running conditions.

Run	RPM	Fuel Load (kg)	Injection Timing (ºBTC)	Pin (kPa)	Tin (K)
1	1200	.00007189	11.5	192	426
2	1200	.00007189	0	192	426
3	1200	.000138	11.5	192	426
4	1200	.000138	0	192	426
5	1680	.000138	11	192	426
6	1680	.000138	1	192	426

5.3. BYU DIESEL ENGINE TEST CELL

The BYU engine will be described in three sections. First the engine itself and the dynamometer will be described. Then the instrumentation and data acquisition will be explained. Last, the experimental conditions used in this work will be described.

5.3.1. ENGINE AND DYNAMOMETER

The research engine at BYU is a single-cylinder, 0.96 L, direct-injection, turbocharged diesel modified from a Cummins B-series 6-cylinder, 5.9 L engine. Holes were cut in five of the pistons eliminating compression and expansion for those cylinders. Valves for those five cylinders were deactivated so that only a single cylinder remained operational. Weight was added to the five modified pistons so the engine would remain

balanced. The injector pump was modified to provide fuel to only the operating cylinder. The injection system is a standard common rail type system. A micrometer attached to the fuel rail gives repeatable fuel load settings. A compressor and tank provide clean and dried air that can be regulated to the desired intake pressure for the engine. The pressure is typically elevated to simulate turbo charging. Engine and injector specifications are included in Table 5.5. The engine is connected to a 150 horsepower motoring dynamometer. The dynamometer can be used to power or load the engine.

Table 5.5. BYU engine parameters

Bore	.102 m
Stroke	.120 m
Compression Ratio	17.9
Connecting Rod Length	.192 m
Piston Area	.014 m^2
Injector Nozzle Size	250 µm
Number of Nozzles	5

5.3.2. INSTRUMENTATION AND DATA ACQUISITION

The air and fuel lines into the engine are instrumented to provide measured flow rates. The airflow was measured using an Omega (Model FV-510B-D) flow meter. This instrument reads a volume flow rate. The intake line is also instrumented with a pressure gauge and a thermocouple that are used with the volume flow rate to provide mass flow. The intake pressure and temperature were also measured near the engine and were assumed to be equal to the cylinder conditions at the start of the compression process. The fuel flow was measured using a Micro Motion (Model D6) mass flow meter that uses

the Coriolis acceleration of the fuel as it passes through a vibrating tube to determine the mass flow. Using these two measurements for fuel and air allow the overall equivalence ratio to be determined. A thermocouple in the exhaust was used to determine when the engine reaches steady state and can be compared with model temperatures at the end of the expansion process.

The cylinder head has been drilled to accommodate a pressure transducer. A piezoelectric crystal type transducer was used (AVL Model QC32C). The pressure data was acquired using a PC with LabVIEW. A 2-channel encoder is attached to the crank. One channel provides a pulse at top dead center. This pulse tells the computer when to start taking pressure data. The other channel gives a pulse every ¼ crank angle. The voltage from the pressure transducer is recorded at each pulse. This provides 2880 data points for each cycle. The LabVIEW program has been written to process the data and produce an ensemble average. To get a reasonably smooth pressure trace, 100 cycles are averaged.

5.3.3. BYU ENGINE EXPERIMENTAL CONDITIONS

Data was taken at four different engine conditions. Overall nominal equivalence ratios of .3 and .5 were used. The engine tests were run at 1500 and 2000 RPM for each equivalence ratio. The fuel load and the dynamometer force were adjusted to reach these conditions. The intake air was held at a constant guage pressure of 103.4 kPa (approximately 189.6 kPa absolute) for all tests. Fuel and airflow measurements were recorded along with intake air pressures and temperatures. The exhaust temperature and RPM were also recorded. These were used to provide inputs for the cycle simulation

model. The injection timing is not known on this engine. Five data sets of 100 averaged cycles were also recorded. Operating conditions for this engine are included in Table 5.6.

Table 5.6. BYU engine running conditions

Run	RPM	ϕ	Pin (kPa)	Tin (K)
1	1500	.3	189.6	304
2	1500	.5	189.6	307
3	2000	.3	189.6	304
4	2000	.5	189.6	306

5.4. DATA PROCESSING

The method of converting pressure data from each of the experimental facilities to heat release data will be discussed first for the constant volume combustion vessel and then for the constant volume combustion vessel and both engines combined.

The pressure history in the constant volume combustion vessel differs from an engine in that the compression process is produced not by mechanical work but through an initial premixed burn combustion process. After the initial combustion event, the bomb cools by heat transfer until reaching a predetermined pressure after which the injection event is triggered. Without fuel injection the pressure would continue to decrease in a smooth and repeatable monotonic fashion. The pressure data provided by Sandia is the difference between the vessel pressures with and without injection. This removes some of the effects of heat transfer from the data but since the flame is now hotter and radiates to the walls, heat transfer during the second or diesel combustion

event is higher than would have existed without the second combustion event and this additional heat transfer is not accounted for.

Sandia provided smoothed pressure data from the bomb in order to produce smooth heat release rates. The smoothing method was dependent on the temporal location of the data. A cubic spline fit was used to smooth the data up to the time when half of the pressure rise due to the premixed burn had taken place. After this point a filter that incorporates a fast Fourier transform was used. This method of smoothing the bomb data attempts to preserve the magnitude and temporal location of the premixed burn spike at the expense of an apparently noisy heat release before the premixed burn. The latter smoothing removes all high frequency fluctuations and would remove the premixed spike. The smoothing technique also adds a low frequency component apparent in the resulting heat release rates as will be shown. A more detailed discussion of method used to process the Sandia bomb data can be found in [32].

The engine data sets were smoothed using a Fourier series expansion (Eqns. 5.1-5.4). This is a different method than was used at Sandia. 200 terms were used in the series. The "points" variable is the number of data points in the set; 1440 for the Sandia engine and 2880 for the BYU engine. The crank angle resolution is $\Delta\theta$; .5 degrees for the Sandia engine and .25 for the BYU engine.

$$P(\theta_j) = \frac{a_0}{2} + \sum_{n=1}^{200} a_n \cos\left(\frac{n p \theta_j}{360}\right) + \sum_{n=1}^{200} b_n \sin\left(\frac{n p \theta_j}{360}\right) \tag{5.1}$$

$$a_0 = \frac{1}{360} \sum_{j=1}^{points} P_j \Delta\theta \tag{5.2}$$

$$a_n = \frac{1}{360} \sum_{j=1}^{points} P_j \cos\left(\frac{n p ?_j}{360}\right) \Delta ? \tag{5.3}$$

$$b_n = \frac{1}{360} \sum_{j=1}^{points} P_j \sin\left(\frac{n p ?_j}{360}\right) \Delta ? \tag{5.4}$$

A roll off function was used to reduce the influence of high frequency components by multiplying each term a_n and b_n in the Fourier series by a value between 0 and 1. The power function in Equation 5.5 was used calculate the multiplier m.

$$m = 1 - \left(\frac{n}{terms}\right)^8 \tag{5.5}$$

Once the pressure was smoothed, the apparent heat release rate was calculated using Equation 5.6.

$$\frac{dQ}{dT} = \frac{k}{k-1} P\,dV + \frac{1}{k-1} V\,dP \tag{5.6}$$

In this equation the specific heat ratio k was held constant at a value of 1.3. For the combustion bomb the first term in Equation 5.6 is zero because, dV, the change in volume is zero. Further inspection shows that dQ/dt is directly proportional to dP/dt, therefore dP/dt represents the rate of apparent heat release in the constant volume vessel and is used for heat release comparisons.

The heat release rate at each ¼ crank angle was numerically integrated to produce a total heat release. The apparent heat release rate (AHRR) and cumulative heat release were plotted as a function of time or crank angle. Log P – log V graphs were also produced for the engine pressure data.

5.5. MODEL RESULTS

An input file for the model was produced for each operating condition on the BYU engine, the Sandia Engine, and the Sandia Bomb. Input file decisions for each data set are discussed below.

5.5.1. SANDIA BOMB

To produce results for the constant volume combustion chamber the compression ratio was set to 1. The bore and stroke used in the input were set equal to each other and calculated to produce the volume of the bomb. The intake temperature and pressure, the injection pressure, and injection duration were set equal to measured values supplied with the data. The model requires an engine speed, which in this case is only relevant in producing a time step. In order to make time steps in the bomb model similar to those in the engine, a speed of 1200 RPM was selected. The heat transfer model was initially removed in the code since mo st of the heat transfer has been removed from the data. Heat transfer was then added and another data set produced. In this set the heat transfer was adjusted to produce matching peak pressures.

The mass of fuel injected was determined from the injection pressure and the injection duration using Equation 4.13. The actual mass injected was not supplied with the data. The input data for the Sadia Bomb results are in Table 5.7.

Table 5.7. Model conditions for Sandia bomb data

Case	Inj dia (μm)	Pinj (MPa)	fuel (mg)	T (K)	P (kPa)	B & S (m)	Rc	RPM	Heat Tran
1	246	140	.0850	1000	4247	.115	1	1200	6.5
2	180	143	.0295	1000	4247	.115	1	1200	3
3	100	143	.0120	1000	4247	.115	1	1200	.25
4	71	143	.0090	1000	4247	.115	1	1200	-

5.5.2. SANDIA ENGINE

The bore, stroke and compression ratio along with other geometry data are given in Table 5.3. The piston area, from Table 5.3, to be used in the heat transfer model was calculated using the bowl shape and measurements given in [2]. The intake air conditions were also given along with the mass of fuel injected, injection duration, and injector nozzle geometry. The injection pressure was calculated based on constant pressure injection using the measured duration and measured total fuel injected. The flame lift-off length used in the engine was 12 mm, which is reasonable based on published data.

Originally, cases were run assuming a constant injection pressure, but it was also recognized that the injection pressure might ramp up during the injection event. This injection rate was investigated as part of a parametric study using the model. For ramped injections the pressure profile increased linearly beginning at 15 MPa. In order to match injection duration data the ending pressure was the same as the constant pressure setting for each case. The conditions used for the Sandia engine are summarized in Table 5.8.

For all cases, both engines and the combustion bomb, the integrated apparent heat release was higher for the model than the engine. The total heat release in the model is

Table 5.8. Model conditions for Sandia engine data

Case	RPM	Tin (K)	Pin (kPa)	SOI (CA)	fuel (mg)	Pinjmax (MPa)	Pinjmin (MPa)	Heat Tran x
1	1200	429	192	348.5	.01789	40	15	12.3
2	1200	429	192	360	.01789	40	15	18
3	1200	429	192	348.5	.138	58	15	7.75
4	1200	429	192	360	.138	44	15	8.25
5	1680	429	192	349	.138	58	15	6.75
6	1680	429	192	360.5	.138	59	15	10.5

equal to the energy in the fuel ($m_f Q_{hv}$) minus the heat transfer given by Equation 4.7. The total predicted heat release was always too high, suggesting that the heat transfer model underpredicts the actual heat transfer. This is consistent with the expected accuracy of the heat transfer correlation as proposed which must be "tuned" or calibrated to each engine configuration. The model heat transfer was multiplied by a constant until the total heat release from the model was equal to the total heat release from the engine. The constant used for each case is included in Table 5.8.

The constant used in the ignition delay model (Eqn. 4.11) was determined using the 1200-RPM, low fuel load, normal injection timing condition. Using the injection pressure ramping model the value of the constant was chosen to match the start of the premixed burn spike. This constant was used for all cases in both engines and in the bomb. This is an admittedly simplified approach to correlating the ignition delay model to data but an improved ignition delay model is beyond the scope of this work.

5.5.3. BYU ENGINE

The geometry of the BYU engine was measured or taken from technical literature. The area of the piston in Table 5.5 was found using a piston that was cut in half. The

shape was plotted and the area calculated from the profile. The intake conditions and fuel mass injected were measured experimentally.

The injector nozzle was removed from the engine and the diameter was measured. The orifice coefficients, C_d and C_a, used in the spray model presented in Chapter 3 were not measured but were assumed to be those of a similar injector nozzle given in the Sandia data [4]. As with the Sandia engine the lift-off length was assumed to be 12 mm.

The start of injection and injection duration in the BYU engine are not known. Therefore, given the fuel mass injected, the start of injection and injection pressure were adjusted to give the best agreement with the data. These cases were run using constant injection pressure and a ramping pressure following the same procedure as the Sandia engine. The heat transfer model was also adjusted to produce the same total heat release as the measured data. The conditions used in the model for the BYU, single cylinder engine are included in Table 5.9.

Table 5.9. Model conditions for BYU engine data

Case	RPM	Tin (K)	Pin (kPa)	SOI (CA)	fuel (mg)	Pinjmax (MPa)	Pinjmin (MPa)	Heat Tran x
1	1500	304	189.6	354.5	.0439	30	15	3.25
2	1500	307	189.6	354	.0763	40	15	2.75
3	2000	304	189.6	359	.0430	55	15	3.25
4	2000	306	189.6	358.75	.0705	62	15	3.5

The results of all BYU engine cases are similar to the Sandia cases and will be compared in Chapter 6. Figures for all cases of both engines and the combustion bomb can be found in Appendix C. A sample input file can be found in Appendix A with the code listing.

6. RESULTS

This chapter will present the results of the spray model, equilibrium model, and cycle simulation program. In order to provide a context for the comparison of the cycle simulation results, a general explanation of heat release data will first be presented. Then the results of the cycle simulation model will be compared to data taken in the combustion bomb, Sandia engine, and BYU engine. Observations regarding similarities and differences between the model and experimental data will be made.

6.1. SPRAY MODEL RESULTS

Once the spray and liquid length models were completed they were compared with results from the papers from which they were taken [25], [27] in order to demonstrate they had been coded correctly. Dimensionless spray penetration (S) and equivalence ratio (ϕ) are plotted as a function of dimensionless time ($\tilde{t}$) on logarithmic scales in Figure 6.1. In this figure the model using the correlation developed in [25] is plotted. Comparing Figure 6.1 with Figure C.2 of Naber and Siebers [25] shows that the spray model and equivalence ratio are being calculated correctly.

Calculated liquid lengths were also compared with experimental data for diesel fuel and heptamethylnonane as reported by Higgins et al [27]. The liquid length was calculated for densities of 7.3, 14.8, and 30.0 kg/m^3 and at temperatures between 700 and 1300 K. The results are shown in Figures 6.2 and 6.3 where the solid lines represent the model

results and the symbols are data taken from the published literature [27]. The results show very good agreement especially in the temperature ranges (≈1000 K) and densities (≈15 kg/m^3) typical of diesel engines.

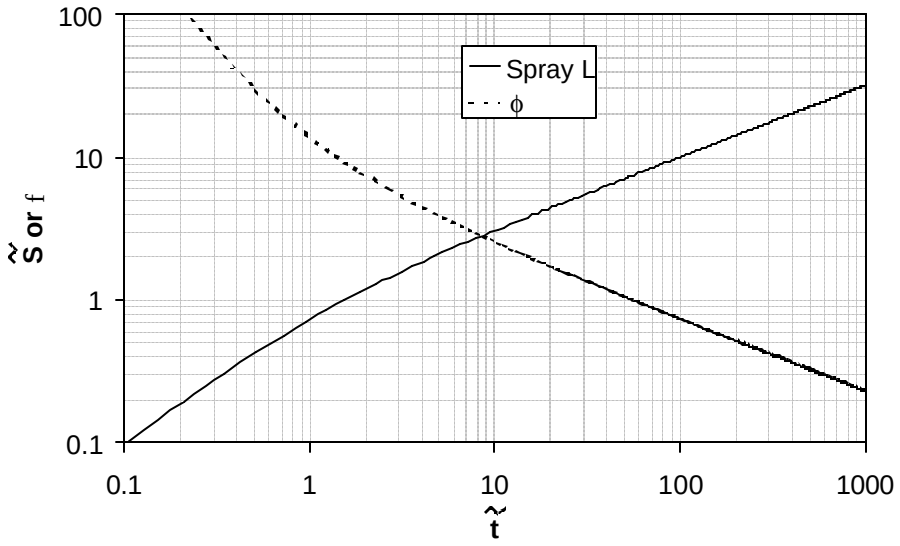


Figure 6.1. Dimensionless Spray Model Results

Calculated liquid lengths were also compared with experimental data for diesel fuel and heptamethylnonane as reported by Higgins et al [27]. The liquid length was calculated for densities of 7.3, 14.8, and 30.0 kg/m^3 and at temperatures between 700 and 1300 K. The results are shown in Figures 6.2 and 6.3 where the solid lines represent the model results and the symbols are data taken from the published literature [27]. The results show very good agreement especially in the temperature ranges (≈1000 K) and densities (≈15 kg/m^3) typical of diesel engines.

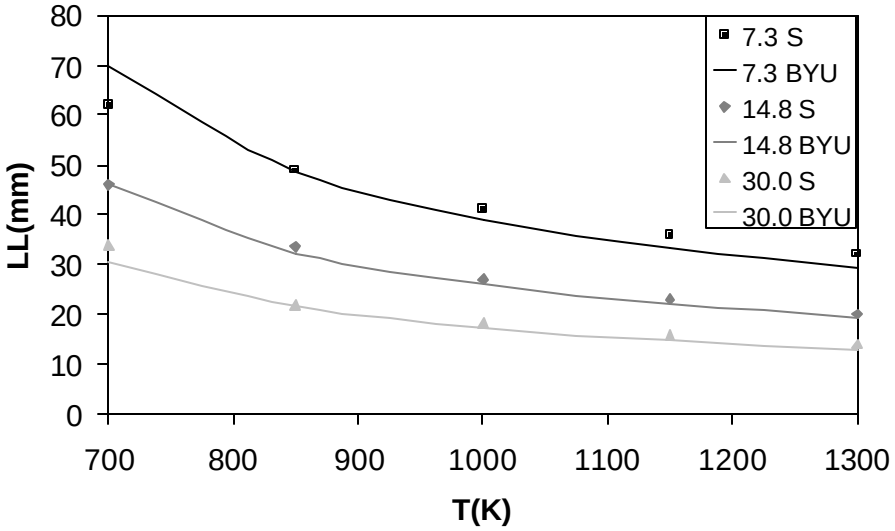


Figure 6.2. Liquid Length Results for Heptamethylnonane Fuel

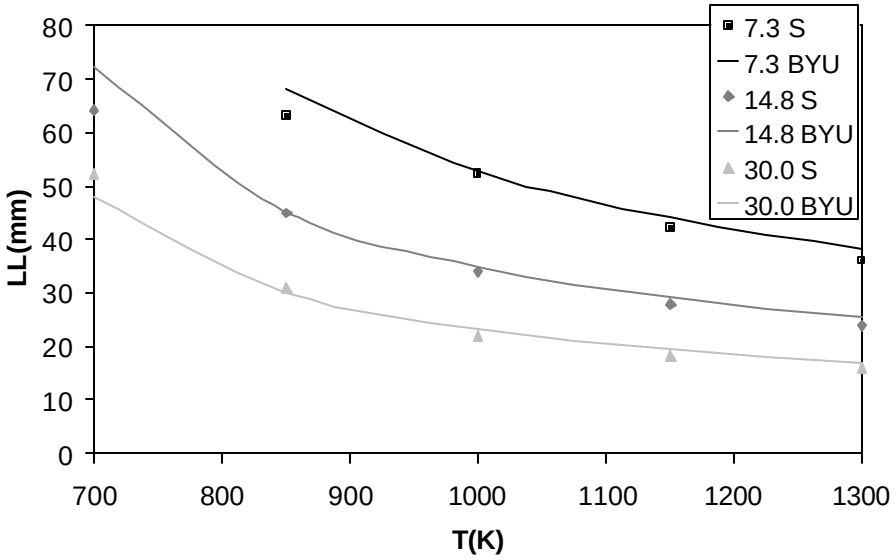


Figure 6.3. Liquid Length Results for Diesel Fuel

6.2. EQUILIBRIUM MODEL RESULTS

The NASA -Lewis code was used as a benchmark to demonstrate the accuracy of the equilibrium code written for this program. Both codes were executed over a series of temperatures between 1500 and 3500 K at an equivalence ratio of 1. They were also run over a range of equivalence ratios between .5 and 5 at a temperature of 2500 K. The pressure for all cases was 100 atmospheres.

The results for mole fractions of major species as function of the temperature at a constant equivalance ratio ($\phi = 1$) are shown in Figures 6.4-6.6. The mole fractions of CO_2 and H_2O are in very good agreement over the entire temperature range. The agreement for CO, H_2, and O_2 are also very good above 1900 K but there is significant deviation below 1900 K (see Figures 6.5, 6.6). This discrepancy is a result of the numerical method used in the solution and can be minimized in two ways. First, by increasing the number of iterations allowed in the solution. This allows the solution to come closer to convergence. Second, by decreasing the size of the equivalence ratio step. The equilibrium program always starts by finding a solution at an equivalence ratio of 1.5 and then steps to the desired equivalence ratio. The solution at an equivalence ratio of 1 is very sensitive at low temperatures. Stepping down in smaller increments of equivalence ratio provides a better guess value for the final solution. Both methods were used and accurate solutions were obtained. However computational time increased dramatically. Since adiabatic flame tempreratures will be higher than 1900 K when the equivalence ratio is 1, increased number of iterations and smaller step size were not used to decrease computational time. The results shown were produced with the code as is appears in Appendix A.

Comment: Why look at Figs 4.2 and 4.3?

Comment: We need to talk about this. It was my impression that the solution simply is not correct at these low temperatures because carbon is not included as one of the products and perhaps other hudrocarbons are also important at low temperatures that are not included. Is this true? I have suggested an alternative explanation in the paragraph I wrote but you need to let me know if I am correct.

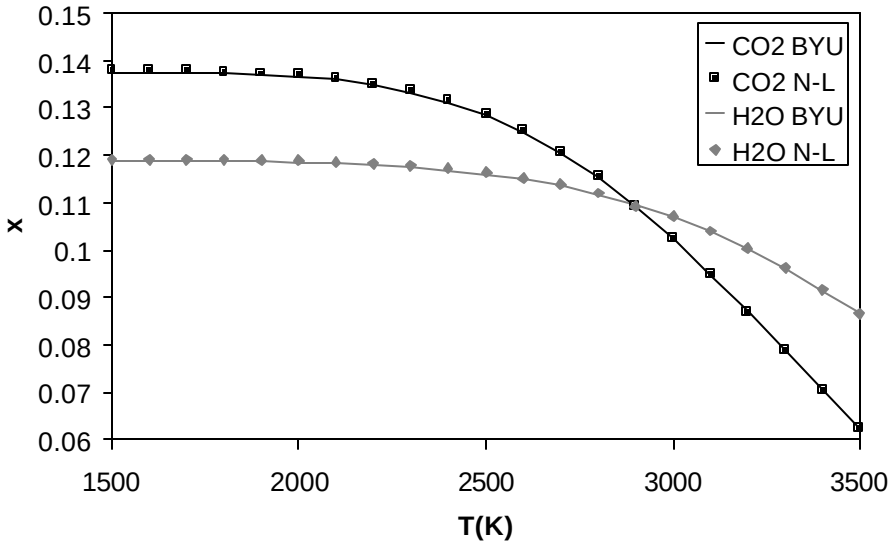


Figure 6.4. Temperature vs. CO_2 and H_2O Mole Fractions

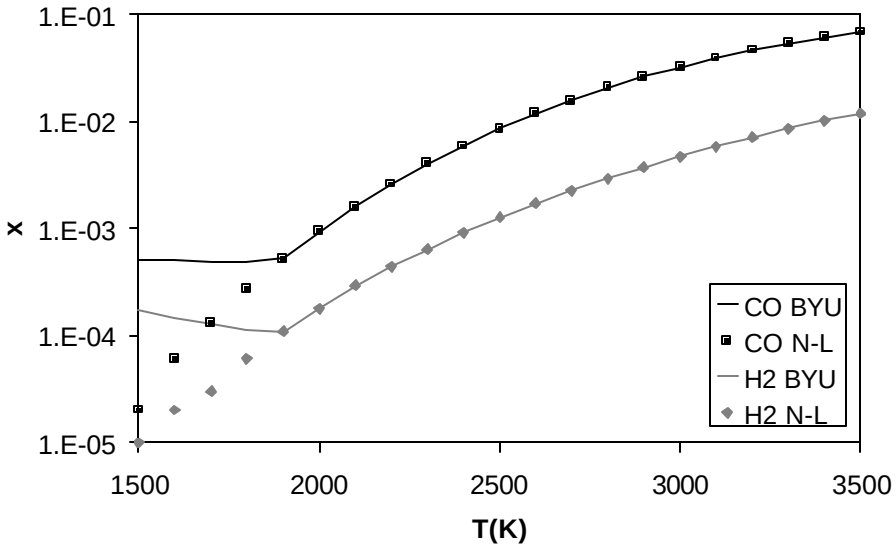


Figure 6.5. Temperature vs. CO and H_2 Mole Fractions

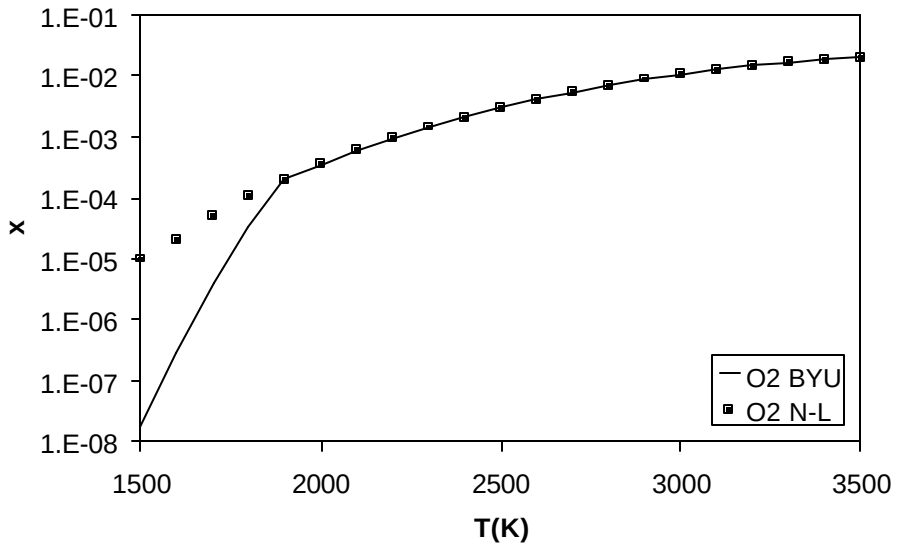


Figure 6.6. Temperature vs. O_2 Mole Fraction

Species mole fractions as a function of equivalence ratio are shown in Figures 6.7, - 6.9. In these figures, the temperature of the products is held constant at 2500 K. For the equivalence ratio tests the mole fractions of O_2, H_2O, and CO_2 are accurate across the whole range (see Figure 6.7). The mole fractions of the rest of the species are accurate up to an equivalence ratio of about 3.5. In Figure 6.8 the mole fractions of CO and H_2 provided by the equilibrium code begin to deviate from NASA-Lewis above an equivalence ratio of 3.5. The CO is over predicted while the H_2 is under predicted. The source of this discrepancy can be seen in Figure 6.9. NASA-Lewis predicts significant amounts of solid carbon above an equivalence ratio of 3.5. The equilibrium code presented here compensates by sending the excess carbon to other carbon containing species such as CO, HCN, and C_2H_2. Because many of these species also contain

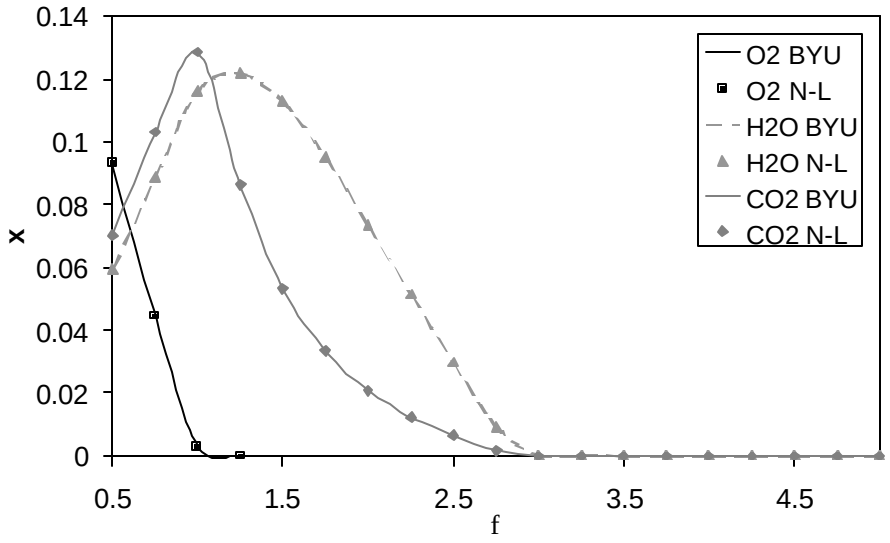


Figure 6.7. Equivalence Ratio vs. O_2, H_2O, and CO_2 Mole Fractions

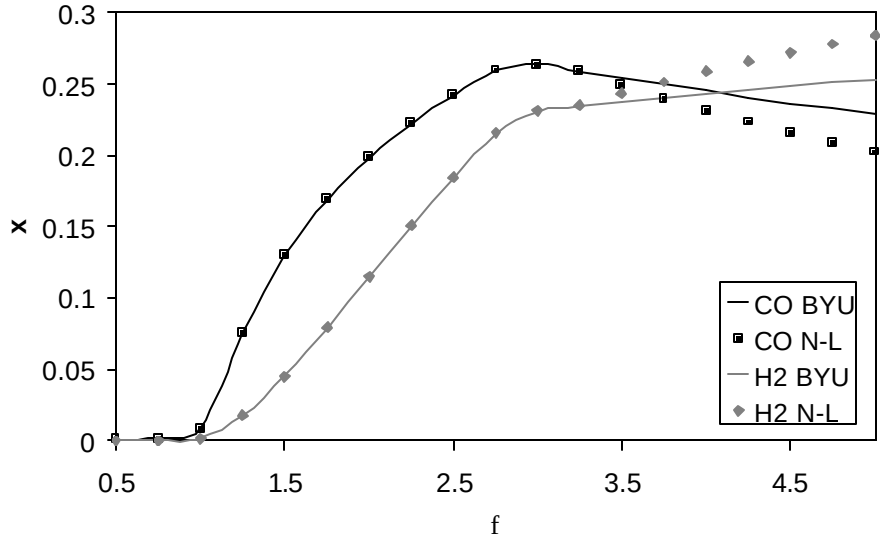


Figure 6.8. Equivalence Ratio vs. CO and H_2 Mole Fractions

hydrogen, H_2 is reduced. The data show that the equilibrium code uses an adequate number of species in the temperature range between 1900 and 2500 K and in a range of equivalence ratios between 0.5 and 3.5. This is adequate to describe most diesel combustion applications.

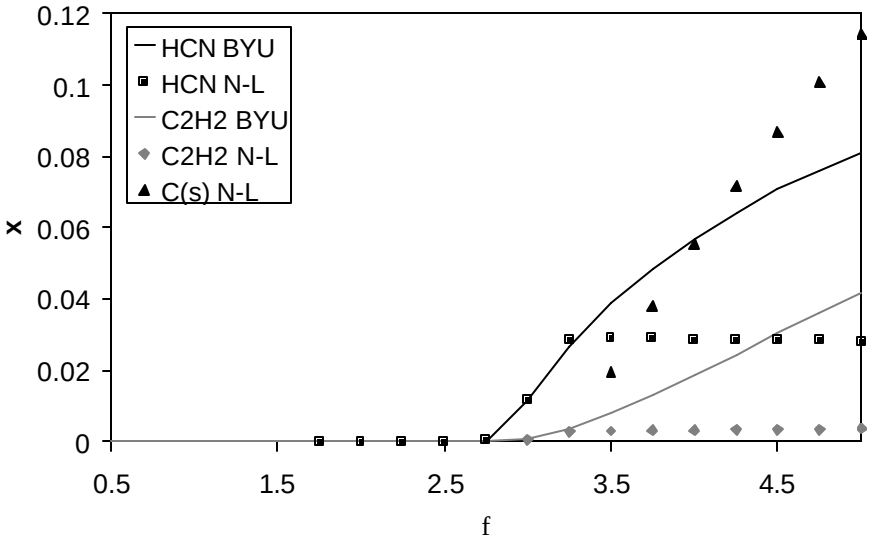


Figure 6.9. Equivalence Ratio vs. HCN, C_2H_2, and C(s) Mole Fractions

One of the primary purposes of the equilibrium code is to calculate the adiabatic flame temperature. The effects of leaving out solid carbon and using a reduced set of species in the equilibrium code have been investigated. The equilibrium code was used to calculate flame temperatures in the same range of equivalence ratios as above and compared with flame temperature results from NASA-Lewis. The results are shown in Figure 6.10.

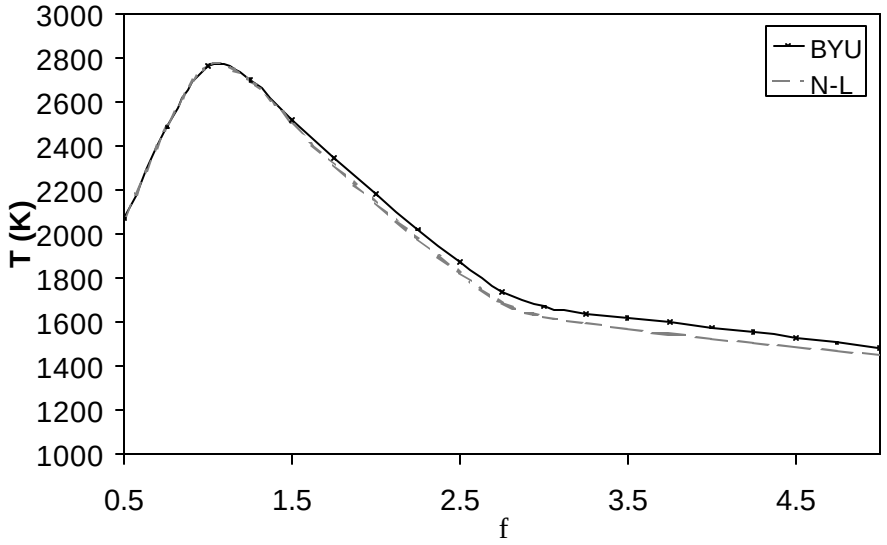


Figure 6.10. Adiabatic Flame Temperature vs. Equivalence Ratio

Temperature agreement between ϕ of .5-1.5 is excellent. Above ϕ=1.5 the model begins to over predict the temperature but follows the same trend. The greatest difference in temperature throughout the range of equivalence ratios shown is 3.4%. These results demonstrate that the equilibrium code produced for this simulation program can predict temperature with 3.4% over the range of conditions needed in a diesel combustion application. Thus, the equilibrium code produced has succeeded at providing a simplified and less computationally intensive equilibrium calculation that is flexible for use in engine simulation programs.

6.3. HEAT RELEASE DATA

The features of diesel engine heat release must be understood in order to compare real data to modeled results. Figure 6.11 shows a typical Apparent Heat Release Rate (AHRR) plot for the Sandia engine. This plot comes from applying Equation 5.6 to the smoothed pressure data. The start of injection (SOI) in this case occurs at 348.5°, after which there is an ignition delay period before the fuel starts to burn. During this time the heat release appears to be negative due to fuel evaporation and heat transfer from the cylinder gas to the cylinder walls. The heat release may also appear negative in this period due to the method selected for smoothing the pressure data. There are a number of ways in which ignition delay has been defined and determined in the literature. In Figure 6.10 it is defined as the time between SOI and when the heat release rate becomes

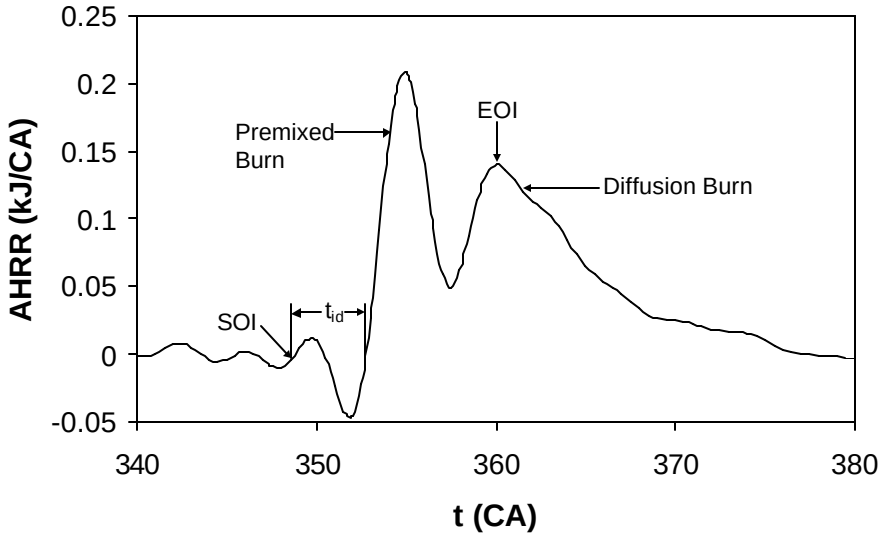


Figure 6.11. Typical Heat Release

positive. It can also be defined as the time when the slope of the heat release goes positive. The choice is somewhat arbitrary and, as will be shown later, changes depending on the smoothing technique used.

The ignition delay is followed by a premixed burn spike in which the vaporized fuel that has mixed with air to within combustible limits burns rapidly. A second peak characterizing the diffusion burn follows the premixed burn. The peak of the diffusion burn roughly corresponds with the end of injection (EOI) when the fuel jet is largest, which in this case is at 360°. After EOI, the rate of apparent heat release decreases as the fuel jet or pockets of remaining fuel are consumed.

Smoothing the pressure data has a significant effect on the shape of the heat release. This is a result of the pressure derivative in Equation 5.6. Figure 6.12 shows a

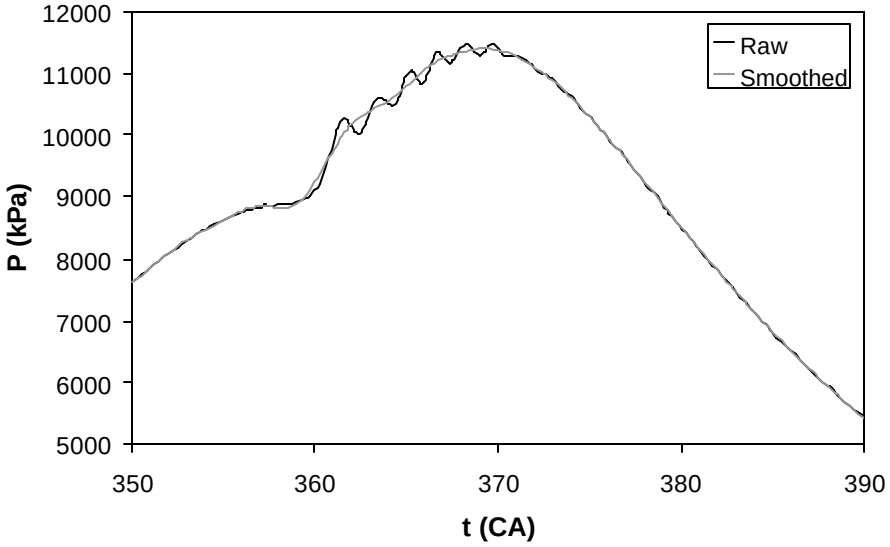


Figure 6.12. Raw and Smoothed Cylinder Pressure Data

raw and smoothed pressure data set from the BYU engine. The raw data oscillates during the combustion event. The result of heat release calculations on the raw data produces the meaningless plot in Figure 6.13.

Filtering the pressure data is necessary to remove the oscillations that produce these results. It should be understood that filtering also smoothes out some of the details of the combustion process. Depending on the filter used, the smoothed heat release can be drastically different. Figure 6.14 shows AHRR results using the Fourier filter described in Chapter 5 changing the number of terms in the series from 100 to 300.

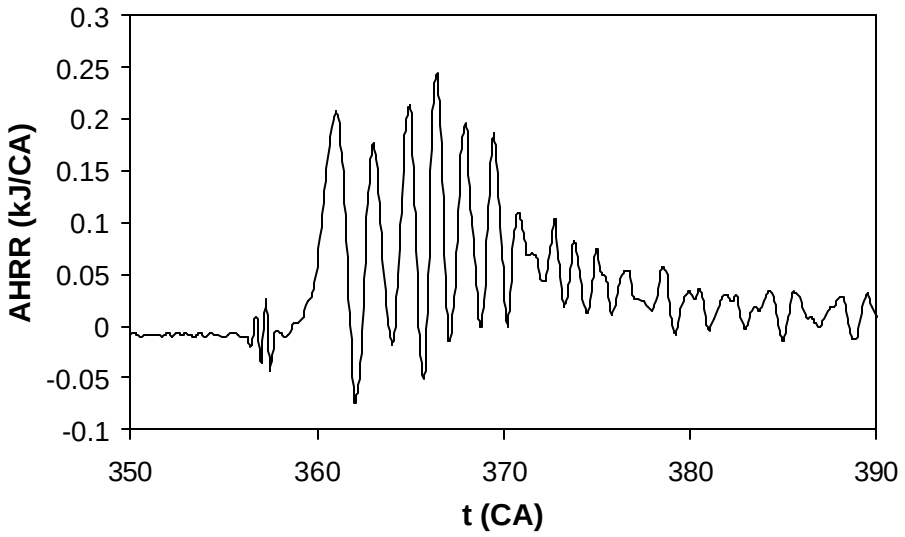


Figure 6.13. Apparent Heat Release from Unfiltered Pressure Data

Using 100 terms the premixed burn spike does not show up at all. The premixed burn can be seen in both the 200 and 300 term cases. Using more terms produces more oscillation and approaches the raw data. As noted earlier 200 terms were used when

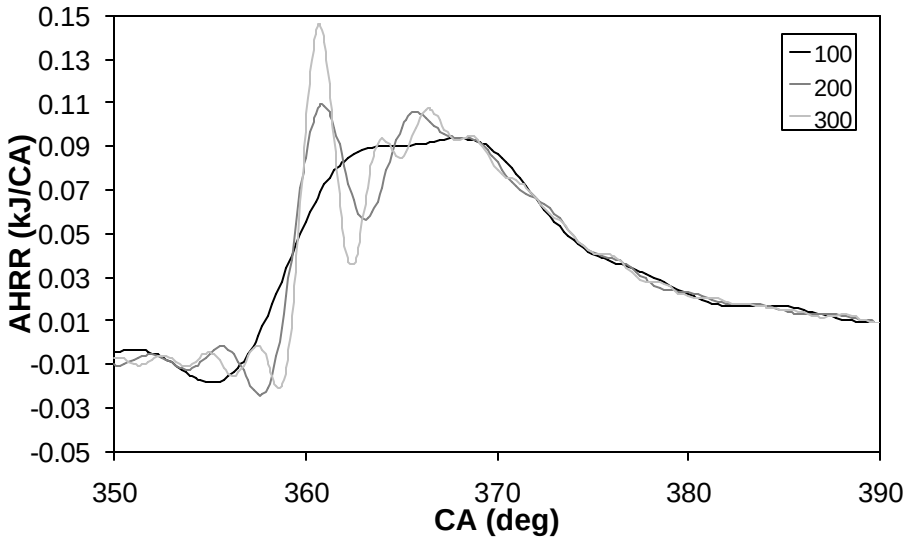


Figure 6.14. Effect of Number of Terms on AHRR

smoothing the data in this work. This seemed to give the best compromise between detail and removing oscillation.

As can be seen, the details of heat release are uncertain because of noise in the pressure data and the need for smoothing. Ignition timing appears to change based on the filter used, as does the shape of the premixed and diffusion burns. Attempts to compare heat release from the engines with results from the model must therefore be made with the uncertainty of the measured data in mind.

Work is being done to better understand and analyze pressure data. The data from the bomb was smoothed using more sophisticated methods developed at Sandia [32]. These methods seek to retain as much information about the premixed spike as possible

by employing different smoothing methods before and after the spike as discussed in Chapter 5.

As a result of work done at Sandia [32] it has been shown that the premixed burn is in reality much shorter and the spike is much higher than is normally shown by the methods used to processes data as shown in Figure 6.11. For years, modelers have been attempting and succeeding in producing premixed burn duration similar to that of Figure 6.11, but Siebers showed that the two factors present in taking the data but not present in the model produce the inaccurate width and height for the premixed burn. First, in an engine there are normally between four and eight fuel jets and each jet ignites at a different time. Thus, the pressure rise of a single jet is smeared by the pressure rise from multiple jets igniting. This smearing is exacerbated by the fact that the distance from each jet to the pressure transducer is different and therefore the time when the transducer records the pressure rise relative to the timing of ignition is different for each jet. Second, the smoothing routine broadens and lowers any premixed spike. Since models normally predict only the behavior of a single jet and they do not contain noise or include wave dynamics within the cylinder, they should predict narrower and higher premixed burn spikes than are seen smoothed pressure data from multiple nozzle injectors in real engines.

The total heat release obtained by numerically integrating the AHRR is shown in Figure 6.15 for one of the BYU engine cases. The integrated heat release shows the amount of energy released up to any point in the combustion process and reaches its peak at approximately 390° for this case. In the absence of heat transfer, the integrated heat release should equal the total fuel energy released during combustion in the engine of

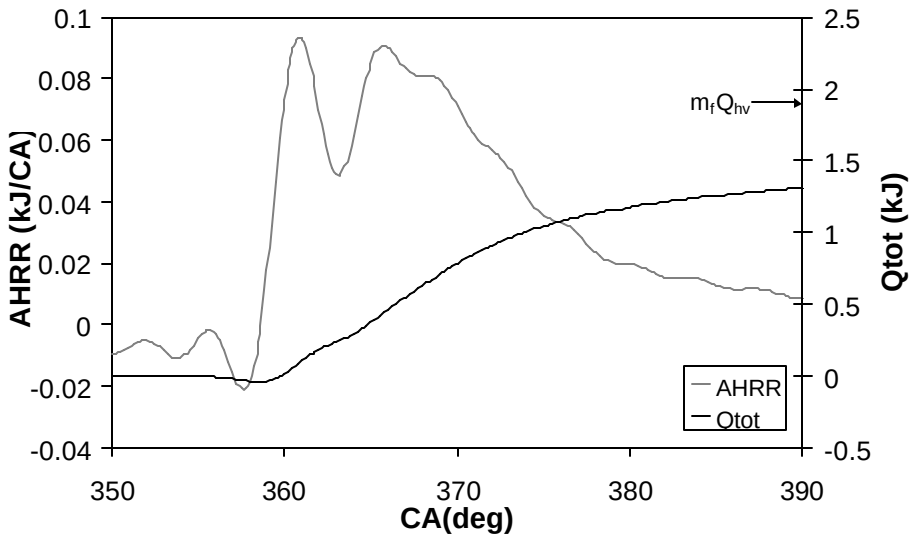


Figure 6.15. Integrated Heat Release

$m_f Q_{hv}$, which is 1.93 kJ for this case. The difference between the total fuel energy released and the integrated apparent heat release rate, or heat transfer from the engine, is about 32%, which is typical for an engine.

6.4. SANDIA BOMB COMPARISON

The pressure rise results will first be compared followed by the apparent heat release comparison. The pressure rise is given as $P-P_0$ where P is the pressure in the chamber and P_0 is the pressure without injection and combustion as discussed in Chapter 5. It should be noted that since the AHRR in this case is represented by dP/dt, the pressure rise represents the integrated heat release.

The pressure increase from SOI for each of the four combustion bomb cases can be seen in Figures 6.16-6.19. The shape of the pressure curve is similar for model and data but there are notable differences. The modeled pressure begins to rise earlier than the experimental data for all cases. The abrupt increase in the bomb pressure data at the start of combustion results from the premixed burn being more rapid in the experiments than in the model results. The rate of pressure rise for the 246-µm and 180-µm nozzles is over predicted while the 100-µm and 71-µm nozzles are reasonably close. The predicted total pressure for the 246-µm and 180-µm nozzles was also higher than the measurement. The 100-µm and 71-µm nozzles are once again reasonably close though the 100-µm is also higher than the data.

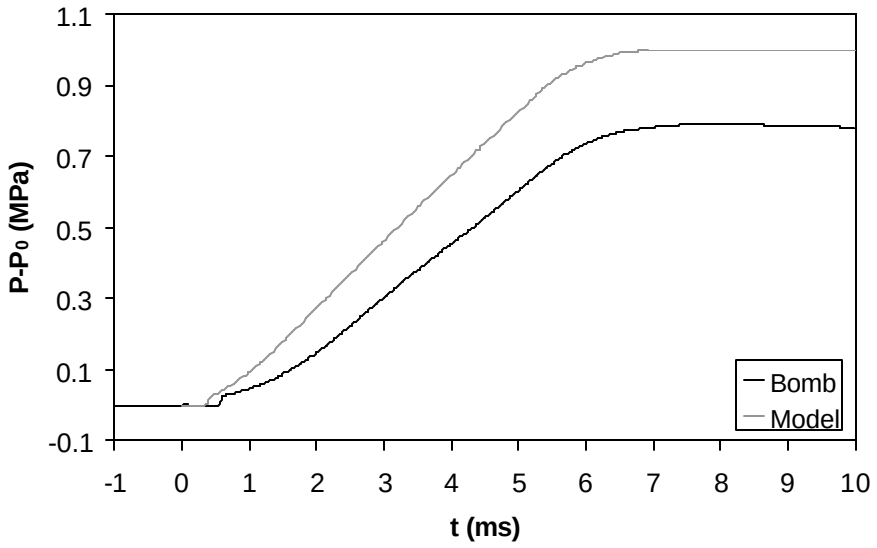


Figure 6.16. Sandia Bomb Pressure Rise Using the 246-µm Nozzle.

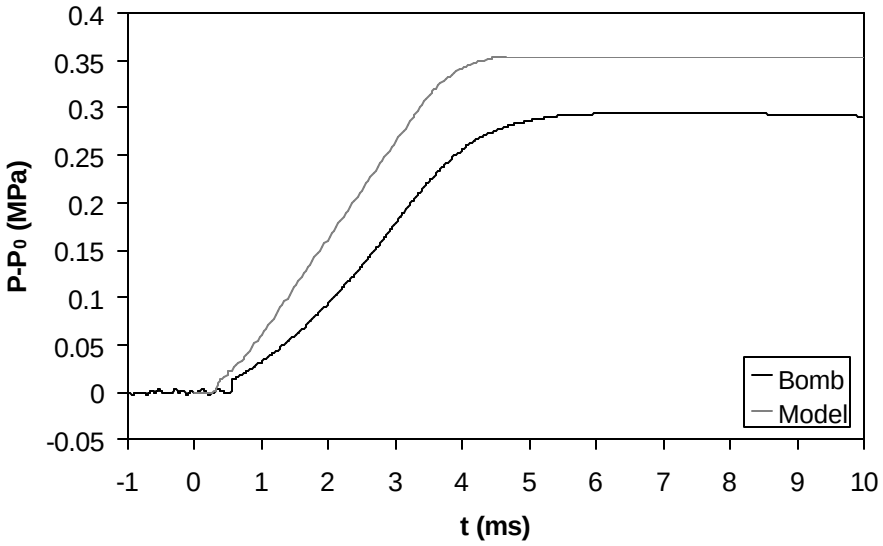


Figure 6.17. Sandia Bomb Pressure Rise Using the 180-µm Nozzle.

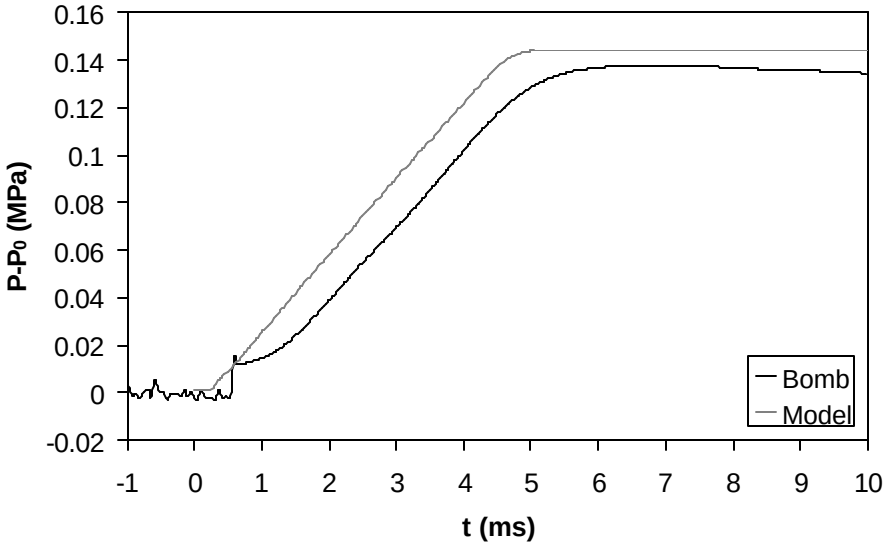


Figure 6.18. Sandia Bomb Pressure Rise Using the 100-µm Nozzle.

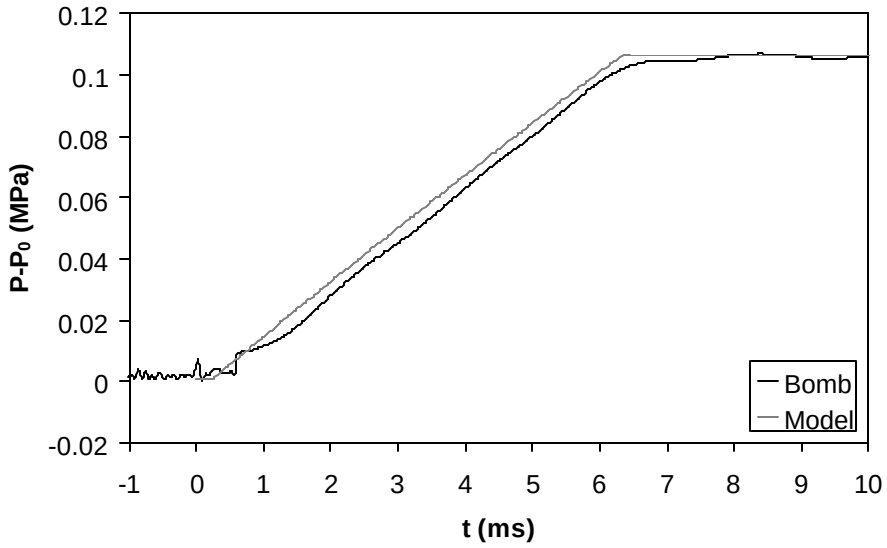


Figure 6.19. Sandia Bomb Pressure Rise Using the 71 µm Nozzle.

There are several possible reasons for the discrepancies. First the fuel mass injected is unknown in the combustion bomb experiments and the calculated value has more uncertainty. The calculated fuel injected was determined using Equations 4.13 and 3.8 which are repeated below. Uncertainty in any of the nozzle parameters or the injection pressure produce uncertainty in the calculated fuel injected..

$$\dot{m} = C_a A_{inj} ?_f U_f \tag{4.13}$$

$$U_f = C_v \sqrt{2\frac{(P_f - P_a)}{?_f}} \tag{3.8}$$

A second reason for the differences may be caused by heat transfer effects, which have not been included in the model for the bomb cases. One compelling argument for

this explanation is that heat transfer would be expected to decrease with decreasing nozzle size. This is because radiating soot is the dominant mode of heat transfer and as the nozzle becomes smaller, the flame actually transitions from heavily sooting to soot free. Assuming heat transfer to be the cause of the discrepancy, cases were run for all nozzles, (except the 71-µm which already matched the total heat release) as shown in Figures 6.20 – 6.22. Heat transfer was increased in each case until the peak pressures in the model matched the data. The multiplier used in the heat transfer model is included in Table 5.7. These Figures eliminate potential differences in heat transfer or fuel injected, and show that in spite of these potential differences in the model and data, the model still over predicts the rate at which the fuel is burned. In each case, the peak pressure is reached in the model before it is reached in the data.

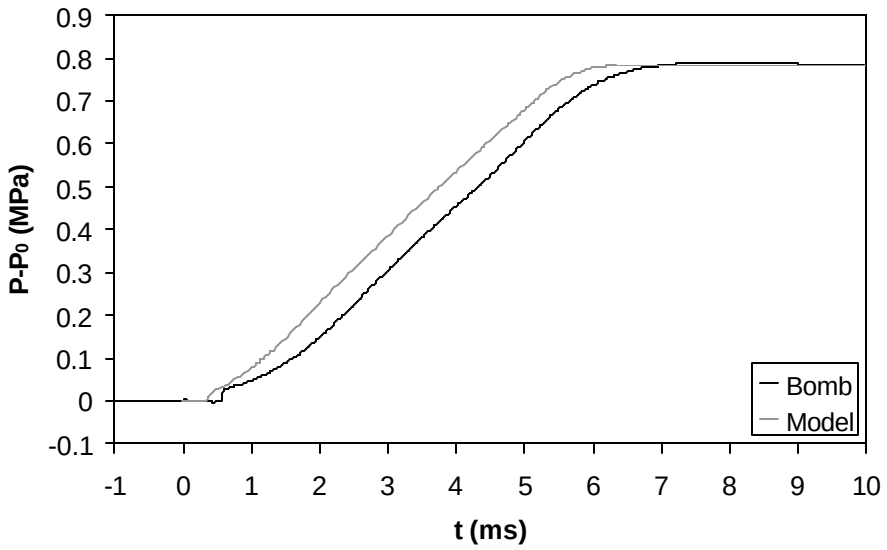


Figure 6.20. Sandia Bomb Pressure Rise Using the 246-µm Nozzle with Extra Heat Transfer.

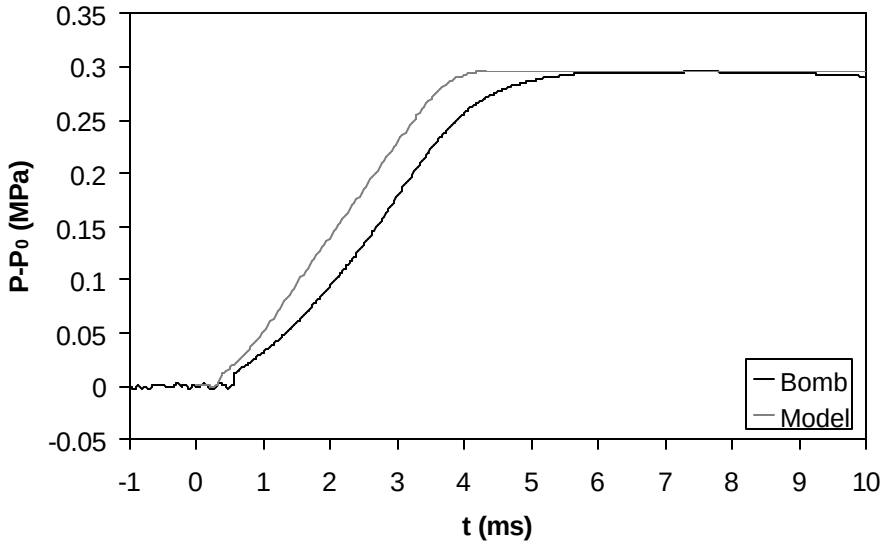


Figure 6.21. Sandia Bomb Pressure Rise Using the 180 -µm Nozzle with Extra Heat Transfer.

It is interesting to note that the slope of pressure rise over the later three fourths of the combustion process is similar for the model and the experiment in each case but it is the first quarter of combustion that differs. The period immediately following the premixed burn is not a linear increase in the actual data but is linear in the model. This difference can be explored more easily using the rate of pressure rise or AHRR results.

Results for heat release rate are shown in Figures 6.23-6.26 for the cases with heat transfer added to match peak pressures except for the 71-µm, which compared well without heat transfer. The model heat release results produce the expected features of diesel combustion. In all of the heat release plots, there is a delay before any fuel is burned followed by a short premixed burn spike. The heat release then increases almost

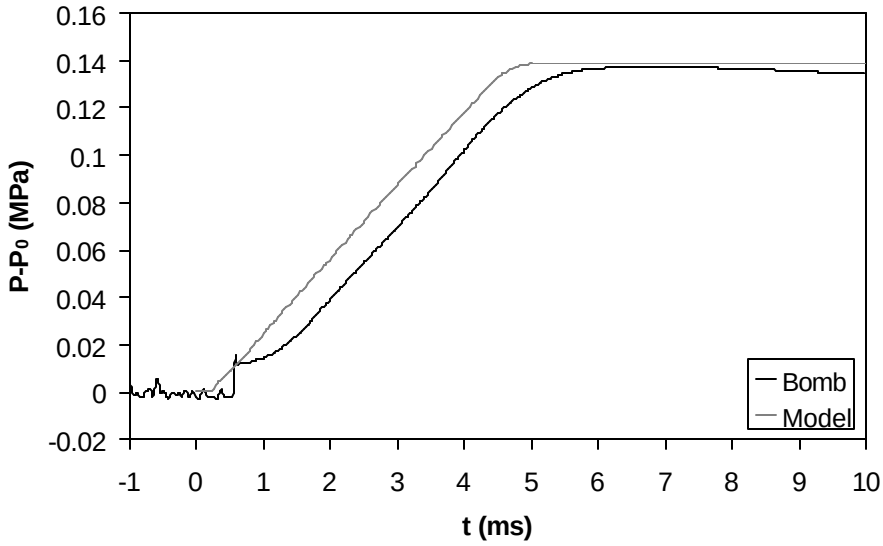


Figure 6.22. Sandia Bomb Pressure Rise Using the 100 -µm Nozzle with Extra Heat Transfer.

linearly until reaching a maximum. If the amount of injected fuel is large enough, the heat release reaches a plateau and remains fairly constant until the end of injection. At the end of injection the heat release then drops off to the end of combustion.

These features can be explained by recalling the physical processes occurring during combustion. After ignition the burning fuel jet penetrates into the chamber and increases in size. The rate of air entrainment increases proportional to the surface area growth of the jet increasing the rate of heat release. This increase in entrainment rate continues until the jet reaches the flame length or the point where the leading edge of the jet is stoichiometric. Beyond this flame length the fuel in the jet is completely burned therefore additional air entrainment no longer increases the heat release rate. The heat

release rate then remains approximately constant until the end of injection. This is clearly seen in both modeled and measured data in Figures 6.23, 6.25, and 6.26. In Figure 6.24 the measured data shows the linear increase in the heat release rate but no plateau. This is because the injection duration was short enough that the flame length had not been reached or had just been reached by the end of injection. It should be noted that the oscillations appearing in the data are most likely due to the filtering process and may not be real. When fuel injection ends the remainder of the fuel in the chamber burns out until the all of the heat has been released.

While the basic features are identifiable in the model there are quantitative differences between the model and the data. The start of combustion occurs to early in the model accompanied by a premixed burn that is too long. This is seen in all the combustion bomb modeling and is consistent with the early and slower initial pressure rise as seen earlier in the pressure data.

The rise in the heat release rate just after the premixed spike is different in the model results than in the data. This rise is more rapid in the model and in the case of the two smaller injector nozzles, the flame length or steady state apparent heat release rate has already been reached at the end of the premixed burn period. The magnitude of the apparent heat release rate at the end of the premixed burn as shown in the figures can be altered by the selection of the filtering routine. The magnitude and slope of the apparent heat release rate predicted by the model are also affected by the quantitative inaccuracies in the ignition delay and premixed burn sub models. Perhaps the best way to compare the model and measured data is to look at the time between the SOI and the time to reach the flame length or steady state of apparent heat release. In each case the model reaches the

flame length before the data, indicating that the model over predicts the rate of air entrainment.

The peak level of heat release agrees reasonably well between the model and the data. This suggests that although the model predicts more rapid air entrainment, the flame length and the amount of air entrain ed prior to the flame length are reasonably accurately predicted.

At the end of injection the model predicts a burnout that is more rapid than the data. This is also consistent with the air entrainment rate being over-predicted in the model.

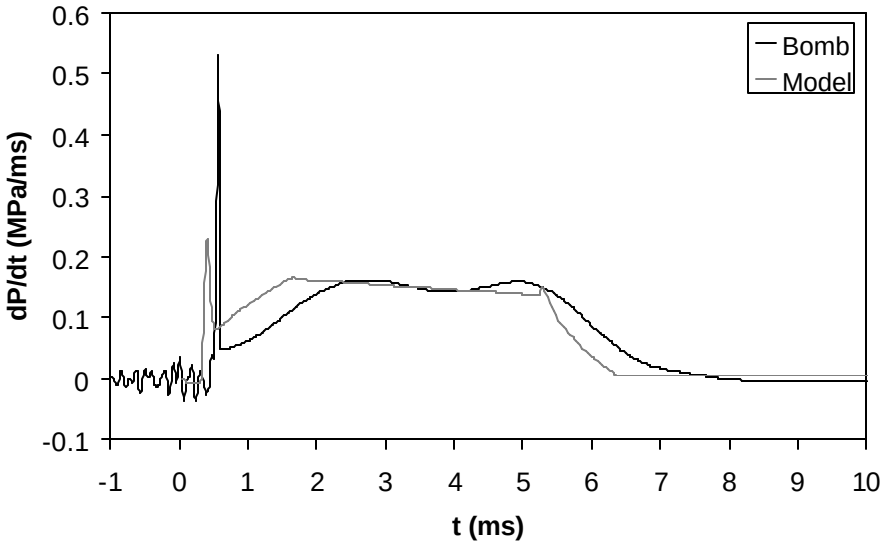


Figure 6.23. Sandia Bomb Heat Release Using the 246-µm Nozzle with Extra Heat Transfer.

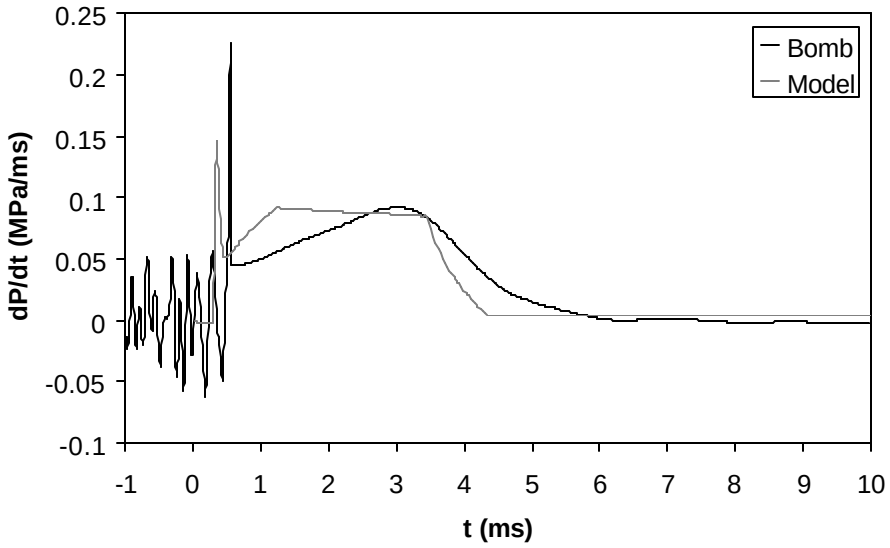


Figure 6.24. Sandia Bomb Heat Release Using the 180-µm Nozzle with Extra Heat Transfer.

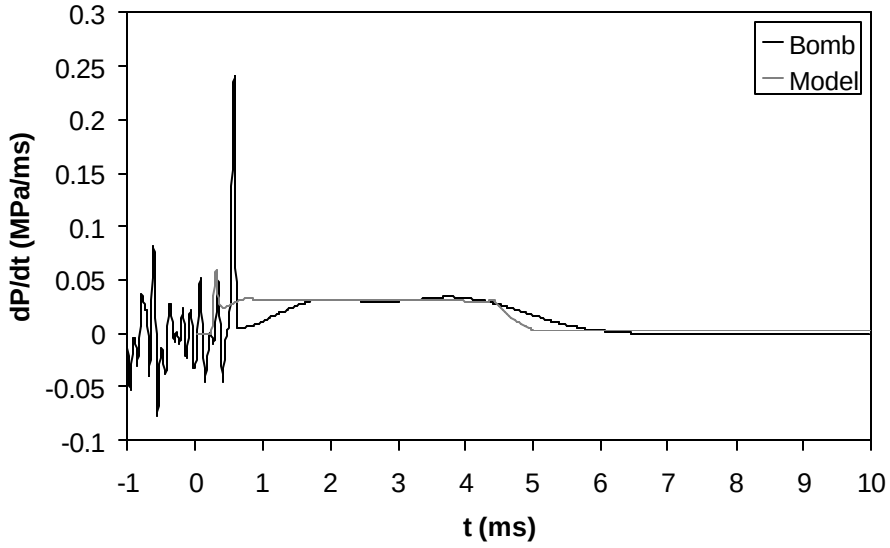


Figure 6.25. Sandia Bomb Heat Release Using the 100-µm Nozzle with Extra Heat Transfer.

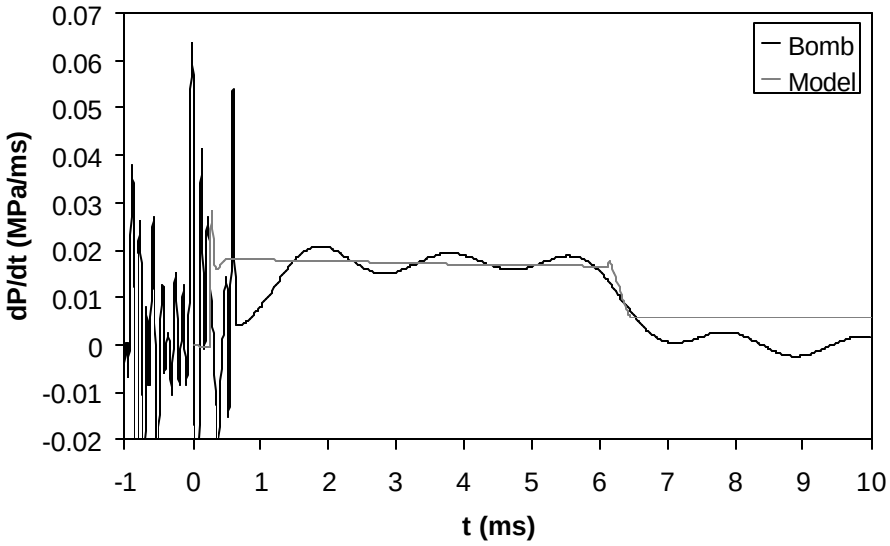


Figure 6.26. Sandia Bomb Heat Release Using the 71-µm Nozzle.

The use of the combustion bomb data allows a unique analysis of the model because in the case of jets coming from the two smaller orifices, the flame length was reached before the jet could reach the wall. In the case of the two larger orifices, the jet ran into the wall before reaching the flame length. It is interesting to note that the data from the 246-µm orifice jet (Figure 6.23) shows the rate of heat release to remain relatively constant even though the jet has reached the wall before the flame length is reached (before 1.7 ms). This suggests that wall effects, which are assumed to be negligible in the model, may actually cause little change to the rate of heat release.

This combustion bomb data does however clearly suggest that the penetration rate of the jet is too rapid in the model. The penetration model is based on a non-reacting, non-vaporizing spray and therefore, poor prediction of reacting and vaporizing sprays

might be expected. In a reacting and vaporizing spray, the density of the jet is decreased relative to a non-reacting spray. Studies of gas jet penetration suggest that lower gas density will decrease jet penetration. The over prediction of penetration leads to an over prediction of entrainment rate and heat release. Similar results will be seen in the engine data which follows but in the engine data, all jets have significant wall impingement before reaching the flame length, therefore, the bomb data presented above is the only evidence that the higher heat release rates are not caused by wall interactions.

6.5. SANDIA ENGINE COMPARISON

The first set of data presented is the 1200-RPM, low fuel load, normal injection timing data set. Heat release results are graphed for constant injection pressure, ramping injection pressure, constant injection pressure with added heat transfer, and ramping injection pressure with added heat transfer in Figures 6.27-6.30 respectively. It should be noted that the constant used in the ignition delay model is derived from the data in Figure 6.28, and therefore the ignition delay in that figure matches by matter of definition.

The basic features of the engine heat release are present in the model. Injection begins at 348.5 CA and apparent heat release drops initially due to evaporating fuel. The sudden rise and first peak in the model apparent heat release are a result of the initial premixed burn where evaporated fuel in zone 2 is burned. This period ends at approximately 354 CA after which combustion limited by air entrainment into the jet begins. Since the jet has already penetrated a fixed distance into the chamber, this burn rate does not start at zero and increases as the jet grows larger. The second peak occurs when fuel injection ends. At this point, the jet and the rate of air entrained into the jet is a maximum and afterwards decreases. An inflection point is visible in the decreasing

apparent heat release at approximately 372 CA. At this point, the jet has reached the flame length and therefore additional air entrainment no longer contributes to combustion in the portion of the jet beyond the flame length. The apparent heat release decreases more rapidly after this point is reached. These same features (premixed burn peak, mixing limited peak or end of injection, and flame length inflection point) can be seen in all of the engine modeling results.

Comparing the modeling results to the engine data, the constant injection pressure results (Fig. 6.27) produce an early ignition similar to the results seen in the bomb data. The heat release following the premixed spike is higher in the model. In the engine injection ends before the jet reaches the flame length and the heat release never reaches a steady value. At the end of injection the model AHRR initially drops faster than the engine data. As with the bomb data, these differences appear to be produced by the faster air entrainment of the model. The inflection point produced by the model at 372 crank angle degrees, when the flame length is reached, is not seen in the engine data. After studying numerous AHRR profiles, it can be seen that engines often produce an inflection point in the heat release near the end of combustion but opposite of the modeling results, the decrease of heat release normally slows at the inflection point producing what is often referred to as a "tail" to the AHRR rather than the decrease in AHRR shown by the model. This suggest that the physical processes near the end of combustion are not well characterized by the model which is not surprising because late in the combustion process, the assumed conical shape of the jet has been completely changed by fluid motion within the chamber.

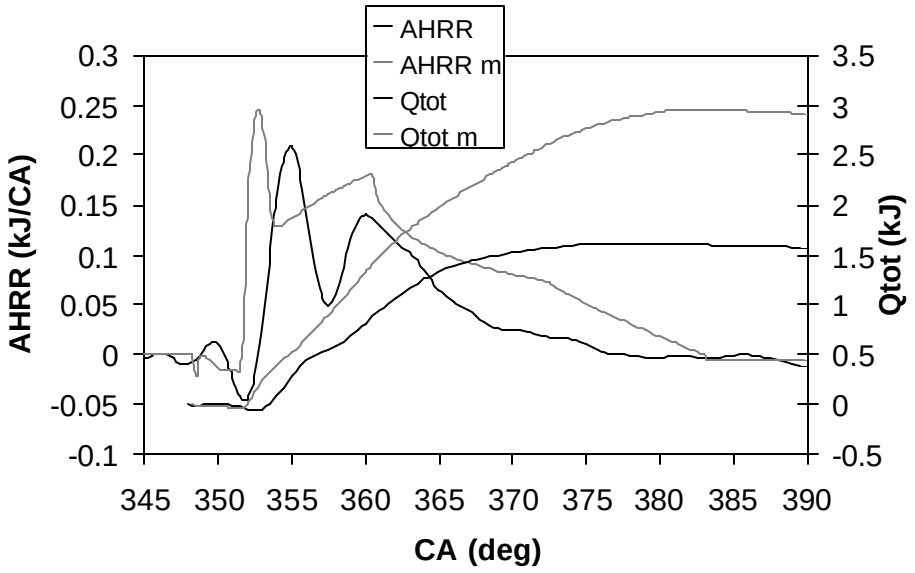


Figure 6.27 Sandia Engine Heat Release Comparison, 1200 RPM, Low Fuel Load, Normal Timing, Constant Injection Pressure.

Ramping the injection pressure in the model improves agreement with the measurement during the early part of heat release as shown in Figure 6.28. Ramping the injection pressure produces a slower start to the injection processes, reducing the initial spray penetration and air entrainment. There is a longer time (more crank angles) from the start of injection until the vapor phase is formed and less vapor in the premixed burn region (zone 2) when ignition begins. This increases the ignition delay period and decreases the peak of the premixed burn to better match the data. Experimental data from the Sandia Cummins engine shows that the injection pressure actual does increase linearly from approximately 60 MPa at the start of injection to 100 MPa and the end injection justifying this modeling approach. The jet penetration correlation presented in

Equation 3.1 was developed with an injector that had a "top hat" or flat pressure profile throughout injection. The results in this figure and the results from the combustion bomb data presented above both suggest that the jet penetration correlation over-predicts the penetration rate of the fuel.

It should be noted that the exact agreement for the start of combustion between the model and the data in this case is caused by the fact that this was the data set used to determine the activation energy constant in the ignition delay model. This operating condition was selected because the good agreement in the magnitude of the peak of the premixed burn fraction (the first peak of the apparent heat release rate). This suggests

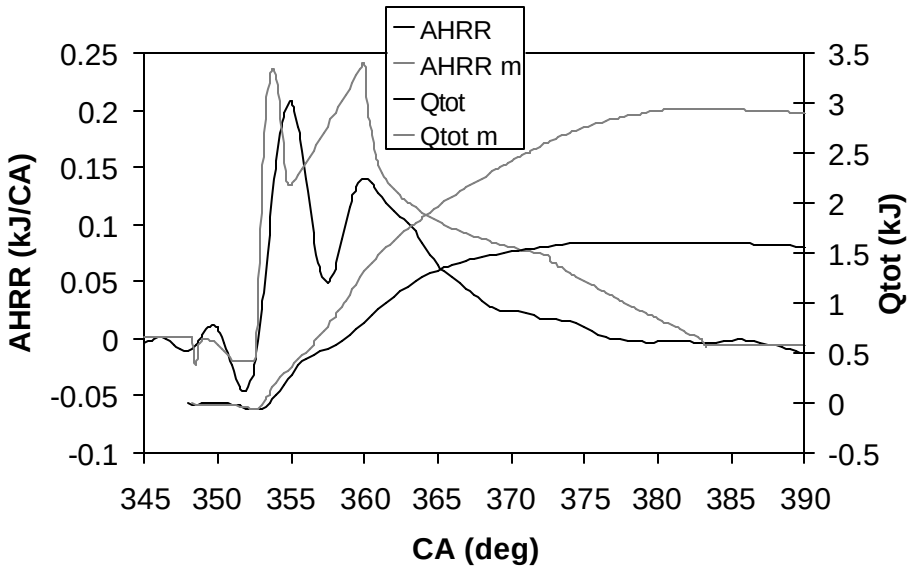


Figure 6.28. Sandia Engine Heat Release Comparison, 1200 RPM, Low Fuel Load, Normal Timing, Ramping Injection Pressure.

that for the ignition delay selected, the correct amount of fuel has been evaporated and burns during the initial premixed burn period.

In both the constant pressure and ramping injection pressure cases, the total heat release is almost double in the model compared to the engine data. Since the model produces the same total heat release as the energy in the fuel, the discrepancy must be the result of heat transfer, blow-by, or incorrect compression ratio. Usually, heat transfer is 25 – 35 % of the total heat release. Since this is an optical engine that is skip fired (therefore the wall are cooler), the heat transfer and blow-by may be greater than is normal. One option commonly used for eliminating the effects of heat transfer to allow a comparison of the model and engine combustion is to increase the heat transfer in the model until the peak heat release in the model and data match. The heat transfer coefficient was multiplied until the total heat release from the model matched the total from the engine data. Results for both the constant pressure and ramping pressure cases are shown in Figures 6.29 and 6.30.

The basic shape of the AHRR is unchanged by increased heat transfer even though heat transfer is not constant. With increased heat transfer the AHRR is decreased at all crank angles and produces a better match to the engine data. It is still clear however, particularly in the constant pressure injection case that the AHRR occurs earlier in the model indicating that the jet penetration and air entrainment are over-predicted causing the heat release to occur too early. As with the no heat transfer results, the ramped pressure improved the model prediction.

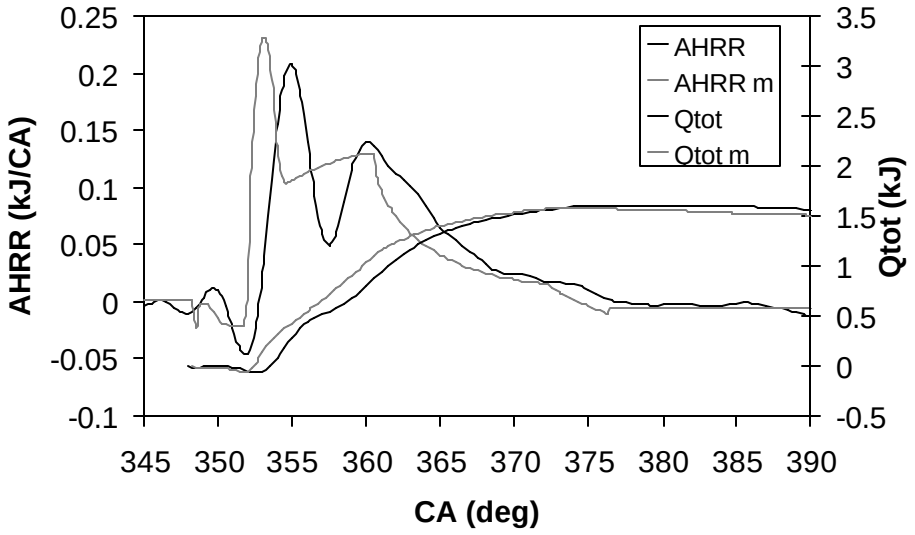


Figure 6.29. Sandia Engine Heat Release Comparison, 1200 RPM, Low Fuel Load, Normal Timing, Constant Injection Pressure, Extra Heat Transfer.

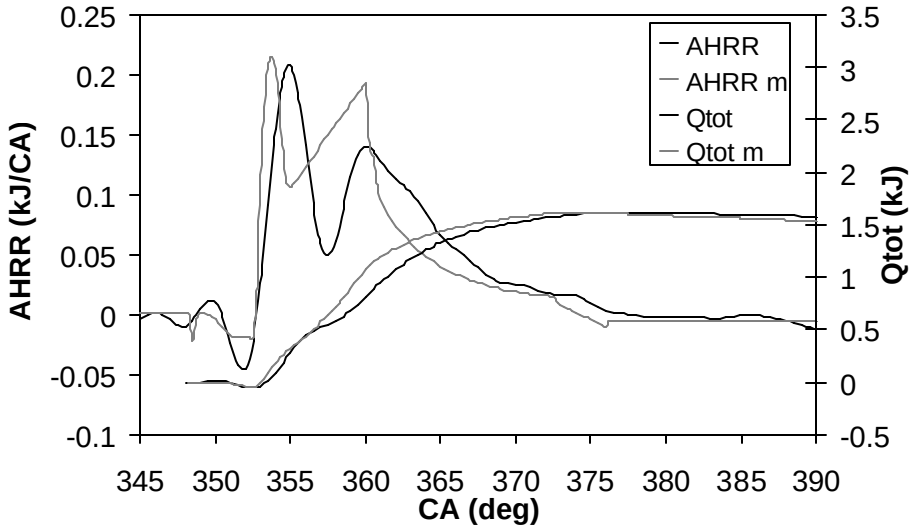


Figure 6.30. Sandia Engine Heat Release Comparison, 1200 RPM, Low Fuel Load, Normal Timing, Ramping Injection Pressure, Extra Heat Transfer.

Figure 6.31 shows a lnP-lnV plot for the ramping injection pressure case in comparison to the measured data. In this case no difference can be seen between the data and the experiments, as most of the detail of the combustion process is lost in showing only the pressure and in using a log scale. Because of this only the heat release results will be used for the remainder of the comparisons. The figure does show however, that if the details of the combustion process are not important, the model does and excellent job of reproducing the basic features of the diesel cycle.

The model was also tested using different fuel loads, engine speeds, and timings. The fuel load for the case seen in Figure 6.30 was raised while all other parameters remained constant and the results are shown in Figure 6.32. The model uses a ramping injection pressure and heat transfer is increased. The ignition delay and premixed burn

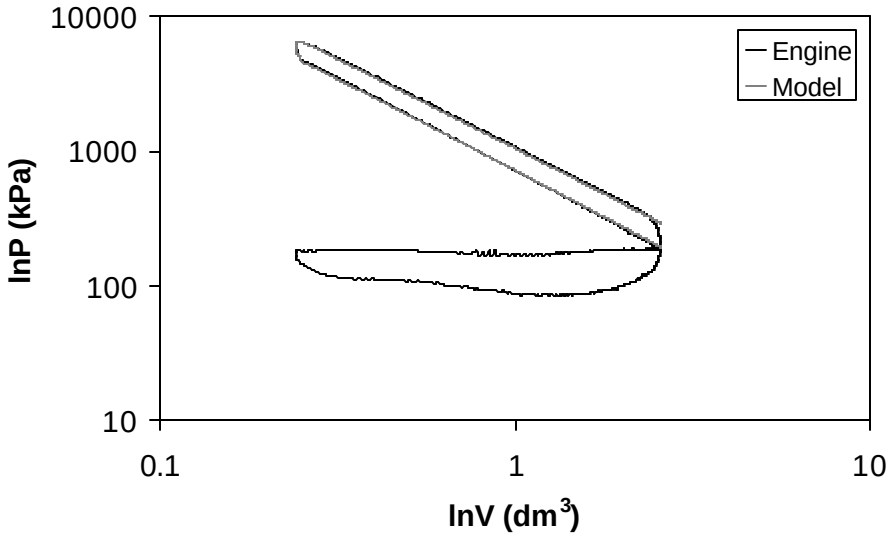


Figure 6.31. Sandia Engine Pressure Comparison, 1200 RPM, Low Fuel Load, Normal Timing, Ramping Injection Pressure, Extra Heat Transfer.

are almost identical to the low fuel load case seen in Figure 6.30. The diffusion burn begins similarly but continues to increase until the end of injection. The heat release rate then drops rapidly. This is due to greater jet penetration and air entrainment rate of the model compared with the actual spray. Although the peak heat release rate is overpredicted the diffusion burn duration is increased in the model, as is the case in the engine.

Figure 6.33 illustrates the results of increasing the engine speed. The ignition delay timing from the model is still reasonable. The modeled premixed burn spike is underpredicted and to short in duration. This could result from the model itself or the injection pressure ramping. If the injection pressure ramping starts too low there will not

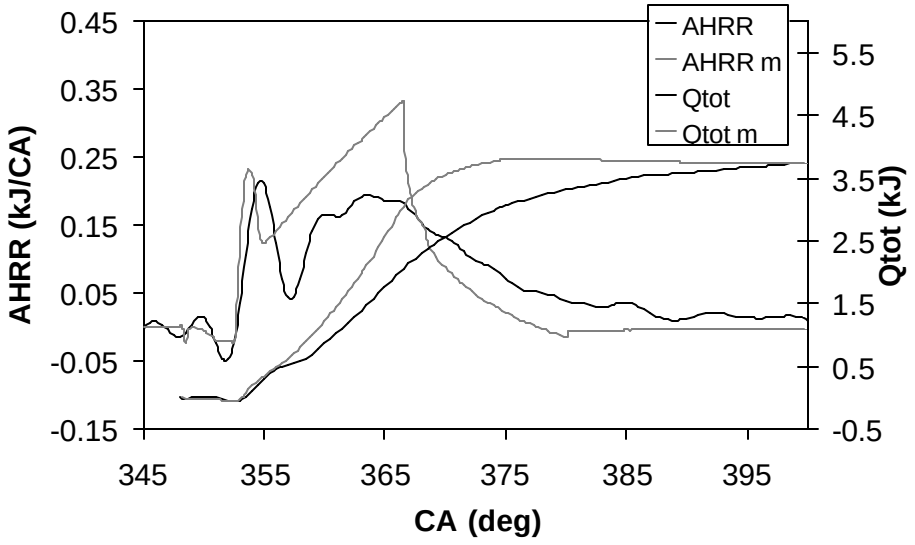


Figure 6.32. Sandia Engine Heat Release Comparison, 1200 RPM, High Fuel Load, Normal Timing, Ramping Injection Pressure, Extra Heat Transfer.

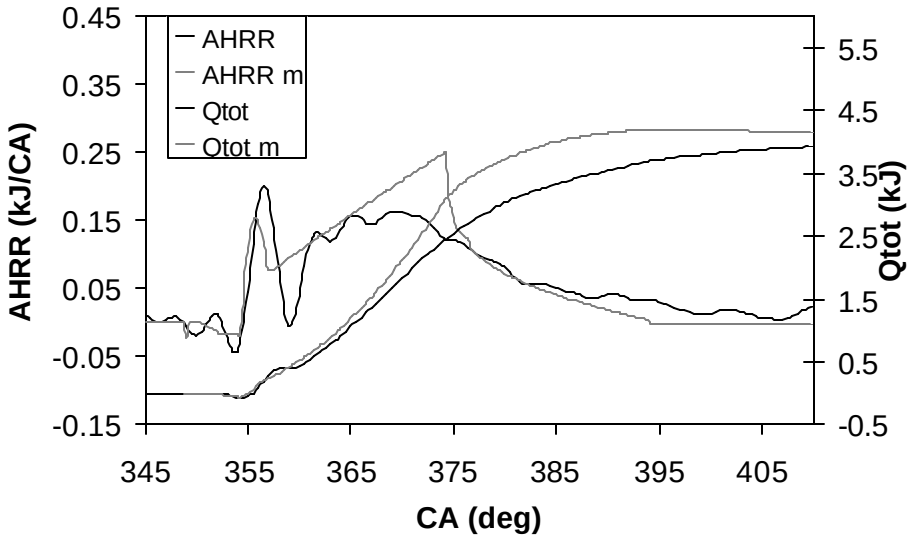


Figure 6.33. Sandia Engine Heat Release Comparison, 1680 RPM, High Fuel Load, Normal Timing, Ramping Injection Pressure, Extra Heat Transfer.

be enough fuel in the premixed burn. The diffusion burn is similar to the cases discussed earlier.

The final case for the Sandia engin e, seen in Figure 6.34, uses retarded injection timing. Here again the modeled ignition delay is accurate and as with the normal timing case at this speed the premixed burn is to small and the premixed duration is short compared with the data. In this ca se the modeled diffusion burn is much greater in magnitude and much shorter than the data shows. The model results for the retarded timing are similar to the normal timing results in shape and timing. In the engine the fuel burns slower at retarded timin g because of cooler in cylinder temperatures as the piston

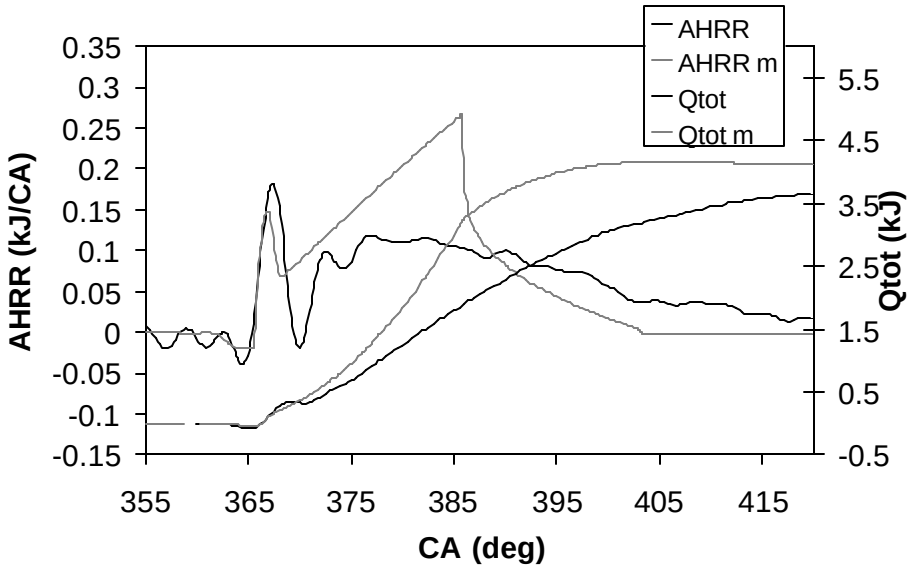


Figure 6.34. Sandia Engine Heat Release Comparison, 1680 RPM, High Fuel Load, Retarded Timing, Ramping Injection Pressure, Extra Heat Transfer.

moves down and the cylinder volume increases. The model does not duplicate this accurately.

6.6. BYU ENGINE COMPARISON

The results of the model for the BYU engine are generally less accurate. Figures 6.35 and 6.36 show results of the 1500-RPM low fuel load and the 2000-RPM high fuel load cases respectively. As with the Sandia engine cases the model results include ramping injection pressures and adjusted heat transfer.

The model ignition delay in these cases was too long. Because of this, the premixed burn was very fast and too large. The diffusion burn results are similar to the

results from the Sandia engine. The trends when the speed and fuel load are changed are the same as with the Sandia engine.

Part of the difficulty of producing accurate model results for this engine is a lack of engine information. The injection timing and injection duration are unknown on this engine. The nozzle diameter was measured but the coefficients C_a, C_d, and C_v are estimated using values for a similar nozzle taken from the literature. These examples show the loss in accuracy that can occur when less information is known about the engine.

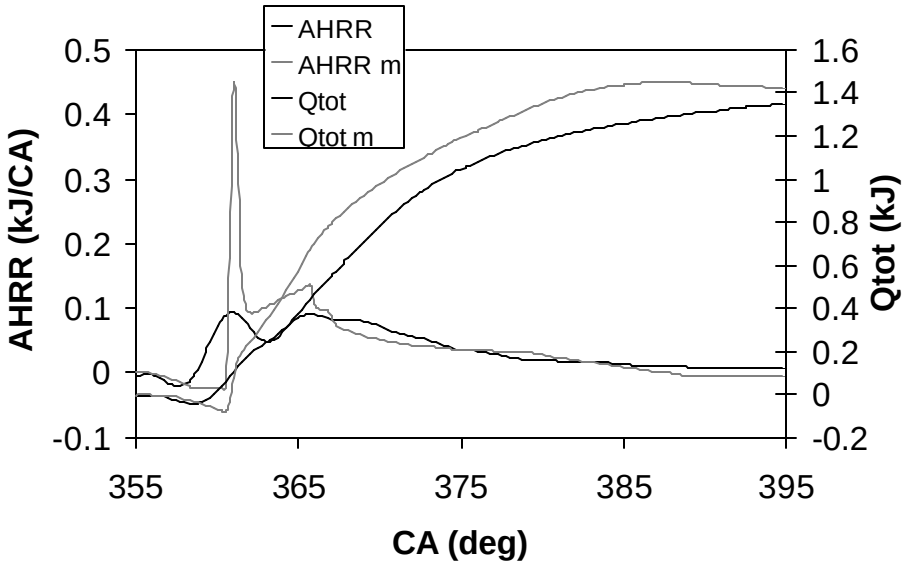


Figure 6.35. BYU Engine Heat Release Comparison, 1500 RPM, f = .3, **Ramping Injection Pressure, Extra Heat Transfer.**

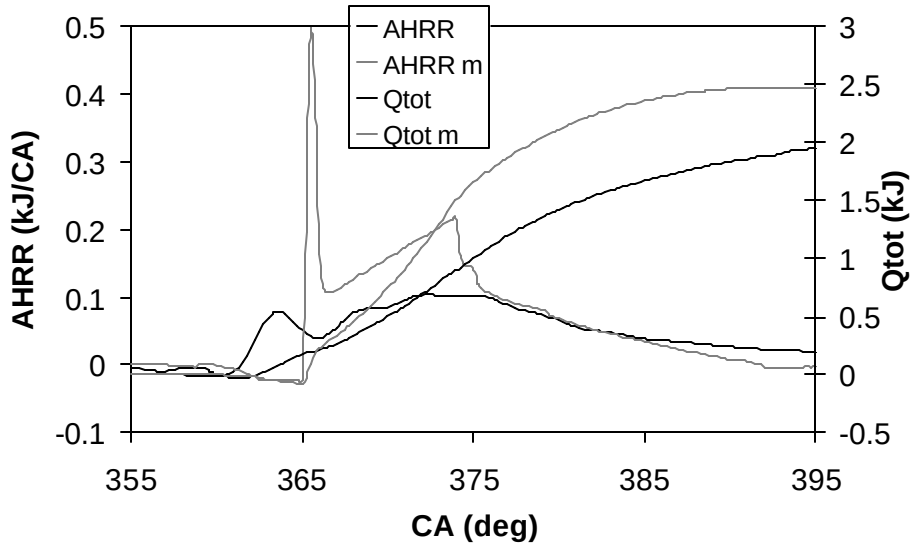


Figure 6.36. BYU Engine Heat Release Comparison, 2000 RPM, f = .5, **Ramping Injection Pressure, Extra Heat Transfer.**

6.7. ZONAL TEMPERATURES

The temperatures in each of the five zones are shown in Figure 6.37 for the Sandia engine 1200 RPM, low fuel load, normal timing case. The average temperature in the cylinder is also included. There is no data available to compare these temperatures with, except for flame temperatures (measured at approximately 2500 K), which should compare with the temperature of zone 4. Zone 4 temperatures produced by the model are overpredicted beginning at 2750 K and increasing to 2834 K at 360 crank angle degrees and reducing to 2650 at the end of combustion. Adiabatic flame temperature calculations overpredict temperatures due to the lack of heat transfer effects. The flame temperature

rises to 2834 K and falls to 2650 K following the temperature in the surrounding bulk gas (Zone 5), which is used to calculate the enthalpy of the reactants.

The zonal temperatures are important in pollutant formation, specifically NO_x and soot. The temperatures inside the spray (Zone 3) are important in the soot formation process. The temperature of the flame sheath surrounding the spray (Zone 4) is important in NO_x formation. While the heat release rate may not be completely accurate at this point, the temperatures of these zones should be very representative of those existing in the engine. Given the lack of experimental data, these temperatures become our best estimate of conditions in critical regions responsible for soot and NO_x formation.

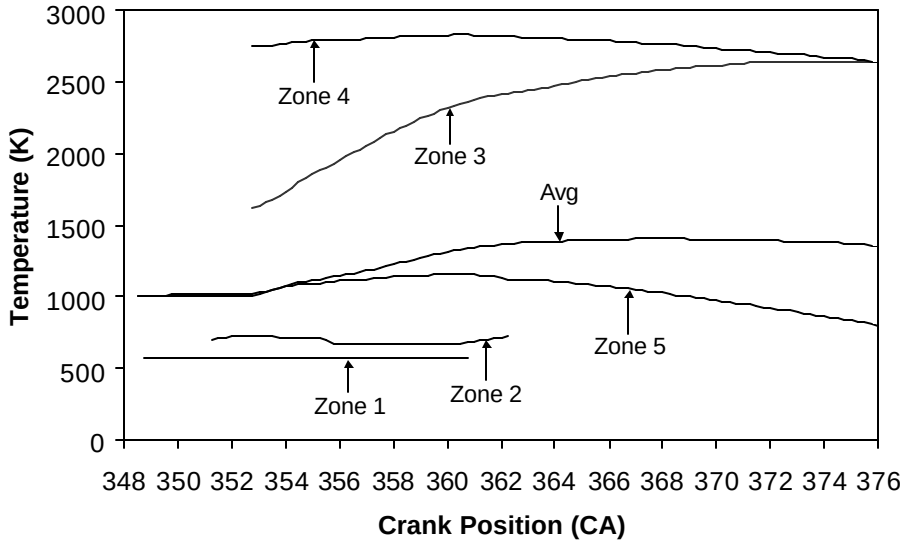


Figure 6.37. Zonal Temperatures vs. Crank Position

7. SUMMARY AND CONCLUSIONS

7.1. SUMMARY

A diesel cycle simulation code has been developed using C language. The code predicts the closed portion of the cycle including the compression, fuel injection auto-ignition, combustion and expansion processes. The model describes the temperature and pressure during compression and expansion using an isentropic process. The composition is also modeled using air and adding ideal combustion products during expansion.

There were two primary objectives of the model. The first was to apply the conceptual ideas of Dec [2] to a computer simulation. This conceptual model is based on recently reported optical diesel combustion measurements. This allowed the creation of five zones to model the quasi-steady burning diesel jet: 1) Liquid phase fuel 2) Vapor phase fuel 3) Rich products 4) Diffusion flame 5) Surrounding bulk gas. These five zones are of equal pressure but differing temperature and describe one burning fuel jet in the engine. The second objective was to investigate the use of a new mixing limited model for diesel combustion using a recently published spray penetration model. This spray model provides information on spray penetration and air entrainment to be used in defining the zones sizes and temperatures. Simplified ignition delay and premixed burn duration models were developed to complete the model.

To determine the temperature and composition of the rich products and the diffusion flame, an equilibrium code was developed. The equilibrium model uses a set of 21 gasseous species but does not include solid carbon due to added complexity though it occurs in significant amounts under certain conditions.

The results of the equilibrium code were compared with results from the NASA-Lewis code. The results matched over wide ranges of temperature and equivalence ratios common to diesel engines. There were discrepancies in two areas. At an equivalence ratio of 1 and temperatures below 1900K the equilibrium code solution did not converge to the NASA-Lewis solution due to the numerical method used. This does not affect the cycle simulation due to adiabatic flame temperatures of about 2700 K at an equivalence ratio of 1. Because the equilib rium code omits solid carbon as a possible product species the solution did not match the NASA-Lewis at equivalence ratios above 3.5. Nevertheless, the temperature of combustion products was predicted to within 3.5% over a large range of equivalence ratios ($.5 - 5$).

The combustion model produced the typical features of a diesel combustion event including an ignition delay period, a premixed burn spike, and a diffusion burn. The apparent heat release rate (AHRR) produced by the model was compared with data from a constant volume combustion vessel and real engines. Realizing that the shape of the AHRR from the data is altered when the data is averaged and smoothed complicates the comparison. The changes in ignition delay timing and the premixed burn when changing engine operating conditions followed the expected trends but were not predictive. Greater differences were seen when engine data such as injection timing and duration, and injector constants were unknown and had to be approximated. The shape of the

diffusion burn was similar for the combustion bomb cases although the heat release rose more rapidly for the model, reached its peak early and decreased faster after the end of injection. The shape of the heat release for the engine cases was significantly different than the data. For these cases the heat release also rose too rapidly and the peak was greater than the data. After the end of injection in the engines the heat release decreases in two phases. The first occurs between the end of injection and when the tip of the jet reaches the flame length. The second begins at the end of the first and ends when the last of the fuel injected reaches the flame length ending combustion. These two periods are not clearly identifiable in engine data.

The model also provides temperatures in each of the zones. The liquid fuel zone temperature is defined as the boiling temperature of the fuel, which is typically near 550 K. Once the fuel is vaporized the temperature rises until it ignites. The average temperature in the vapor fuel zone is approximately 700 K. At the start of combustion the average temperature of the rich premixed products is approximately 1600 K and increases to about 2600 K at the end of combustion when the equivalence ratio is near 1. The diffusion flame sheath begins at an adiabatic flame temperature of approximately 2700 K, peaks just over 2800 K, and reduces to approximately the temperature of the premixed products zone at the end of combustion. This adiabatic flame follows the temperature of the surrounding gas as it is heated from combustion and cools due to expansion of the cylinder. These temperatures are important in the formation of pollutants such as soot and NO_x.

7.2. CONCLUSIONS

The new diesel combustion model produced for this work provides a framework for a more accurate zero-dimensional model by providing physical zones within the model that correspond to real structures within a diesel engine. The approach of modeling diesel combustion through a mixing limited jet and determining the mixing limit through an empirical correlation of jet penetration shows promise but also exhibits some obvious shortcomings.

As discussed above the modeled heat release increases more rapidly and reaches its peak earlier than the data and the heat release reduces more quickly after the end of injection. This suggests that jet penetration is overpredicted in the model providing more air and a quicker burn rate. There are several explanations for this. First, the spray model was developed for a non-vaporizing, non-combusting fuel jet. Measurements indicate that the spray model overpredicts penetration in a vaporizing jet by as much as 18% [25]. Second, the spray model is sensitive to injector coefficients, particularly C_v. This could in part explain the greater discrepancy in the BYU engine results where injector coefficients were approximated using a similar injector but actual measurements were not taken. Constants in the spray model could be adjusted to reduce spray penetration but changing the values for these constants could not be justified without new data.

Once the flame length is reached the heat release rate matches the results from the combustion bomb cases where wall effects and jet interactions are not present. Therefore once the flame length is reached the model does a reasonably good job of predicting the air entrainment into an individual unobstructed jet and the assumption that combustion is

mixing limited also seems reasonable. In the combustion bomb cases where wall interaction was involved the limited data used in this work showed little influence of jet impingement. It seems reasonable that the spray penetration correlation is relatively accurate for wall interaction conditions.

In diesel engine the jet impinges on the wall soon after the premixed burn and usually after only 15 – 25 % of the fuel is burned. Because there are multiple nozzles in a typical engine injector, shortly after the jets reach the wall they begin to interact with adjacent jets. At this point the modeled and real AHRR are fundamentally different. This suggests that the air entrainment rate into the jet is different at this point and a different mixing model is needed.

The ignition delay and premixed burn sub-models were not a focus of this work but the simple models selected produced expected trends when compared with the data. Quantitatively, the ignition delay and premixed burn models were unable to match data from various engines. The proposed model should offer advantages in predicting the duration of the premixed burn because the model is kinetically controlled. The new zero-dimensional model should be able to provide better predictions of temperature to be used in the kinetics than previous zero-dimensional models. The premixed burn duration is, however, coupled to the ignition delay model and can appear inaccurate if the ignition delay model incorrectly predicts the mass of fuel available.

The ignition delay is very difficult to predict because non-uniformity in the spray is typical and has a large effect on when and where ignition occurs. The model proposed here has both a mixing (liquid length) and kinetic (temperature) component that could allow improved prediction of ignition delay, but the assumption that concentration and

temperature are uniform in the spray in the radial direction severly restrict the possibility of achieving large improvements in predicting ignition delay.

7.3. RECOMMENDATIONS

The mixing-limited model for diesel combustion shows promise but will require several improvements before being useful as a predictive tool for engine testing, development, and control. The spray penetration should be modified to account for vaporization effects and more accurate injector parameters should be obtained. A mixing model for the time period after jet interaction should be developed. A model predicting mixing based in the kinetic energy could be investigated as a first step. A lift-off length correlation is now available, which could be easily added to the model inplace of the constant included in the input file. Improvements to the ignition delay and premixed burn sub models sould be investigated and at a minimum, constants optimal for the range of data investigated could be used.

Future modeling work could involve coupling detailed kinetic mechanisms with the current code to predict soot and NO_x formation. The current model makes a time, temperature stoichiometry relationship available for important zones within the combustion process that are responsible for the formation and destruction of pollutants. It would be of interest to see if this zero-dimensional model could be used with detailed kinetics to produce a reasonable prediction.

REFERENCES

[1] Heywood, J. B. Internal Combustion Engine Fundamentals. McGraw-Hill, New York, 1988.

[2] Dec, J. E.: A Conceptual Model of Diesel Combustion Based on Laser-Sheet Imaging, SAE paper 970873, 1997.

[3] Flynn, P. F., Durrett, R. P., Hunter, G. L., zur Loye, A. O., Akinyemi, O. C., Dec, J. E., and Westbrook, C. K.: Diesel Combustion: An Integrated View Combining Laser Diagnostics, Chemical Kinetics, and Empirical Validation, SAE paper 1999-01-0509, 1999.

[4] Siebers, D. L.: Liquid-Phase Fuel Penetration in Diesel Sprays, SAE paper 980809, 1998.

[5] Espey, C., and Dec, J. E.: The Effect of TDC Temperature and Density on the Liquid-Phase Fuel Penetration in a D. I. Diesel Engine, SAE paper 952456, 1995.

[6] Kouremenos, D. A., Rakopoulos, and C. D., Hountalas, D. T.: Multi-Zone Combustion Modelling for the Prediction of Pollutants Emissions and Performance of DI Diesel Engines, SAE paper 970635, 1997.

[7] Foster, D. E.: An Overview of Zero-Dimensional Thermodynamic Models for IC Engine Data Analysis, SAE paper 852070, 1985.

[8] Primus, R. J., and Wong, V. W.: Performance and Combustion Modeling of Heterogeneous Charge Engines, SAE paper 850343, 1985

[9] Schihl, P. J., Atreya, A., Bryzik, W., and Schwarz, E.: Simulation of Combustion in Direct-Injection Low Swirl Heavy-Duty Type Diesel Engines, SAE paper 1999-01-0228, 1999.

[10] Miyamoto, T., Hayashi, A., Harada, A., Sasaki, S., Akagawa, H., and Tsujimura, K.: A Computational Investigation of Premixed Lean Diesel Combustion, SAE paper 1999-01-0229, 1999.

[11] Kong, S. C., Zhiyu, H., and Reitz, R.D.: The Development and Application of a Diesel Ignition and Combustion Model for Multidimensional Engine Simulation, SAE paper 950278, 1995.

[12] Whitehouse, N. D., and Way, R. J. B.: Simple Method for the Calculation of Heat Release Rates in Diesel Engines Based on the Fuel Injection Rate, SAE paper 710134, 1971.

[13] Woschni, G., and Anisits, F.: Experimental Investigation and Mathematical Presentation of Rate of Heat Release in Diesel Engines Dependent Upon Engine Operating Conditions, SAE paper 740086, 1974.

[14] Salem, H., El-Bahsasy, S. H., and Elbaz, M.: Prediction of the Effect of Injection Parameters on NOx Emission and Burning Quality in the Direct Injection Diesel Engine Using a Modified Multi-Zone Model, Proceedings of the Inst. Of Mech. Eng. Pt. D5 vol. 212, pg. 427, 1998.

[15] Gupta, A. K., Mehta, and Gupta, C. P.: Model for Predicting Air-Fuel Mixing and Combustion for Direct Injection Diesel Engine, SAE paper 860331, 1986.

[16] Lipkea, W. H., and Dejoode, A. D.: A Model of a Direct Injection Diesel Combustion System for use in a Cycle Simulation and Optimization Studies, SAE paper 870573, 1987.

[17] Nishida, K. and Hiroyasu, H.: Simplified Three-Dimensional Modelling of Mixture Formation and Combustion in a D.I. Diesel Engine, SAE paper 890269, 1989.

[18] Mehta, P. S., Singal, S. K., and Pundir, B. P.: A Comprehensive Simulation Model for Mixing and Combustion Characteristics of Small Direct Injection Diesel Engines, Proceedings of the Inst. Of Mech. Eng. Pt. D2 vol. 209, pg. 117, 1995.

[19] Hiroyasu, H. Nishida, K.: Fuel Spray Trajectory and Dispersion in a D.I. Diesel Combustion Chamber, SAE paper 890462, 1989.

[20] Stiesch, G., and Merker, G. P.: A Phenomenological Model for Accurate and Time Efficient Prediction of Heat Release and Exhaust Emissions in Direct-Injection Diesel Engines, SAE paper 1999-01-1535, 1999.

[21] Rakopoulos, C. D., Hountalas, D. T., and Agaliotis, N.: Application of a Multi-Zone Combustion Model for the Prediction of Large Scale Marine Diesel Engines Performance and Pollutants Emissions, SAE paper 1999-01-1227, 1999.

[22] Chiu, W. S., Shahed, S. M., and Lyn, W. T.: AA Transient Spray Mixing Model for Diesel Combustion, SAE paper 790128, 1976.

[23] Dent, J. C., and Mehta, P. S.: Phenomenological Combustion Model for a Quiescent Chamber Diesel Engine, SAE paper 811235, 1981.

[24] Kuo, T. W., Yu, R. C., and Shahed, S. M.: A Numerical Study of the Transient Evaporating Spray Process in the Diesel Environment, SAE paper 831735, 1983.

[25] Naber, J. D., and Siebers, D. L.: Effects of Gas Density and Vaporization on Penetration and Dispersion of Diesel Sprays, SAE paper 960034, 1996.

[26] Siebers, D. L.: Scaling Liquid-Phase Fuel Penetration in Diesel Sprays Based on Mixing-Limited Vaporization, SAE paper 1999-01-0528, 1999.

[27] Higgins, B. S., Mueller, C. J., and Siebers, D. L.: Measurement of Fuel Effects On Liquid-Phase Penetration in DI Sprays, SAE paper 1999-01-0519, 1999.

[29] Hardenberg, H. O., and Hase, F. W.: An Empirical Formula for Computing the Pressure Rise Delay of a Fuel from its Cetane Number and from the Relevant Parameters of Direct Injection Diesel Engines, SAE paper 790493, 1979.

[28] Olikara, C., and Borman, G. L.: A Computer Program for Calculating Properties of Equilibrium Combustion Products with Some Applications to I.C. Engines, SAE paper 750468, 1975.

[30] Chase, M. W.: "JANAF Thermochemical Tables," 4th Ed., American Chemical Society, Woodbury, N.Y. 1998.

[31] Press, M.W., Teukolsky, S.A., Vetterling, W.T., Flannery, B.P.: "Numerical Recipes in C," 2nd Ed., Cambridge University Press, New York, N.Y. 1994.

[32] Higgins, B., Siebers. D., and Aradi, A.: Diesel Spray Ignition and Premixed Burn Behavior, SAE paper 2000-01-0940, 2000.

[33] Turns, Stephen R. An Introduction to Combustion. 2nd ed. McGraw-Hill, Boston, 2000.

APPENDIX

APPENDIX A. CODE LISTING

Sample input file for the Sandia engine 1200 RPM low fuel load normal injection timing condition including ramping injection pressure.

```
.1397   B(m)
.1524   L(m)
.3048   l(m)
10.75   rc
192     Pin(kPa)
426     Tin(K)
348.5   SOI(TDC=360)
8       fuel
.00007189       mf(kg)
2       nozzle
8      #holes
15000  Pinj0
40000  Pinj(kPa)
1200    RPM
12      flameLO(mm)
.024    Apiston(m^3)
```

```
/*************************************************************************/
/* Diesel Cycle Simulation Code                                          */
/* Programmed by Rich Asay                                                  */
/* Version Completed Feb. 2003                                           */
/*                                                                       */
/* This code models the compression combustion and expansion processes in */
/* direct injection diesel engines.  The compression and expansion are    */
/* isentropic processes.  The combustion process uses the conceptual model */
/* by Jon Dec and the spray model by Dennis Siebers to define the spray and */
/* combustion zones.  An equilibrium model gives products of combustion.  */
/* The code also allows for constant volume combustion simulations        */
/*************************************************************************/

#include <stdlib.h>          // c function libraries
#include <stdio.h>
#include <math.h>
#include <conio.h>
#include <ctype.h>
#define PI 3.14159           // global constants
```

```
#define Ru 8.314
#define INFILE "cyclein.txt"                    // input and output file names
#define OUTFILE "cycleout.txt"

main()
{
        /* main function variable definitions

          B ---------- bore
          L ---------- stroke
          l ---------- rod length
          rc ---------- compression ratio
          Pin --------- intake pressure
          Tin --------- intake temperature
          SOI --------- start of injection
          phi --------- overall equivalence ratio
          Pinj -------- ending injection pressure
          Pinj0 ------- starting injection pressure (same as ending for constant injection pressure cases)
          RPM --------- engine speed
          Ap ---------- piston area
          name -------- # corresponding to fuel choice
          inj --------- # corresponding to injector choice
          holes ------- number of nozzles in the injector
          dummy[15] --- dumps units info from input file
          Vt ---------- total cylinder volume
          Vd ---------- displacement volume
          Vc ---------- clearance volume
          a ---------- crank radius
          A ---------- bore area for Vd calculation
          Rl ---------- rod length to crank radius ratio
          Sp ---------- mean piston speed
          Cr[3] ------- reference state for heat transfer model (ht)
          Cr[0] ---- reference temperature
          Cr[1] ---- reference pressure
          Cr[2] ---- reference volume
          Pm ---------- motored pressure for ht in combustion and expansion
          Tres -------- old residual temperature
          Tresn ------- new residual temperature
          Tin1 -------- first temperature with residual gas
          dTres ------- change in residual temperature i n loop
          k ---------- specific heat ratio
          h[21] ------- enthalpies of all species (see enthalpy function for definitions of each)
          Cp[21] ------ const pressure specific heats for species (see enthalpy function for
                        definitions of each)
          Cpa --------- total specific heat
          ma ---------- air moles
          mres -------- residual moles
          mt ---------- total moles of air and residual
          mf ---------- fuel mass or moles
          AFs --------- stoichiometric air fuel ratio
          Cv ---------- nozzle velocity coefficient
          w ----------- fuel flow rate constant
          Ufuel ------- fuel injection velocity
          Ainj -------- injector hole area
          mdotf ------- fuel flow rate
          mdotCA ------ fuel injected per 1/4 crank angle
```

```
  Lf ---------- lift-off length
  in1[9] ------ passes info between functions
    in1[0] --- cylinder temperature
    in1[1] --- cylinder pressure
    in1[2] --- cylinder volume
    in1[3] --- start of loop crank angle
    in1[4] --- end of loop crank angle
    in1[5] --- specific heat ratio
    in1[6] --- cylinder energy
    in1[7] --- motored pressure
    in1[8] --- motored temperature
  injdata[3] – carries nozzle specs (see injector function for definitions of each)
  fuel[7] ----- carries fuel properties (see properties function for definitions of each)
  fuelcomp[4] - carries fuel composition (see properties function for definitions of each)
  i ----------- loop control variable
  st ---------- stroke variable for isentropic function 0 for compression 1 for expansion */

// define variables
double B, L, l, rc, Pin, Tin, SOI, phi, Pinj, Pinj0, RPM, Ap;     // input variables
int name, inj, holes;
char dummy[15]; // space for description column in input file
double Vt, Vd, Vc, a, A, Rl, Sp, Cr[3]={0}, Pm=0;                 // engine geometry
double Tres=751, Tresn, Tin1=0, dTres=2, k, h[21], Cp[21], Cpa;
// thermodynamic variables
double ma, mres, mt, mf, AFs;       // mass variables
double Cv, w, Ufuel, Ainj, mdotf, mdotCA, Lf;          // injection variables
double in1[9], injdata[3], fuel[7], fuelcomp[4];          // program variables
int i=0, st;

FILE *input;                    // file pointers
FILE *output;

/* function prototypes
  isentropic function for compression and expansion processes
  injection function for fuel injection and combustion process
  enthalpy function retrieves enthalpy values for all species
  injector function retrieves injector parameters
  properties function retrieves fuel properties and composition
  airfuel function calculates molar stoichiometric air fuel ratio */

void isentropic(double in1[], double Vc, double rc, double Rl, double mt, FILE *output, int st,
          double phi, double fuel[], double fuelcomp[], double AFs, double Sp, double B,
          double Ap, double a, double l, double RPM, double Vd, double Cr[]);
void injection(double in1[], double Vc, double rc, double Rl, double fuel[], double fuelcomp[],
          double mt, double mf, FILE *output, double mdotCA, double Ufuel, double Sp,
          double RPM, int holes, int inj, double Lf, double B, double Vd, double a, double l,
          double Ap, double Pinj, double injdata[], double Ainj, double Pinj0);
void enthalpy(double T, double h[], double Cp[]);
void injector(int inj, double injdata[]);
void properties(int name, double fuel[], double fuelcomp[]);
double airfuel(double fuelcomp[]);

// open file
input=fopen(INFILE, "r");

if (input==NULL)
```

```
{
        printf("could not open input file\n");                  // check for input file
        return EXIT_FAILURE;    // end program if input file doesn't exist
}

// read input data
fscanf(input,"%lf %s",&B,&dummy);       // bore
fscanf(input,"%lf %s",&L,&dummy);       // stroke
fscanf(input,"%lf %s",&l,&dummy);       // rod length
fscanf(input,"%lf %s",&rc,&dummy);      // compression ratio
fscanf(input,"%lf %s",&Pin,&dummy);     // intake pressure
fscanf(input,"%lf %s",&Tin,&dummy);     // intake temperature
fscanf(input,"%lf %s",&SOI,&dummy);     // start of injection
fscanf(input,"%i %s",&name,&dummy);     // fuel choice
fscanf(input,"%lf %s",&mf,&dummy);      // fuel mass injected
fscanf(input,"%i %s",&inj,&dummy);      // injector choice
fscanf(input,"%i %s",&holes,&dummy);    // # holes in injector
fscanf(input,"%lf %s",&Pinj0,&dummy);   // starting injection pressure
fscanf(input,"%lf %s",&Pinj,&dummy);    // ending injection pressure
fscanf(input,"%lf %s",&RPM,&dummy);     // engine speed
fscanf(input,"%lf %s",&Lf,&dummy);      // lift-off length
fscanf(input,"%lf %s",&Ap,&dummy);      // piston area

// cylinder geometry calculations
A=PI/4*pow(B,2);            // bore area
Vd=A*L;             // displacement volume
if(rc==1)           // geometry for constant volume combustion bomb
{
        Vc=0;               // no clearance volume for constant V bomb
}
else
{
        Vc=Vd/(rc-1);               // clearance volume for engine cases
}
Vt=Vd+Vc;       // total volume
a=.5*L;  // crank radius
Rl=l/a;             // rod l to crank radius ratio
ma=Pin*Vd/(Ru*Tin);     // air mole calculation [kmol]
injector(inj,injdata);                  // get injector geometry and coefficients
Cv=injdata[1]/injdata[2];   // velocity coefficient Cd/Ca
Ainj=pow(injdata[0]/1000000,2)*PI/4;        // nozzle area [m^2]
properties(name,fuel,fuelcomp);     // retrieve fuel properties
mf=mf/fuel[7];    // fuel mole calculation
AFs=airfuel(fuelcomp);    // molar air/fuel ratio
Sp=2*RPM*L/60;          // mean piston speed

// loop calculates residual gas temperature
do
{
        output=fopen(OUTFILE, "w");     // open output file
        fprintf(output,"theta       T(K)    P(kPa)  V(dm^3)         h(kJ/kg)
                E1(kJ/kg)       E2(kJ/kg)       liquid L(mm)    Tad(K)  k\n");    // output
                file header

        mres=Vc*Pin/(Ru*Tres);  // residual moles [kmol]
        mt=ma+mres;     // total moles during compression [kmol]
```

```
		phi=mf*AFs/mt;		// overall equivalence ratio
		Tin1=(Tin*ma+Tres*mres)/mt;	// initial temp

		// compression process
		in1[0]=Tin1;	// initialize variables to be passed to isentropic function for
				compression process
		in1[1]=Pin;
		in1[2]=Vt;
		in1[3]=180;
		in1[4]=SOI;
		st=0;	// st=0 for compression
		isentropic(in1,Vc,rc,Rl,mt,output,st,phi,fuel,fuelcomp,AFs,Sp,B,Ap,a,l,RPM,Vd,Cr);
			// compression process calculated using isentropic function

		// injection and combustion process
		w=2*((Pinj-in1[1])*1000/fuel[3]);		// fuel flow rate calculations
		Ufuel=Cv*sqrt(w);		// injection velocity
		mdotf=injdata[2]*Ainj*fuel[3]*Ufuel/1000;		// fuel flow rate [kg/ms]
		mdotCA=holes*mdotf/(.006*RPM*fuel[7]);		// fuel flow rate [kmol/CA]
		in1[3]=SOI;	// initialize for start of injection and combustion process

		injection(in1,Vc,rc,Rl,fuel,fuelcomp,mt,mf,output,mdotCA,Ufuel,Sp,RPM,holes,inj,Lf,B,
		Vd,a,l,Ap,Pinj,injdata,Ainj,Pinj0);		// injection function called to calculate
							injection and combustion process
		// expansion process
		in1[4]=540.25;		// initialize to tell expansion when to stop
		mt=mt+mf;	// total moles in expansion process
		st=1;	// st=1 for expansion
		isentropic(in1,Vc,rc,Rl,mt,output,st,phi,fuel,fuelcomp,AFs,Sp,B,Ap,a,l,RPM,Vd,Cr);
			// expansion process calculated using isentropic function

		// residual gas temperature calculations
		enthalpy(in1[0],h,Cp);	// retrieve enthalpies at end of expansion process
		Cpa=(Cp[7]+3.7274*Cp[10]+. 0444*Cp[11])/(1+3.7274+.0444);		// specific heat of
			air at end of expansion
		k=Cpa/(Cpa-Ru);		// specific heat ratio
		Tresn=in1[0]*pow(Pin/in1[1],(k-1)/k);	// calculates new residual temp using
			isentropic process
		dTres=Tresn-Tres;		// change in residual temp from last guess
		Tres=Tresn;	// update Tres
		if(Vc==0)		// for bomb cases no Tres calculations
		{
			dTres=1;		// dTres set below tolerance for continuing loop
		}
		fclose(output);	// close output file (opening and closing in each loop insures only the
			final set is saved)
	}while(fabs(dTres)>2);	// when residual temp change is small end loop
	fclose(input);	// close input file

	return EXIT_SUCCESS;		// end program successfully
}
/*********************************************************************/
/* function calculates properties at each crank angle                */
/* using a isentropic process for compression                        */
/* and expansion strokes                                             */
/*********************************************************************/
```

```
void isentropic(double in1[], double Vc, double rc, double Rl, double mt, FILE *output, int st, double phi,
double fuel[], double fuelcomp[], double AFs, double Sp, double B, double Ap, double a, double l, double
RPM, double Vd, double Cr[])
{
        /* isentropic function variable definitions

          theta ------- crank angle
          j ----------- CA loop counter
          S ----------- starting crank angle for loops
          E ----------- ending crank angle for loops
          dur --------- crank angle duration for compression or expansion stroke
          T ----------- cylinder temperature
          Ta1 --------- first temperature guess for secant method
          Ta2 --------- second temperature guess for secant method
          Ta3 --------- updated temperature guess for secant method
          Tm ---------- motored temperature for ht in expansion stroke
          Tr ---------- reference temperature for ht
          P ----------- cylinder pressure
          Pi ---------- new cylinder pressure
          Po ---------- old cylinder pressure
          Pa3 --------- new pressure from temperature after secant method
          Pr ---------- reference pressure for ht
          Vf ---------- new volume
          vr ---------- volume ratio
          Vi ---------- old volume
          Vr ---------- reference volume for ht
          n ----------- # C atoms in fuel molecule
          m ----------- # H atoms in fuel molecule
          f ----------- # O atoms in fuel molecule
          g ----------- # N atoms in fuel molecule
          r[4] -------- simplifying constants for stoichiometry
          Nsum -------- total number of moles
          MWa ---------molecular weight of air or air and products
          x[21] ------- mole fractions of all species
          ha ---------- enthalpy of air or air and products units
          h[21] ------- enthalpies of all species (see enthalpy function for definitions of each)
          Cp[21] ------ const pressure specific heats for species (see enthalpy function for definitions of
                        each)
          Cpa --------- specific heat of air or air and products
          k ----------- specific heat ratio
          Ecyl -------- cylinder energy calculated from E + P*dV
          Ecyl2 ------- cylinder energy calculated from temperatures
          hc ---------- heat transfer function
          q0 ---------- secant function at first guess updated in subsequent loops
          q1 ---------- secant function at second guess updated in subsequent loops
          E1 ---------- energy in secant function at first guess updated in subsequent loops
          E2 ---------- energy in secant function at second guess updated in subsequent loops
          dq ---------- change in secant function
          Ach --------- surface area in the cylinder
          sl ---------- variable used in area calculation
          C1 ---------- constant in heat transfer correlation
          C2 ---------- constant in heat transfer correlation
          w ----------- variable in heat transfer model
          km ---------- specific heat ratio for motored calculation
          MW[] -------- molecular weights of all species
```

```
  hf[] -------- heats of formation of all species
  i ----------- loop counter
  ii ---------- loop counter */

// declare variables
double theta, j, S, E, dur;                 // loop variables
double T, Ta1, Ta2, Ta3, Tm=1, Tr=0;        // temperature variables
double P, Pi, Po, Pa3, Pr=1, Pm, Vf=0, vr, Vi, Vr=1;            // Pressures and volumes
double n, m, f, g, r[4], Nsum, MWa=0, x[21]={0};                // composition variables
double ha=0, h[21], Cp[21], Cpa, k, Ecyl=0, Ecyl2, hc;          // energy variables
double q0, q1, E1, E2, dq, Ach, sl, C1=2.28*Sp, C2=.00324, w, km;     // energy and heat tran
                                                                      variables
double MW[]={1.00794, 15.9994, 14.00674, 2.0159, 17.0079, 28.0104, 30.006, 31.999, 18.016,
             44.011, 28.013, 39.948, 15.035, 16.043, 26.038, 28.054, 16.023, 17.030, 27.026, 29.019,
             25.030};// molecular weights
double hf[]={217977, 249197, 472629, 0, 38985, -110541, 90297, 0, -241845, -393546, 0, 0,
             145687, -74873, 226748, 52283, 167653, -45898, 135143, 12134, 476976};
            // heats of formation
int i=0, ii;            // loop counters

/* function prototypes
   enthalpy function retrieves enthalpy values for all species */
   void enthalpy(double T, double h[], double Cp[]);

// initialize variables using inputs from main
T=in1[0];           // state variables
P=in1[1];
Vi=in1[2];
S=in1[3];           // CA loop controls
E=in1[4];
Ecyl=in1[6];        // energy
Pm=in1[7];          // motored state
Tm=in1[8];
Tr=Cr[0];           // reference state for ht
Pr=Cr[1];
Vr=Cr[2];
Po=P;    // old pressure
dur=(E-S)*4;        // # of steps to loop through process
n=fuelcomp[0];              // fuel composition
m=fuelcomp[1];
f=fuelcomp[2];
g=fuelcomp[3];

// initial specific heat ratio and energy calculation for compression stroke (st=0)
if (st==0)
{
          x[7]=.209126;     // mole fractions for air
          x[10]=.781589;
          x[11]=.009258;
          enthalpy(T,h,Cp);           // enthalpies and specific heats of species at start of
                                      compression
          Cpa=x[7]*Cp[7]+x[10]*Cp[10]+x[11]*Cp[11];         // specific heat of air
          k=Cpa/(Cpa-Ru);             // specific heat ratio
          ha=x[7]*h[7]+x[10]*h[10]+x[11]*h[11];     // air enthalpy
          MWa=x[7]*MW[7]+x[10]*MW[10]+x[11]*MW[11];           // molecular weight of air
          mt=mt*MWa;      // mass of air [kg]
```

```
        Ecyl=mt*ha/MWa-Po*Vi;           // energy in cylinder to start compression
}
// initial specific heat ratio and energy calculation for expansion stroke (st=1)
if (st==1)
{
        r[0]=(n+0.25*m-0.5*l)/phi;      //r0    // stoichiometry constants
        r[1]=0.5*l+r[0];                 //r
        r[2]=0.5*g+3.7274*r[0];          //r'
        r[3]=0.0444*r[0];               //r''
        Nsum=m/4+r[2]+r[3]+r[0]+l/2;    // total number of moles
        x[7]=(r[0]+l/2-n-m/4)/Nsum;     // O2    // mole fractions of product species
        x[8]=(m/2)/Nsum;                // H2O
        x[9]=n/Nsum;                    // CO2
        x[10]=r[2]/Nsum;                // N2
        x[11]=r[3]/Nsum;                // Ar
        enthalpy(T,h,Cp);           // specific heats of species at start of expansion (energy from
                                        injection function)
        for(Cpa=0,ii=0;ii<21;ii++)
        {
                Cpa=Cpa+x[ii]*Cp[ii];   // calculate total specific heat
                MWa=MWa+x[ii]*MW[ii];       // calculate total molecular weight
        }
        mt=(mt+phi*mt/AFs)*MWa;         // total mass in expansion
        k=Cpa/(Cpa-Ru);             // specific heat ratio
}

// calculate properties at each 1/4 CA
for(i=0; i<dur; i++)            // loops from start CA to end CA
{
        j=i;    // loop counter as double instead of int for CA calculation
        theta=(S+j/4)*PI/180;       // CA in radians
        Vf=Vc*(1+.5*(rc-1)*(Rl+1-cos(theta)-pow(Rl*Rl-(sin(theta)*sin(theta)),.5)));
                // volume at end of step
        if(Vc==0)
        {
                Vf=Vi;  // new volume equal old volume for constant volume case
        }
        vr=Vi/Vf;           // volume ratio for step
        sl=a*cos(theta)+pow((l*l-a*a*sin(theta)*sin(theta)),.5);        // calculate sl variable fore
                                                                        area equation
        Ach=Ap+PI*B*B/4+PI*B*(l+a-sl);          // cylinder area for ht
        if(Vc==0)
        {
                Ach=Ap;         // area equal to input file area for constant volume case
        }
        Pi=P;   // save initial pressure
        T=T*pow(vr,(k-1));          // new temperature and pressure using isentropic process
        P=P*pow(vr,k);
        if (st==0)          // enthalpy, specific heat ratio, and energy for each step in compression
                               stroke
        {
                enthalpy(T,h,Cp);
                ha=x[7]*h[7]+x[10]*h[10]+x[11]*h[11];
                Cpa=x[7]*Cp[7]+x[10]*Cp[10]+x[11]*Cp[11];
                k=Cpa/(Cpa-Ru);
                w=C1;   // value for ht correlation
```

```
        Ecyl2=mt*ha/MWa-(Pi+P)/2*Vi;
}
if (st==1)          // enthalpy, specific heat ratio, and energy for each step in expansion
                       stroke
{
        enthalpy(T,h,Cp);
        for(Cpa=0,ha=0,ii=0;ii<21;ii++)
        {
                Cpa=Cpa+x[ii]*Cp[ii];
                ha=ha+x[ii]*(h[ii]);
        }
        k=Cpa/(Cpa-Ru);
        Ecyl2=(mt)*ha/MWa-(Pi+P)/2*Vi;
        enthalpy(Tm,h,Cp);
        for(Cpa=0,ii=0;ii<21;ii++)
        {
                Cpa=Cpa+x[ii]*Cp[ii];     // specific heat at motored conditions
        }

        km=(Cp[7]+3.7274*Cp[10]+.0444*Cp[11])/(1+3.7274+.0444)/((Cp[7]+3.7274*
                Cp[10]+.0444*Cp[11])/(1+3.7274+.0444)-Ru);
                // specific heat ratio at motored conditions
        Tm=Tm*pow(vr,(km-1));           // new motored Temp and Press
        Pm=Pm*pow(vr,km);
        w=C1+C2*Vd*Tr/(Pr*Vr)*(P-Pm);           // value for ht correlation
}
Ecyl=Ecyl-(Pi+P)/2*(Vf-Vi);         // new energy
hc=(3.26*pow(B,-.2)*pow(P,.8)*pow(T,-.55)*pow(w,.8)/1000)*Ach*
        (T-600)*(.25/(.006*RPM))/1000;    // heat transfer correlation
//hc=0;  // zeros heat transfer correlation - can be uncommented to neglect heat transfer

// calculate new temperature and pressure after heat transfer
Ta1=T;          // temperature guesses for secant method
Ta2=T-50;
enthalpy(Ta1,h,Cp);         // secant method using enthalpy to get avg Temperature
for(E=0,ii=0;ii<21;ii++)
{
        E=E+x[ii]*h[ii];  // enthalpy at first temp guess
}
q0=(ha*mt/MWa-hc)*MWa/mt-E;  // calculate function at first temp - trying to zero
enthalpy(Ta2,h,Cp);
for(E1=0,ii=0;ii<21;ii++)
{
        E1=E1+x[ii]*h[ii];          // enthalpy at first temp guess
}
q1=(ha*mt/MWa-hc)*MWa/mt-E1;          // calculate function at second temp - trying
                                        to zero
do
{
        dq=(q1-q0);         // change in function between temp guesses
        Ta3=Ta2-q1*((Ta2-Ta1)/dq);         // new temperature guess
        Ta1=Ta2;          // update values for next guess
        q0=q1;
        Ta2=Ta3;
        enthalpy(Ta3,h,Cp);
        for(E2=0, ii=0;ii<21;ii++)
```

```
                        {
                                E2=E2+x[ii]*h[ii];          // enthalpy at subsequent temp guesses
                        }
                        q1=(ha*mt/MWa-hc)*MWa/mt-E2;          // calculate function at
                                                              subsequent temps - trying to zero
                }while(fabs(q1)>.00001);              // when function is small exit loop
                Pa3=P*Ta3/T;    // new pressure after heat transfer
                T=Ta3;          // update cylinder pressure and temperature
                P=Pa3;

                fprintf(output,"%.2lf       %lf       %lf       %lf       %lf       %lf       %lf       %lf
                        %lf       %lf\n",S+j/4,T,P,Vf*1000,ha,Ecyl,Ecyl2,g,g,hc);      // print output file
                                                                                       for step
                printf("CA=%.2lf\n",S+j/4);          // print current crank angle to screen
                Vi=Vf;              // update volume
        }       // end of CA for loop
        in1[0]=T;           // update variables to send to main
        in1[1]=P;
        in1[2]=Vf;
        in1[4]=S+j/4;
        in1[5]=k;
        in1[6]=Ecyl;
        Cr[0]=T;
        Cr[1]=P;
        Cr[2]=Vf;

        return;             // end function and return to main function
}
/*-------------------------------------------------------------------*/
/* function calculates properties during fuel injection                                 */

void injection(double in1[], double Vc, double rc, double Rl, double fuel[], double fuelcomp[], double mt,
double mf, FILE *output, double mdotCA, double Ufuel, double Sp, double RPM, int holes, int inj, double
Lf, double B, double Vd, double aa, double la, double Ap, double Pinj, double injdata[], double Ainj,
double Pinj0)

{
        /* injection function variable definitions
          theta ------- crank angle
          j ----------- loop counter for crank angle
          S ----------- starting crank angle
          rhoa -------- air density for spray model (only calculated at SOI)
          no1[21] ----- mole fraction guess for equilibrium - when stepping through zone 3 guess for new
                        step is old step
          Pinj1 ------- injection pressure at current CA
          phia -------- equivalence ratio past flame length for heat release model
          durCA ------- injection duration in crank angles
          AFs --------- stoichiometric air fuel ratio
          phi --------- equivalence ratio up to flame length for heat release model
          rst[6] ------ simplifying constants for st oichiometry for heat release model
          Nsum -------- total number of moles
          x3fuel ------ mole fraction of fuel in zone 3 before combustion begins for premixed burn
                        duration
          ll[10] ------ array passed to spray model
            ll[0] ---- liquid length
            ll[1] ---- equivalence ratio at liquid length (molar)
```

ll[2] ---- spray length at given time
ll[3] ---- equivalence ratio at x (molar)
ll[4] ---- time to reach liquid length
ll[5] ---- spray spread angle
ll[6] ---- time to reach given length
ll[7] ---- equivalence ratio at given length (mass)
ll[8] ---- distance to back of spray after EOI
ad[2] ------- array passed to adiabatic flame temp calculator for flame sheath phi=1
ad[0] ---- flame temperature
ad[1] ---- product enthalpy
ad3[2] ------ array passed to adiabatic flame temp calculator for zone 3 kept separate for equilibrium code and temperature guesses
C ---------- # C atoms in fuel molecule
H ---------- # H atoms in fuel molecule
O ---------- # O atoms in fuel molecule
N ---------- # N atoms in fuel molecule
LL ---------- liquid length
LS ---------- spray penetration length
L2 ---------- end of zone 2 length
rxe --------- spray radius at xe
xe ---------- back end of spray after EOI
Lstoic ------ flame length (phi=1)
rL2 --------- spray radius at L2
rLstoic ----- spray radius at Lstoic
LL1 --------- new liquid length when calculating liquid length with lift-off length in place
LLd --------- change in liquid length when it moves back with lift-off length
rL ---------- spray radius at liquid length
rLf --------- spray radius at lift-off length when liquid length moves back with lift-off length
r ---------- spray radius at step for zone 2
dLf --------- change in lift-off length
Lfa --------- lift-off length that moves back to given Lf
dLL --------- change in liquid length when calculating liquid length with lift-off length in place
rS ---------- spray radius at penetration length
rLfa -------- spray radius at penetration length
an ---------- spray angle multiplier from Siebers =.66
angle ------- spray spread angle
dur --------- injection duration (ns)
te ---------- time at EOI
tstoic ------ time to reach flame length
tlf --------- time lift-off length reaches minimum
time0 ------- time at beginning of slice when stepping through zone 3 (for fuel mass calculation)
tid --------- ignition delay time
t ---------- time after SOI
dtCA -------- time for 1/4 CA step
A2 ---------- constant in exponential that burns back zone 2
r0 ---------- radius at beginning of slice while stepping through zone 3
r1 ---------- radius at end of slice while stepping through zone 3
l0 ---------- length to beginning of slice while stepping through zone 3
l1 ---------- length to end of slice while stepping through zone 3
steps ------- # steps taken through zones 2 and 3
t1 ---------- starting length for stepping through zones 2 and 3
t2 ---------- ending length for stepping through zones 2 and 3
ts ---------- step size
P ---------- cylinder pressure
cfuel ------- fuel concentration for premixed burn duration
cO2 --------- oxygen concentration for premixed burn duration

sl1 --------- slope of oxygen concentration for premixed burn duration
sl2 --------- slope of temperature for premixed burn duration
Tpm --------- temperature in premixed burn durat ion model
timepm ------ premixed burn time
dfuel ------- change in fuel concentration for premixed burn duration
cO2a -------- change in oxygen concentration for premixed burn duration
Tdel -------- temperature for ignition delay model =T at L2
T ---------- cylinder temperature
Tinj -------- fuel injection temperature
Tsm --------- spray model temperature (constant at cylinder value at SOI)
Tad --------- adiabatic flame temperature fore flame sheath
T2 ---------- zone 2 temperature
T2o --------- stores last zone 2 temperature before zone disappears
Ttot -------- mass avg temperature for whole cylinder
trf --------- temperature constant for fuel enthalpy curve fit
T5 ---------- zone 5 temperature
T2r --------- temperature at the end of zone 2 for premixed burn duration
T3p --------- temperature of the products in first step of zone 3 for premixed burn duration
phi23 ------- equivalence ratio at the line between zone 2 and 3
AF23 -------- air fuel ratio at the line between zone 2 and 3
h2r --------- enthalpy at the end of zone 2 for premixed burn duration
hpm --------- enthalpy in the premixed burn
Tx2 --------- temperature at a position in zone 2 while stepping through
T1m ---------- mass times temperature for zone 1
T2m --------- mass times temperature for zone 2
T3m --------- mass times temperature for zone 3
Tme --------- mass times temperature for volume of zone 5 where spray would be after EOI
T2in -------- temperature of air coming into zone 2 from zone 5
Ta1 --------- first temperature guess for new cylinder temperature using total energy (secant method)
Ta2 --------- second temperature guess for new cylinder temperature using total energy (secant method)
Ta3 --------- subsequent temperature guesses for new cylinder temperature using total energy (secant method)
Vi ---------- initial cylinder volume
Vf ---------- final cylinder volume
vf ---------- spray volume up to lift-off length for moving back lift-off length
v ----------- step volume for stepping through zones 2 and 3
vS ---------- spray volume
q0 ---------- secant function at first guess updated in subsequent loops
q1 ---------- secant function at second guess updated in subsequent loops
dq ---------- change in secant function
V5 ---------- zone 5 volume using ideal gas
v1 ---------- zone 1 volume
v2 ---------- zone 2 volume
v12 --------- zone 1 and 2 volume
v2a --------- spray vol through zone 2 used to calculate zone 2 volume
ve ---------- volume of back end of spray after EOI
v3 ---------- zone 3 volume
mp ---------- mass of products in zone 2
ms5 --------- mass in zone 5
me ---------- mass in back end of spray after
m1b --------- air mass beyond lift-off length in zone 1 for moving liquid length back with lift-off length
ma1b -------- unburned air mass beyond lift-off length in zone 1 for moving liquid length back with lift-off length

mp1b -------- burned air mass beyond lift-off length in zone 1 for moving liquid length back with lift-off length
m1 ---------- mass in zone 1
m2 ---------- mass in zone 2
m3 ---------- mass in zone 3
mas --------- mass of air for 1 kg fuel for phi=1 diffusion flame temperature calculation
m2min ------- minimum mass in zone 2 as it burns back
ma ---------- mass of air for stepping through zone 2
ma1a -------- air mass before lift-off length in zone 1 for moving liquid length back with lift-off length
m ----------- mass of air for stepping through zone 3
m5in -------- mass of air in zone 3 from zone 5 for stepping
mS ---------- mass of air in the spray
mo ---------- air mass in last step while stepping through zone 2
ms3[21] ----- number of moles of each species in zone 3
ms5a -------- total mass of air in cylinder at SOI
m3a --------- mass of air in zone 3
mp1 --------- mass of air in products in zone 1
ma1 --------- mass of air in zone 1
m2a --------- mass of air in zone 2
m2p --------- mass of air in products in zone 2
m2ain ------- mass of air in zone 3 from zone 2 for stepping
mdotf ------- mass flow rate of fuel in kg/ms
mfuel ------- total mass of fuel
mfS --------- mass of fuel in the spray
mfuelo ------ fuel mass in last step while stepping through zone 2
mfdt -------- molar flow rate of fuel kmol/ms
mfp1 -------- mass of fuel in products in zone 1
mf1 --------- mass of fuel in zone 1
mfp --------- mass of fuel in products in zone 2 for stepping through zone
mf2 --------- mass of fuel in zone 2
mf3 --------- mass of fuel in zone 3
mf2u -------- mass of unburned fuel in zone 2
mf1u -------- mass of unburned fuel in zone 1
mf2p -------- mass of fuel in products in zone 2
m1fv -------- mass of vaporized fuel in zone 1
h4 ---------- enthalpy of zone 4
dh4 --------- enthalpy change of diffusion flame products from zone 4 temp to zone 1 temp
dh5 --------- enthalpy change of air at zone 5 temp to zone 1 temp
dh ---------- enthalpy change of air for liquid length calculation
h[21] ------- enthalpies of all species
hf4 --------- enthalpy of zone 4 for moving liquid length with lift-off length
hf5 --------- enthalpy of zone 5 for moving liquid length with lift-off length
hr ---------- enthalpy of reactants for adiabatic flame temp in zone 4
h5[21] ------ enthalpies of species in zone 5
deltadh ----- change in dh when moving liquid length with lift-off length loop
dho --------- old enthalpy change for moving liquid length with lift-off length loop
h5a --------- enthalpy of air in zone 5
h1a --------- enthalpy of air in zone 1
h1p --------- enthalpy of products in zone 1
h2a --------- enthalpy of air in zone 2
h2p --------- enthalpy of products in zone 2
h3p --------- enthalpy of products in zone 3
h1[21] ------ enthalpies of species when enthalpies at 2 temps required for a calculation
hr1 --------- enthalpy of reactants for adiabatic flame temp in zone 3
hfuel ------- enthalpy of the fuel

Cp[21] ------ specific heats of all species
Cpas -------- specific heat of air at SOI
Cpp --------- specific heat of products of diffusion flame
E1 ---------- energy in zone 1 (also used in secant method to get cylinder temperature)
E2 ---------- energy in zone 2 (also used in secant method to get cylinder temperature)
E3 ---------- energy in zone 3
E5 ---------- energy in zone 5
Ecyl2 ------- cylinder energy calculated using energies of zones added
Ecyl -------- cylinder energy calculated using E+PdV
EA ---------- activation energy
E ----------- zone 5 energy used in secant method to get cylinder temperature
x4[21] ------ mole fractions of diffusion flame products
x5[21] ------ mole fractions in zone 5
x3[21] ------ mole fractions in zone 3
x3a[21] ----- mole fractions in spray beyond lift-off length before flame length
x3b[21] ----- mole fractions in spray beyond flame length
MW4 --------- molecular weight in zone 4
MW2 --------- molecular weight in zone 2
MWa --------- molecular weight of air
MW3 --------- molecular weight in zone 3
MWP --------- molecular weight of products from lift-off length to flame length for heat release calculation
MWR --------- molecular weight of reactants from lift-off length to flame length for heat release calculation
MW[] -------- molecular weights of all species
hf[] -------- heats of formation of all species
HP ---------- enthalpy of products from lift-off length to flame length for heat release calculation
HR ---------- enthalpy of reactants from lift-off length to flame length for heat release calculation
U1 ---------- old internal energy used in secant method to get cylinder temperature
U2 ---------- new internal energy used in secant method to get cylinder temperature
Qn ---------- new heat release
Qo ---------- old heat release
Qstep ------- heat release in 1/4 CA step
Ea ---------- spray energy used in secant method to get cylinder temperature 1st guess
Ea1 --------- spray energy used in secant method to get cylinder temperature 2nd guess
Ea2 --------- spray energy used in secant method to get cylinder temperature subsequent guesses
mstoic ------ mass of air beyond flame length
mfstoic ----- mass of fuel beyond flame length (in products with mstoic)
hT ---------- enthalpy at temp at SOI for fuel vaporization effect on heat release
hTs --------- enthalpy at vaporization temp for fuel vaporization effect on heat release
Qs ---------- fuel vaporization effect on heat release
mfn --------- new fuel mass in spray for heat release calculations
mfo --------- old fuel mass in spray for heat release calculations
mfst -------- change in fuel mass in spray for heat release calculations
Qid --------- heat release at ignition delay
tq ---------- premixed burn time counter (CA)
tqt --------- total premixed burn time (CA)
m1c --------- new air in spray in CA step until reaching lift off length for fuel vaporization effect on heat release
ma1p -------- mass of air in products in zone 1
tlfq -------- time to reach lift-off length for heat release calculation
mfq --------- mass of fuel from lift-off length to flame length for heat release calculation
maq --------- mass of air from lift-off length to flame length for heat release calculation
rlf --------- spray radius at lift-off length
MWPs -------- molecular weight of products beyond flame length for heat release calculation
HRs --------- enthalpy of reactants beyond flame length for heat release calculation

```
  HPs ---------- enthalpy of products beyond flame length for heat release calculation
  Vsq ---------- actual spray volume
  maqs --------- mass of air beyond flame length for heat release calculation
  mfqs --------- mass of fuel beyond flame length for heat release calculation
  xsum --------- sum of mole fractions
  Cp5 ---------- specific heat of zone 5
  Cp3 ---------- specific heat of zone 3
  Cpt ---------- average specific heat of all zones
  Cpf ---------- specific heat of the fuel
  xfuelpma ----- starting mole fraction of fuel in premixed burn model
  xfuelpm ------ mole fraction of fuel in premixed burn model
  xafbpm ------- mole fraction of products in premixed burn model
  xairpm ------- mole fraction of air in premixed burn model
  sl ----------- variable used in area calculation
  Ach ---------- surface area in the cylinder
  hc ----------- heat transfer
  w ------------ variable in heat transfer model
  C1 ----------- constant in heat transfer correlation
  C2 ----------- constant in heat transfer correlation
  Tr ----------- reference temperature in heat transfer model
  Pr ----------- reference pressure in heat transfer model
  Vr ----------- reference volume in heat transfer model
  Tm ----------- motored temperature in heat transfer model
  Cpa ---------- specific heat of air
  k ------------ specific heat ratio
  vr ----------- volume ratio used to get motored data
  ww ----------- fuel flow rate constant
  Pm ----------- motored pressure in heat transfer model
  i ------------ loop counter
  c ------------ loop counter
  z ------------ flag insures minimum mass in zone 2 only calculated once
  y ------------ flag insures calculations to move lift-off length done once
  sto ---------- flag insures calculations for flame length done once
  yy ----------- flag insures calculations for minimum liquid length done once
  flag1 -------- tells equilibrium code weather to make a new initial guess
  flag2 -------- tells adiabatic flame calculation weather to start at hard coded temp guesses or use
                 temperature from last aft calculation
  ii ----------- loop counter
  flag6 -------- insures premixed burn duration calculation only done once
  flag7 -------- changes direction of steps through zone 3 to help equilibrium solution
  flag8 -------- insures calculations for heat release for premixed burn done once
  flag9 -------- other calculations for changing direction of steps through zone 3 to help equilibrium
                 solution
  flag10 ------ saves temperature for ignition delay model
  flag11 ------ insures ignition delay calculated once */

// declare variables
double theta, j=1, S, rhoa, no1[21]={0}, Pinj1, phia=0, durCA;
double AFs, phi=0, rst[6]={0}, Nsum, x3fuel;
double ll[10]={0}, ad[2], ad3[2], C, H, O, N;
double LL, LS, L2=0, rxe, xe=0, Lstoic=0, rL2, rLstoic=0, LL1=0, LLd, rL, rLf=0, r, dLf, Lfa=50,
       dLL, rS, rLfa, an=.66, angle=0;
double dur, te=0, tstoic=0, tlf=50, time0, tid=10, t, dtCA;
double A2=1, r0, r1, l0, l1=0, steps, t1, t2, ts;
double P, cfuel, cO2, sl1, sl2, Tpm, timepm=0, dfuel, cO2a, Tdel=0;
double T, Tinj=411, Tsm, Tad=0, T2, T2o, Ttot, trf, T5, T2r, T3p, phi23, AF23, h2r, hpm;
```

```
double Tx2, T1m, T2m, T3m, Tme=0, T2in, Ta1, Ta2, Ta3;
double Vi, Vf=0, vf, v, vS=0, q0, q1, dq;
double V5=0, v1, v2, v12, v2a=0, ve=0, v3;
double mp, ms5, me=0, m1b, ma1b, mp1b, m1, m2=0, m3, mas, m2min=-100;
double ma, ma1a, m, m5in=0, mS, mo, ms3[21]={0}, ms5a, m3a=0;
double mp1, ma1, m2a=0, m2p=0, m2ain, mdotf;
double mfuel, mfS=0, mfuelo, mfdt, mfp1=0, mf1, mfp, mf2, mf3, mf2u=0, mf1u, mf2p=0, m1fv;
double h4=0, dh4, dh5, dh=0, h[21], hf4, hf5, hr=0, h5[21]={0}, deltadh=0, dho=0, h5a;
double h1a, h1p, h2a, h2p, h3p, h1[21]={0}, hr1, hfuel;
double Cp[21], Cpas, Cpp=0, E1, E2, E3, E5, Ecyl2, Ecyl, EA, E;
double x4[21], x5[21]={0}, x3[21], x3a[21], x3b[21]={0}, MW4, MW2=0, MWa=0, MW3,
       MWP,MWR;
double MW[]={1.00794, 15.9994, 14.00674, 2.0159, 17.0079, 28.0104, 30.006, 31.999, 18.016,
        44.011, 28.013, 39.948, 15.035, 16.043, 26.038, 28.054, 16.023, 17.030, 27.026, 29.019,
        25.030};
double hf[]={217977, 249197, 472629, 0, 38985, -110541, 90297, 0, -241845, -393546, 0, 0,
       145687, -74873, 226748, 52283, 167653, -45898, 135143, 12134, 476976};
double HP, HR, U1, U2, Qn=0, Qo=0, Qstep, Ea, Ea1, Ea2, mstoic=0, mfstoic=0;
double hT, hTs, Qs, mfn=0, mfo=0, mfst=0,  Qid=0, tq=0, tqt=0, m1c=0, ma1p=0, tlfq=0, mfq=0,
       maq=0, rlf=0;
double MWPs, HRs=0, HPs=0, Vsq=0, maqs=0, mfqs=0, xsum, Cp5, Cp3, Cpt, Cpf=0;
double xfuelpma, xfuelpm, xafbpm, xairpm;
double sl, Ach, hc, w, C1=2.28*Sp, C2=.00324, Tr, Pr, Vr, Tm, Cpa, k, vr, ww, Pm;
int i=0, c, z=0, y=0, sto=0, yy=0, flag1=0, flag2=0, ii=0, flag6=0, flag7=0, flag8=0, flag9=0,
       flag10=0, flag11=0;

/* function prototypes
  adiabatic function calculates adiabatic flame temperature using the equilibrium code
  enthalpy function retrieves enthalpy values for all species
  airfuel function calculates molar stoichiometric air fuel ratio
  liquidl function calculates the spray model */

void adiabatic(double ad[], double P, double hr, double x[], double fuelcomp[], double phi,
        int flag1, int flag2, double no1[]);
void enthalpy(double T, double h[], double Cp[]);
double airfuel(double fuelcomp[]);
void liquidl(double ll[], double fuel[], double fuelcomp[], double Tinj, double T, double rhoa,
       double AFs, double t, double Ufuel, int inj, double MWa, double Cpa, double te,
       double dh, double angle, double l1);


FILE *check;      // check is an output file used to check variables not in the regular output file
check=fopen("check.txt","w");
C=fuelcomp[0];   // composition of the fuel
H=fuelcomp[1];
O=fuelcomp[2];
N=fuelcomp[3];
AFs=airfuel(fuelcomp);    // stoichiometric air fuel ratio [mol air/mol fuel]
T=in1[0];          // inputs from main
P=in1[1];
Vi=in1[2];
S=in1[4];
Ecyl=in1[6];
Tr=T;              // reference temperature for heat transfer model
Tm=T;              // first motored temperature
Pr=P;              // reference pressure
```

```
Pm=P;               // first motored pressure
Vr=Vi;              // reference volume
U1=Ecyl;            // energy in the cylinder
dtCA=.25/(.006*RPM);                // time for 1/4 CA [ms]
x5[7]=.209126;              // zone 5 composition [mole fractions]
x5[10]=.781589;
x5[11]=.009285;
for(i=0;i<21;i++)           // zone 5 molecular weight [kg/kmol]
{
        MWa=MWa+x5[i]*MW[i];
}
mfdt=mdotCA*(.006*RPM)/holes; // fuel flow rate per nozzle [kmol/ms]
dur=(mf/holes)/(mfdt);      // injection duration [ms]
durCA=floor(4*mf/mdotCA)/4;     // injection duration in CA
EA=618840/(fuel[4]+25);             // activation energy [J/mol]
ms5a=mt*MWa;                // total air in cylinder [kg]
ms5=ms5a;           // initial air mass in zone 5 [kg]
rhoa=ms5/Vi;                // initial density (constant for spray calculations) [kg/m^3]
Tsm=T;              // initial Temperature (constant for spray calculations) [K]
h5a=0;
enthalpy(Tsm,h,Cp);
for(i=0;i<21;i++)
{
        h5a=h5a+x5[i]*h[i];         // initial enthalpy in zone 5
}
Ecyl=ms5*h5a/MWa-P*Vi;          // internal energy in cylinder
U1=Ecyl;
enthalpy((T+fuel[1])/2,h,Cp);

for(Cpas=0,i=0;i<21;i++)
{
        Cpas=Cpas+x5[i]*Cp[i];  // initial specific heat (constant for spray calculations)
                                    [kJ/kmol*K]
}
do      // do loop to move through 1/4 CA steps
{
        theta=(S+j/4)*PI/180;       // CA [radians]
        Vf=Vc*(1+.5*(rc-1)*(Rl+1-cos(theta)-pow(Rl*Rl-(sin(theta)*sin(theta)),.5)));
                // volume at end of step [m^3]
        if(Vc==0)
        {
                Vf=Vi;  // final volume for constant volume case
        }
        sl=aa*cos(theta)+pow((la*la-aa*aa*sin(theta)*sin(theta)),.5);           // area calculation
                                                                                    variable
        Ach=Ap+PI*B*B/4+PI*B*(la+aa-sl);        // Chamber surface area
        if(Vc==0)
        {
                Ach=Ap;         // chamber surface area for constant volume case
        }
        enthalpy(Tm,h,Cp);
        Cpa=(Cp[7]+3.7274*Cp[10]+.0444*Cp[11])/(1+3.7274+.0444);        // specific heat of
                                                                            air for motored calculation
        k=Cpa/(Cpa-Ru);         // specific heat ratio for motored calculation
        vr=Vi/Vf;           // volume ratio for motored calculation
        Tm=Tm*pow(vr,(k-1));    // new motored Temp and Press
```

```
Pm=Pm*pow(vr,k);
t=((j-1)/4)/(.006*RPM);    // time after SOI [ms]
if(t<=dur)
{
Pinj1=Pinj0+((Pinj-Pinj0)/durCA)*((j-1)/4);          // fuel injection pressure for this
                                                     CA step
if(Pinj1>Pinj)
{
        Pinj1=Pinj;        // keeps injection pressure at max value
}
ww=2*((Pinj1-in1[1])*1000/fuel[3]);          // fuel flow rate calculations
Ufuel=(injdata[1]/injdata[2])*sqrt(ww);      // fuel injection velocity
mdotf=injdata[2]*Ainj*fuel[3]*Ufuel/1000;            // fuel flow rate [kg/ms]
mdotCA=holes*mdotf/(.006*RPM*fuel[7]);               // fuel flow rate [kmol/CA]
mfdt=mdotCA*(.006*RPM)/holes;                // fuel flow rate [kmol/ms]
}
if(t>dur)
{
        te=t-dur;          // time after EOI [ms]
}
mfS=mfdt*t*fuel[7];        // fuel mass in spray before EOI [kg]
if(t>dur)
{
        mfS=mfdt*dur*fuel[7];     // fuel mass in spray after EOI [kg]
}
liquidl(ll,fuel,fuelcomp,Tinj,Tsm,rhoa,AFs,t,Ufuel,inj,MWa,Cpas,te,0,angle,l1);
        // calculate spray model at new CA

LS=ll[2];          // spray length [mm]
angle=ll[5];       // spread angle [deg]
xe=ll[8];          // length of back end of spray EOI [mm]
if(Lfa==50)
{
        LL=ll[0];          // liquid length before lift-off length moves back [mm]
}
rS=LS*an*angle;            // spray radius at tip
rxe=xe*an*angle;           // spray radius at back end after EOI
vS=(PI/3)*(rS*rS*LS)/1000000000;             // spray volume to tip
ve=(PI/3)*(rxe*rxe*xe)/1000000000;           // spray volume to back end after EOI
mS=rhoa*vS;      // spray mass to tip
me=ve*rhoa;      // spray mass to back end after EOI
MW4=1;           // MW4 non zero for upcoming loop
if(LS<LL)
{
        m1c=mS-m1c;    // new air in spray for this step
}
if(ll[3]>=AFs && sto==0)          // spray characteristics when phi at tip=1 (flame
                                  length)
{
        Lstoic=LS;         // length
        rLstoic=Lstoic*an*angle;             // radius
        tstoic=t;          // time to reach
        mstoic=mS;         // air mass
        mfstoic=mfS;       // fuel mass
        sto=1;             // change flag so values are not recalculated on next step
}
```

```
if(t>=tid && z==0)
{
        m2min=m2*.25;           // minimum mass of zone 2
        z=1;    // change flag so values are not recalculated on next step
}
if(m2min/m2>=.70 && y==0)       // set up to move lift-off length back from L2
{
        tlf=t;  // time when zone 2 is small enough to move lift-off length
        dLf=(L2-Lf)/3;  // lift-off length moves in 3 steps
        Lfa=L2;         // lift-off starts from L2
        y=1;    // change flag so values are not recalculated on next step
}
if(Lfa>=Lf && y==1)
{
        Lfa=Lfa-dLf;    // step back lift-off
}
if(Lfa<Lf)
{
        Lfa=Lf;         // minimum lift-off
}
if(xe>=Lf)
        {
                Lf=xe;          // when back of spray moves out after EOI lift-off
                                moves with it
                mfq=mfS;
                Lfa=Lf;
        }
rLfa=Lfa*an*angle;      // radius at lift-off
Tme=me*T;       // back end of spray mass*Temperature

// adiabatic flame temperature calculation for phi=1 sheath
if(t>=tid)
{               // starts after ignition delay
        enthalpy(T5,h,Cp);
        hr=0;
        for(i=0;i<21;i++)
        {
                hr=hr+x5[i]*h[i];
        }
        mas=(C+.25*H)/x5[7];
        hr=(mas*hr+fuel[5])/(mas+1);    // enthalpy of reactants
        if(Tad==0)      // flag lets adiabatic set temp guesses the first time through
        {
                flag2=0;
        }else
        {
                flag2=1;        // after first loop temperature guesses are based on
                                last loop temperature to speed convergence
                ad[0]=Tad;
        }
        flag1=0;        // tells equilibrium code to make mole fraction guess
        adiabatic(ad,P,hr,x4,fuelcomp,1,flag1,flag2,no1);       // adiabatic flame function
        Tad=ad[0];      // flame temperature
        h4=ad[1];       // product enthalpy
        MW4=0;
        for(i=0;i<21;i++)
```

```
        {
                MW4=MW4+x4[i]*MW[i];          // products molecular weight
        }
        enthalpy(Tad,h,Cp);
        Cpp=0;
        for(i=0;i<21;i++)
        {
                Cpp=Cpp+x4[i]*Cp[i];     // products specific heat
        }
        Cpp=Cpp/MW4;
}
// moving liquid length with lift-off length
if(Lfa<LL && yy<1)      // calculates minimum liquid length when lift-off length is at
                        its minimum using energy balance
{
        rLf=Lf*an*angle;            // new radius at lift-off for loop
        vf=(PI/3)*(rLf*rLf*Lf)/1000000000;          // volume to lift-off
        ma1a=rhoa*vf;    // air mass to lift-off
        enthalpy(fuel[1],h,Cp);     // enthalpies at boiling temperature of the fuel
        hf4=0;
        hf5=0;
        for(c=0;c<21;c++)
        {
                hf4=hf4+x4[c]*h[c];         // enthalpy of zone 4 species at Tboil
                hf5=hf5+x5[c]*h[c];         // enthalpy of zone 5 species at Tboil
        }
        h5a=0;
        enthalpy(Tsm,h,Cp); // enthalpies at spray calculation temperature (T at SOI)
        for(i=0;i<21;i++)
        {
                h5a=h5a+x5[i]*h[i];         // enthalpy of zone 5 species at spray temp
        }
        dh4=(ad[1]-hf4)/MW4;    // enthalpy change of zone 4 from flame temp to
                                boiling temp
        dh5=(h5a-hf5)/MWa;      // enthalpy change of zone 5 from spray calculation
                                temp to boiling temp
        dho=0;
        LL1=LL;
        do      // loop calculates minimum liquid length
        {
                rL=LL1*an*angle;            // spray radius at liquid length
                v1=(PI/3)*(rL*rL*LL1)/1000000000; // spray volume to liquid length
                m1=rhoa*v1;      // mass to liquid length
                m1b=m1-ma1a;   // mass between lift-off and liquid length
                ma1b=rhoa*PI*rLf*rLf*(LL1-Lf)/1000000000;       // mass of air
                                                        between lift-off and liquid length
                mp1b=m1b-ma1b;          // mass of products between lift-off and
                                        liquid length
                dh=((ma1a+ma1b)*dh5+mp1b*dh4)/(m1);    // total enthalpy change
                deltadh=dh-dho;  // change in dh (used to find solution)
                if(fabs(deltadh)>10 && dho>0)
                {
                        dh=(dh+dho)/2;  // if change in dh is to big use avg of old and
                                        new values (helps in finding solution)
```

```
                    }
                    liquidl(ll,fuel,fuelcomp,Tinj,T,rhoa,AFs,t,Ufuel,inj,MWa,0,te,dh,angle,
                              l1);      // use spray model to calculate new liquid length
                    dLL=(LL1-ll[0]);            // change in liquid length in loop
                    LL1=ll[0];          // update variables
                    dho=dh;
          }while(fabs(dLL)>.5);     // when change in liquid length is small stop loop
          yy++;               // insures minimum liquid length only calculated once
          LLd=(LL-LL1)/2;           // size of step used to move liquid length back
}
if(yy>0)            // move liquid length back with lift-off
{
          if(yy>1)
          {
                    LL=LL-LLd;      // step liquid length back
          }
          if(LL<LL1)
          {
                    LL=LL1;           // minimum liquid length
          }
          yy++;
}
rL=LL*an*angle;             // radius at liquid length

// zone 1 calculations
if(LS>LL)           // volume of zone 1
{
          v1=(PI/3)*(rL*rL*LL)/1000000000;   // zone 1 vol when liquid length set up
}
else
{
          v1=vS;               // zone 1 vol before reaching liquid length
}
liquidl(ll,fuel,fuelcomp,Tinj,Tsm,rhoa,AFs,t,Ufuel,inj,MWa,Cpas,te,0,angle,LL);
          // spray model to get time to reach liquid length
mf1=mfdt*ll[6]*fuel[7];   // fuel mass in zone 1 using above time
if(t>dur)
{
          mf1=mfdt*(ll[6]-te)*fuel[7];          // fuel mass in zone 1 after EOI
}
if(mf1>mfS)
{
          mf1=mfS;          // fuel mass in zone 1 before reaching liquid length
}
v1=v1-ve;           // volume of zone 1
m1=rhoa*v1;         // air mass in zone 1
if(v1<=0)           // if statements to calculate zone 1 masses for stages of spray
                    development
{
          m1=0;               // zone 1 total mass after zone disappears
          v1=0;
}
if(Lfa>LL)
{
          ma1=m1;           // zone 1 masses before lift-off length moves back
          mp1=0;
```

```
        mfp1=0;
}
if(Lfa<LL)
{
        mp1=m1+me-rhoa*(PI/3)*(rLfa*rLfa*Lfa)/1000000000-
                rhoa*PI*rLfa*rLfa*(LL-Lfa)/1000000000;  // zone 1 masses after lift-
                                                           off length moves back
        ma1=m1-mp1;
        ma1p=mp1;
        mfp1=mp1/(AFs*MWa/fuel[7]);
        mp1=mp1+mfp1;
}
mf1u=mf1-mfp1;
if(m1<=0)
{
        mp1=0;          // zone 1 masses after zone disappears
        ma1=0;
        ma1p=0;
        mfp1=0;
        mf1=0;
        mf1u=0;
}
T1m=fuel[1]*(m1+mf1);   // mass*Temperature for zone 1

// zone 2 calculations
if(t<tid)
{
        L2=LS;          // before ignition zone 2 length is spray length
}
if(t<=tid)          // volumes and masses of zone 2 for stages of spray development
{
        v2=(PI/3)*(rS*rS*L2)/1000000000-v1;     // zone 2 volume and mass before
                                                   ignition
        if(v2<0)
        {
                v2=0;
        }
        m2=rhoa*v2;
}
if(t>tid && t<=tlf)
{
        m2=m2-(m2-m2min)*A2*exp((-EA/(Ru*fuel[1])));  // zone 2 volume mass and
                                                         length during exponential burn back
        v2=m2/rhoa;
        v12=v1+v2+ve;
        L2=pow(3*v12/(PI*an*angle*an*angle),.333333)*1000;
}
if(t>tlf && t<dur)
{
        rL2=L2*an*angle;        // zone 2 volume and mass after burn back and before
                                   EOI (L2 now constant)
        v2=(PI/3)*(rL2*rL2*L2)/1000000000-(PI/3)*(rL*rL*LL)/1000000000;
        m2=rhoa*v2;
}
if(t>dur)
{
```

```
        rL2=L2*an*angle;        // zone 2 mass and volume after EOI (L2 moves with
                                back end of spray)
        v2a=(PI/3)*(rL2*rL2*L2)/1000000000;
        v2=v2a-v1-ve;
        m2=rhoa*v2;
}
if(m2<0)
{
        m2=0;   // zone 2 mass and volume when zone disappears
        v2=0;
}
t1=0;   // set up start and end of loop for stepping through zone 2
t2=0;
if(t<tid && LS>LL)
{
        t1=LL;          // start and end after zone 2 forms but before ignition
        t2=LS;
}
if(t>=tid)
{
        t1=LL;          // start and end after ignition
        t2=L2;
}
l1=LL;
if(xe>=LL && xe<L2)
{
        t1=xe;          // start and end after EOI before zone disappears
        l1=t1;
}
if(xe>=L2 && xe>0)
{
        t1=0;   // start and end after zone disappears (0 steps)
        t2=0;
}
steps=ceil(4*(t2-t1));      // number of steps through zone 2
if(steps==0)
{
        ts=0;           // step size after zone disappears (0 size)
}
else
{
        ts=(t2-t1)/steps;   // step size
}
T2m=0;
mo=m1+me;       // calculation uses energy balance including zone 1 so first mass is mass
                of one 1
mfuelo=mf1+mfdt*te*fuel[7];     // first fuel mass is fuel in zone 1
mf2=0;
T2in=Tsm;
ma=0;
mp=0;
mfp=0;
for(i=0;i<steps;i++)        // loop steps through zone 2
{
        l1=l1+ts;           // length to new step
        r=l1*an*angle;          // radius at new step
```

```
        v=(PI/3)*(r*r*l1)/1000000000;      // volume at new step
        m=rhoa*v;          // mass at new step
        ma=m;  // composition of zone 2 air and fuel before lift-ff length moves back
        mp=0;
        if(l1>Lfa)
        {
                ma=rhoa*((PI/3)*(rLfa*rLfa*Lfa)+PI*rLfa*rLfa*
                (l1-Lfa))/1000000000;      // mass of air in zone 2 after lift-off length
                                           moves
                mp=m-ma;          // mass of products in zone 2 after lift-off length
                                  moves
        }
        mfp=mp/(AFs*MWa/fuel[7]);        // mass of products including burned fuel
                                         mass in zone 2 after lift-off length moves
        liquidl(ll,fuel,fuelcomp,Tinj,Tsm,rhoa,AFs,t,Ufuel,inj,MWa,Cpas,te,0,angle,l1);
                mfuel=mfdt*ll[6]*fuel[7];              // fuel mass in new step
        Tx2=(mfuel*(-fuel[2]+fuel[0]*fuel[1])+ma*Cpas*T2in/MWa+
        mp*Cpp*Tad/MW4)/((mfuel-mfp)*fuel[0]+ma*Cpas/Mwa+(mp+mfp)*
        Cpp/MW4);        // Temperature at position x in zone 2 from energy balance
        T2m=T2m+Tx2*(m-mo+mfuel-mfuelo);      // sum of mass*Temperature in
                                              zone 2
        mf2=mf2+mfuel-mfuelo;  // total fuel mass in zone 2
        mfuelo=mfuel;    // update masses
        mo=m;
        if(flag10==0)
        {
                Tdel=Tx2;          // save first temperature in zone 2 for ignition delay
                                    model
                flag10=1;          // flag insures value only saved once
        }
}
if(flag11==0 && Tdel>0)              // ignition delay model
{
        tid=exp(-.0109*Tdel);      // ignition delay calculated
        tid=tid*1000+t;            // ignition delay added to time for current loop
        flag11=1;          // insures delay only calculated once
}
T2r=Tx2;          // last temperature in zone 2 saved for premixed burn model
phi23=ll[7];      // equivalence ratio at the end of zone 2 for premixed burn model
AF23=AFs/phi23;           // air fuel ratio at the end of zone 2 for premixed burn model
if(m2<=0)
{
        L2=xe;             // zone 2 length after EOI
}
rL2=L2*an*angle;          // radius at end of zone 2
m2a=0;
m2p=0;
mf2p=mfp;
if(mf2p>mf2)      // composition of zone 2
{
        mf2p=mf2;        // maximum burned fuel
}
mf2u=mf2-mf2p;            // unburned fuel
if(ma>0 && m2>0)
{
        m2a=ma-m1-me;             // mass of air in zone 2
```

```
        m2p=mp+mf2p;            // mass of products in zone 2 (air and fuel)
        if(xe>LL)       // zone 2 masses as zone disappears
        {
                m2a=ma-rhoa*((PI/3)*(rLfa*rLfa*Lfa)+PI*rLfa*rLfa*
                        (xe-Lfa))/1000000000;
                m2p=m2p-(me-rhoa*((PI/3)*(rLfa*rLfa*Lfa)+PI*rLfa*rLfa*
                        (xe-Lfa))/1000000000);
                mf2p=m2p/(AFs*MWa/fuel[7]);
        }
}

// zone 3 calculations
v3=vS-v1-v2-ve;         // volume and mass of zone 3
m3=mS-m1-m2-me;
if(v3<0)
{
        v3=0;           // volume and mass of zone 3 if zone doesn't exist
        m3=0;
}
m3a=m3;         // mass of air in zone 3
if(flag7==1)
{
        time0=ll[6];    // time to reach first step in zone 3 if stepping from zone 2 out
        if(m2<.000000010 && flag7==1)
        {
                time0=te;   // start time if zone disappearing and stepping from zone 2
                        end
        }
}
else
{
        liquidl(ll,fuel,fuelcomp,Tinj,T,rhoa,AFs,t,Ufuel,inj,MWa,0,te,dh,angle,LS);
        time0=ll[6];    // time to reach first step in zone 3 if stepping from the tip
                        back to zone 2
}
if(t<tid)  // set up to step through zone 3
{
        t1=0;   // start and end before ignition
        t2=0;
}
else
{
        t1=L2;  // start and end after ignition stepping from the tip
        t2=LS;
        l1=t2;
        if(flag7==1)
        {
                l1=t1;          // start and end after ignition stepping from zone 2
                flag9=1;
        }
}
if(xe>=L2)
{
        t1=xe;  // start and end after back end reaches zone 3 stepping from the tip
        l1=t1;
        l1=t2;
```

```
        if(flag7==1)
        {
                l1=t1;  // start and end after back end reaches zone 3 stepping from
                        zone 2
        }
}
steps=ceil(4*(t2-t1));      // number of steps
if(steps==0)
{
        ts=0;   // zero step size if zone doesn't exist
}
else
{
        ts=(t2-t1)/steps;   // step size
}
T3m=0;

flag2=0;
m3=0;
m2ain=rhoa*PI*rL2*rL2*(ts)/1000000000;  // mass of air from zone 2 in each step
if(flag7==1)
{
        ts=-ts;             // reverse step size for stepping  from zone 2 end
}
T2o=T2;
T2=T2m/(m2+mf2);        // zone 2 temperature
if(m2<=0)
{
        T2=T2o;         // save zone 2 temp after zone 2 disappears
}
enthalpy(T2,h,Cp);
enthalpy(T5,h1,Cp);
h2a=0;
h5a=0;
for(i=0;i<21;i++)
{
        h2a=h2a+x5[i]*h[i];     // enthalpy of zone 2 coming into zone 3
        h5a=h5a+x5[i]*h1[i];    // enthalpy of zone 5 coming into zone 3
}
for(ii=0;ii<21;ii++)
{
        ms3[ii]=0;      // initialize variables for loop
}
MW3=0;
h3p=0;
mf3=0;

flag1=0;            // first loop has equilibrium code make first mole fraction guess
for(i=0;i<steps;i++)        // step through zone 3
{
        l0=l1;              // length at start of step
        r0=l0*an*angle;             // radius at start
        l1=l1-ts;           // length at end
        r1=l1*an*angle;             // radius at end
        v=(PI/3)*(r0*r0*l0-r1*r1*l1)/1000000000;   // step volume moving from tip
        if(flag7==1 && flag9==1)
```

```
        {
                v=(PI/3)*(r1*r1*l1-r0*r0*l0)/1000000000;  // step volume moving
                                                            from zone 2
        }
        m=rhoa*v;       // air mass in step
        m5in=m-m2ain;           // mass from zone 5 in step
        hr1=(m2ain/MWa*h2a+m5in/MWa*h5a)/(m2ain/MWa+m5in/MWa);
                // enthalpy of reactants in step
        liquidl(ll,fuel,fuelcomp,Tinj,Tsm,rhoa,AFs,t,Ufuel,inj,MWa,Cpas,te,0,angle,l1);
        mfuel=fuel[7]*mfdt*(time0-ll[6]);  // fuel mass in step moving from tip
        if(ll[6]>time0)
        {
                mfuel=fuel[7]*mfdt*(ll[6]-time0);  // fuel mass in step moving from
                                                    zone 2
        }
        if(ll[7]<1.3)
        {
                flag7=1;   // change direction of steps when phi at tip reaches 1.3

        }
        if(flag1==0)
        {
                ad3[0]=1900;    // flag sets initial temperature guess for loop
                                calculation
                flag2=1;
        }
        adiabatic(ad3,P,hr1,x3a,fuelcomp,ll[7],flag1,flag2,no1);        // determine
                                        temperature at position x using adiabatic flame
        flag1=1;        // after first loop uses result from last loop as temperature and
                        mole fraction guess for next loop
        flag2=1;
        T3m=T3m+ad3[0]*(m+mfuel);       // sum mass*temperature for zone 3
        m3=m3+m;        // total air mass in zone 3
        mf3=mf3+mfuel;          // total fuel mass in zone 3
        time0=ll[6];
        for(ii=0;ii<21;ii++)
        {
                ms3[ii]=ms3[ii]+x3a[ii]*(m+mfuel);      // total mass of individual
                                                        products in zone 3
                x3a[ii]=0;
        }

}
T3p=ad3[0];     // temperature and enthalpy of step next to zone 2 for premixed burn
                model
h3p=ad3[1];
ms5=ms5a-(m1+m2+m3)*holes;    // mass in zone 5
MW3=0;
for(i=0;i<21;i++)
{
        x3[i]=ms3[i]/(m3+mf3);  // mole fractions of products in zone 3
        MW3=MW3+x3[i]*MW[i];        // molecular weight of products in zone 3
}
x3fuel=0;

// premixed burn duration model
```

```
if(m3>0)
{
        if(flag6==0)
        {
                Tpm=T2r;        // temperature at the start of the model (temperature
                                at the end of zone 2)

                cfuel=1-(1/(1+1/AF23));   // fuel mole fraction
                xfuelpma=cfuel;  // save initial fuel mole fraction
                cO2=x5[7]*(1-cfuel);      // oxygen mole fraction
                h2r=0;
                enthalpy(T2r,h,Cp);
                for(i=0;i<21;i++)
                {
                        h2r=h2r+x5[i]*(1-cfuel)*h[i];       // enthalpy of air for last
                                                            zone 2 temp
                }
                trf=T2r/1000;      // reduced temperature for fuel enthalpy curve fit
                hfuel=4184*(-9.1063*trf+246.97*trf*trf/2-143.74*trf*trf*trf/3+
                        32.329*trf*trf*trf*trf/4-.0518/trf-50.128);
                        // fuel enthalpy curve fit (for diesel fuel)
                h2r=h2r+cfuel*hfuel;      // enthalpy from zone 2 including fuel
                cfuel=cfuel*P/(Ru*Tpm)/1000;      // fuel concentration
                cO2=cO2*P/(Ru*Tpm)/1000;      // oxygen concentration
                cO2a=cO2;       // save initial oxygen concentration

                // slopes below allow calculation of oxygen concentration temperature
                and enthalpy assuming the changes are linear between the end of zone
                2 and the beginning of zone 3
                sl1=cO2/cfuel;    // change of oxygen concentration with changing fuel
                                  concentration
                sl2=(T3p-T2r)/cO2;        // change of temperature with changing
                                          oxygen concentration
                timepm=0;
                do
                {
                        dfuel=-380000000000*exp((-15098)/Tpm)*pow(cfuel,.25)
                                *pow(cO2,1.5)*.25/(.006*RPM)/1000;
                                // change in fuel concentration
                        cfuel=cfuel+dfuel;          // new fuel concentration
                        if(cfuel<=0)
                        {
                                cfuel=0;  // insures concentration can't go negative
                        }
                        cO2=sl1*cfuel;   // oxygen concentration using above slope
                        Tpm=T2r+sl2*(cO2a-cO2);  // temperature using above slope
                        xfuelpm=cfuel*Ru*T2r*1000/P;    // mole fraction of fuel
                        xairpm=xfuelpm*AF23;  // mole fraction of unburned air
                        xafbpm=1-(xairpm+xfuelpm);      // mole fraction of
                                                        products
                        hpm=h2r*(xfuelpm+xairpm)/MWa+h3p*xafbpm/MW3;
                                // energy in premixed burn
                        hpm=holes*((m2+mf2)+(m3+mf3))*hpm;

                        if(hpm-holes*((m2+mf2)+(m3+mf3))*h2r/Mwa
                                >=holes*(mf1)*fuel[2])
```

```
                {
                        timepm=timepm+.25/(.006*RPM); // if the change in
                        energy for the premixed burn is greater than the
                        energy required to vaporize the fuel then add time to
                        the premixed burn duration
                }
        }while(cfuel>0.000001);          // when the fuel concentration is
                                         small end loop
        flag6=1;    // insures premixed burn duration only calculated once
}
// Heat Release model
liquidl(ll,fuel,fuelcomp,Tinj,Tsm,rhoa,AFs,t,Ufuel,inj,MWa,Cpas,te,0,angle,Lf);
tlfq=ll[6];          // time to reach lift-off length for the heat release model
mfq=mfS-(tlfq-te)*mfdt*fuel[7];    // fuel burned in heat release
rlf=Lf*an*angle;
maq=m1+m2+m3+me-rhoa*(PI/3)*(rlf*rlf*Lf)/1000000000;  // air in heat
                                                      release
if(t>tstoic && tstoic!=0)   // equivalence ratio and products for spray beyond
                            phi=1
{
        maqs=m1+m2+m3+me-mstoic;      // mass of air beyond flame length
        mfqs=mfdt*(t-tstoic)*fuel[7];      // mass of fuel beyond flame length
        phi=(AFs*MWa/fuel[7])/(maqs/mfqs);      // equivalence ratio
                                                beyond flame length
        phia=phi;          // save phi
        rst[0]=(C+0.25*H-0.5*O)/phi;      //r0      // simplifying constants
                                                    for stoichiometry
        rst[1]=0.5*O+rst[0];              //r
        rst[2]=0.5*N+3.7274*rst[0];       //r'
        rst[3]=0.0444*rst[0];             //r''
        Nsum=H/4+rst[2]+rst[3]+rst[0]+O/2;
        // total number of moles
        x3b[7]=(rst[0]+O/2-C-H/4)/Nsum;  // O2     // products beyond flame
                                          length assuming ideal combustion
        x3b[8]=(H/2)/Nsum;                // H2O
        x3b[9]=C/Nsum;                    // CO2
        x3b[10]=rst[2]/Nsum;              // N2
        x3b[11]=rst[3]/Nsum;              // Ar
        x3fuel=0;
        enthalpy(298,h,Cp);

        MWPs=0;
        HRs=0;
        HPs=0;
        xsum=0;
        for(i=0;i<21;i++) // energy beyond phi=1 at 298 K
        {
                HRs=HRs+x5[i]*h[i];      // enthalpy of reactants
                HPs=HPs+x3b[i]*h[i];     // enthalpy of products
                MWPs=MWPs+x3b[i]*MW[i];      // molecular weight of
                                                products
                xsum=xsum+x3b[i];        // add up mole fractions
                x3b[i]=0;
        }
}
```

```
        if(t>tstoic && tstoic!=0)   // equivalence ratio and products for spray before
                                    phi=1
        {
                maq=maq-maqs;               // air before flame length
                mfq=mfq-mfqs;               // fuel before flame length
        }
        phi=(AFs*MWa/fuel[7])/((maq)/(mfq));    // equivalence ratio before flame
                                                    length
        rst[0]=(C+0.25*H-0.5*O)/phi;    // simplifying constants for stoichiometry
        rst[1]=1-1/phi;
        rst[2]=C-rst[1]*C;
        rst[3]=H/2 -rst[1]*H/2;
        rst[4]=N/2 -rst[1]*N/2+rst[0]*3.7274;
        rst[5]=rst[0]*.0444;
        Nsum=rst[1]+rst[2]+rst[3]+rst[4]+rst[5];    // total number of moles
        x3b[8]=rst[3]/Nsum;                 // H2O  // products before flame length
                                                    assuming ideal combustion
        x3b[9]=rst[2]/Nsum;                 // CO2
        x3b[10]=rst[4]/Nsum;                // N2
        x3b[11]=rst[5]/Nsum;                // Ar
        x3fuel=rst[1]/Nsum;         // unburned fuel
        enthalpy(298,h,Cp);
        MWP=0;
        HR=0;
        HP=0;
        xsum=0;
        for(i=0;i<21;i++)           // energy before phi=1 at 298 K
        {
                HR=HR+x5[i]*h[i];       // enthalpy of reactants
                HP=HP+x3b[i]*h[i];      // enthalpy of products
                MWP=MWP+x3b[i]*MW[i];       // molecular weight of products
                xsum=xsum+x3b[i];       // add up mole fractions
        }
        xsum=xsum+x3fuel;       // add mole fractions including fuel
        hfuel=4184*(-9.1063*.298+246.97*.298*.298/2 -143.74*.298*.298*.298/3+
                32.329*.298*.298*.298*.298/4 -.0518/.298-50.128);
                // enthalpy of fuel at 298 K
        HP=HP+x3fuel*fuel[5];   // enthalpy of products including unburned fuel
        MWP=MWP+x3fuel*fuel[7];     // molecular weight of products including
                                        unburned fuel
        MWR=((maq+maqs)*MWa+(mfq+mfqs)*fuel[7])/(maq+mfq+maqs+mfqs);
                // molecular weight of the reactants unto flame length and beyond
                flame length
        HR=((maq+maqs)*(HR/MWa)+(mfq+mfqs)*(hfuel)/fuel[7])/
                (maq+mfq+maqs+mfqs);            // enthalpy of the reactants up to
                                                    flame length and beyond flame length

        Qn=-((HP/MWP)*(maq+mfq)/(maq+mfq+maqs+mfqs)+
                (HPs/MWPs)*(maqs+mfqs)/(maq+mfq+maqs+mfqs)-
                HR)*((maq+mfq+maqs+mfqs));  // heat released by fuel burning
}
trf=T/1000;
hfuel=4184*(-9.1063*trf+246.97*trf*trf/2 -143.74*trf*trf*trf/3+32.329*trf*trf*trf*trf/4-
        .0518/trf-50.128)-fuel[5];  // fuel enthalpy at cylinder temperature
mfn=mf1+mf2+mf3;        // total fuel mass in this CA step
mfst=mfn-mfo;           // added fuel mass in CA step
```

```
mfo=mfn;          // save old fuel mass for next loop
enthalpy(Tsm,h,Cp);
enthalpy(fuel[1],h1,Cp);
for(hT=0,hTs=0,i=0;i<21;i++)
{
        hT=hT+x5[i]*h[i];          // enthalpy of air at spray model temperature
        hTs=hTs+x5[i]*h1[i];       // enthalpy of air at fuel boiling temperature
}
if(t<=tid)
{
        Qs=m1c*(hT-hTs)/MWa;              // energy to vaporize fuel
}
if(m3>0)
{
        Qs=0;               // after ignition turn off vaporization
}
if(t>=tid && flag8==0)    // slow premixed burn spike
{
        Qid=Qn-Qo;       // heat released in first step (at ignition delay)
        tq=.25/(.006*RPM);         // time for a CA
        tqt=(timepm*.006*RPM*tq*4);      // total time for premixed burn
        flag8=1;            // insures only calculate this once
}
if(t>=tid && tq<=tqt)
{
        Qn=Qn-Qid+Qid*(1-(.5*cos(PI*tq/tqt)+.5));            // function spreads heat
                release from first step over time given by premixed burn model using a
                cosine function
        tq=tq+.25/(.006*RPM);    // adds tq to know when premixed burn is over
}
w=C1+C2*Vd*Tr/(Pr*Vr)*(P-Pm);             // heat transfer model variable

hc=(3.26*pow(B,-.2)*pow(P,.8)*pow(T,-.55)*pow(w,.8)/1000)*Ach*(T-600)*
        (.25/(.006*RPM))/1000;    // heat transfer
//hc=0;  // zeros heat transfer correlation - can be uncommented to neglect heat transfer
Qstep=Qn-Qo-Qs-.25*hc/holes;      // heat release for 1/4 CA step
if(m3<=0)
{
        Qstep=-Qs;        // heat release is vaporization only before ignition
        MWP=1;
}
Qo=Qn;            // update variables
Qn=0;
U2=U1-P*(Vf-Vi)+Qstep*holes+mfst*hfuel*holes/fuel[7];     // internal energy

// secant method using enthalpy to get avg Temperature
Ta1=T;            // temperature guesses for secant method
Ta2=T+50;
trf=Ta1/1000;
enthalpy(Ta1,h,Cp);
for(E=0,Ea=0,i=0;i<21;i++)
{
        E=E+x5[i]*(h[i]-hf[i]);
                // air enthalpy at first temperature guess
        Ea=Ea+x3b[i]*(h[i]-hf[i]);
                // products enthalpy at first temperature guess
```

```
}
hfuel=4184*(-9.1063*trf+246.97*trf*trf/2-143.74*trf*trf*trf/3+32.329*trf*trf*trf*trf/4-
        .0518/trf-50.128)-fuel[5]; // fuel enthalpy at first temperature guess
Ea=Ea+x3fuel*hfuel;
E=E*(ms5a-(maq+maqs)*holes)/MWa+Ea*holes*(maq+mfq+maqs+mfqs)/MWP+
        hfuel*holes*(mfS-mfq-mfqs)/fuel[7]-P*(Vf);          // total enthalpy at first
                                                            temperature guess
q0=U2-E;        // secant method function at first temperature guess
trf=Ta2/1000;
enthalpy(Ta2,h,Cp);
for(E1=0,Ea1=0,i=0;i<21;i++)
{
        E1=E1+x5[i]*(h[i]-hf[i]);           // air enthalpy at second temperature guess
        Ea1=Ea1+x3b[i]*(h[i]-hf[i]);        // products enthalpy at second temperature
                                            guess
}
hfuel=4184*(-9.1063*trf+246.97*trf*trf/2-143.74*trf*trf*trf/3+32.329*trf*trf*trf*trf/4-
        .0518/trf-50.128)-fuel[5];  // fuel enthalpy at second temperature guess
Ea1=Ea1+x3fuel*hfuel;
E1=E1*(ms5a-(maq+maqs)*holes)/MWa+Ea1*holes*(maq+mfq+maqs+mfqs)/MWP+
        hfuel*holes*(mfS-mfq-mfqs)/fuel[7]-P*(Vf);          // total enthalpy at second
                                                            temperature guess
q1=U2-E1;       // secant method function at second temperature guess
do
{
        dq=(q1-q0);         // change in function
        Ta3=Ta2-q1*((Ta2-Ta1)/dq);          // new temperature guess
        Ta1=Ta2;            // update variables
        q0=q1;
        Ta2=Ta3;
        trf=Ta3/1000;
        enthalpy(Ta3,h,Cp);
        for(E2=0,Ea2=0,i=0;i<21;i++)
        {
                E2=E2+x5[i]*(h[i]-hf[i]);  // air enthalpy at subsequent temperature
                                           guess
                Ea2=Ea2+x3b[i]*(h[i]-hf[i]);        // products enthalpy at subsequent
                                                    temperature guess
        }
        hfuel=4184*(-9.1063*trf+246.97*trf*trf/2-143.74*trf*trf*trf/3+
                32.329*trf*trf*trf*trf/4-.0518/trf-50.128)-fuel[5];     // fuel enthalpy at
                                                                        subsequent temperature guess
        Ea2=Ea2+x3fuel*hfuel;
        E2=E2*(ms5a-(maq+maqs)*holes)/MWa+
                Ea2*holes*(maq+mfq+maqs+mfqs)/MWP+
                hfuel*holes*(mfS-mfq-mfqs)/fuel[7]-P*(Vf);          // total enthalpy
                                                                    at subsequent temperature guess
        q1=U2-E2;       // secant method function at subsequent temperature guess
}while(fabs(q1)>.00001);            // end when function small
enthalpy(Ta3,h,Cp);
for(Cp5=0,Cp3=0,i=0;i<21;i++)
{
        Cp5=Cp5+x5[i]*Cp[i];    // specific heat of air
        Cp3=Cp3+x3b[i]*Cp[i];   // specific heat of products
}
trf=Ta3/1000;
```

```
Cpf=4.184*(-9.1063+246.97*trf-143.74*trf*trf+32.329*trf*trf*trf+.0518/(trf*trf));
        // specific heat of fuel
Cp3=Cp3+x3fuel*Cpf;
Cpt=(Cp5*(ms5a-(maq+maqs)*holes)+Cp3*holes*(maq+mfq+maqs+mfqs)+
        fuel[0]*holes*(mfS-mfq-mfqs))/(ms5a+mfS);          // avg specific heat
T=Ta3;          // new cylinder Temperature
P=((ms5a-(maq+maqs)*holes)/MWa+holes*(maq+mfq+maqs+mfqs)/MWP+
        holes*(mfS-mfq-mfqs)/fuel[7])*Ru*T/(Vf); // new cylinder pressure
U1=U2;          // update energy
enthalpy(T1m/(m1+mf1),h,Cp);    // enthalpies at zone 1 temperature
h1a=0;
h1p=0;
for(i=0;i<21;i++)
{
        h1a=h1a+x5[i]*(h[i]-hf[i]);         // air enthalpy in zone 1
        h1p=h1p+x4[i]*(h[i]-hf[i]);         // product enthalpy in zone 1
}
m1fv=(ma1*(h5a-h1a)/MWa)/(fuel[2]);     // fuel enthalpy in zone 1
E1=(ma1*h1a/MWa+(mp1+mfp1)*h1p/MW4+m1fv*fuel[2]);
// energy in zone 1
enthalpy(T2m/(m2+mf2),h,Cp);    // enthalpies at zone 2 temperature
h2a=0;
h2p=0;
for(i=0;i<21;i++)
{
        h2a=h2a+x5[i]*(h[i]-hf[i]);         // air enthalpy in zone 2
        h2p=h2p+x4[i]*(h[i]-hf[i]);         // product enthalpy in zone 2
}
E2=(m2a*h2a/MWa+m2p*h2p/MW4+mf2u*fuel[0]*(T2m/(m2+mf2)-fuel[1]));
        // energy in zone 2
enthalpy(T3m/(m3+mf3),h,Cp);    // enthalpies at zone 3 temperature
h3p=0;
for(i=0;i<21;i++)
{
        h3p=h3p+x3[i]*(h[i]-hf[i]);         // product enthalpy in zone 3
}
E3=((m3+mf3)*h3p/MW3);          // energy in zone 3
if(m1<=0)       // set energies of zones to zero if zone doesn't exist
{
        E1=0;
}
if(m2<=0)
{
        E2=0;
}
if(m3<=0)
{
        E3=0;
}
E5=U2-(E1+E2+E3)*holes+P*Vf;            // energy in zone 5

// secant method using energy in zone 5 to get Temperature of zone 5
Ta1=1000;       // temperature guesses
Ta2=T+50;
trf=Ta1/1000;
enthalpy(Ta1,h,Cp);
```

```
		for(E=0,i=0;i<21;i++)
		{
			E=E+x5[i]*(h[i]-hf[i]);	// zone 5 energy at first guess
		}

		q0=E5*MWa/(ms5)-E;	// secant method function at first guess
		enthalpy(Ta2,h,Cp);
		for(Ea1=0,i=0;i<21;i++)
		{
			Ea1=Ea1+x5[i]*(h[i]-hf[i]);		// zone 5 energy at second guess
		}
		q1=E5*MWa/(ms5)-Ea1;  // secant method function at second guess
		do
		{
			dq=(q1-q0);
			Ta3=Ta2-q1*((Ta2-Ta1)/dq);
			Ta1=Ta2;
			q0=q1;
			Ta2=Ta3;
			enthalpy(Ta3,h,Cp);
			for(Ea2=0,i=0;i<21;i++)
			{
				Ea2=Ea2+x5[i]*(h[i]-hf[i]);		// zone 5 energy at subsequent
								guess
			}
			q1=E5*MWa/(ms5)-Ea2;		// secant method function at subsequent
								guess
		}while(fabs(q1)>.00001);
		T5=Ta3;		// zone 5 Temperature
		V5=((ms5/holes)/MWa*Ru*T5)/P;		// zone 5 volume
		Vsq=Vf/holes-V5;		// actual spray volume
		Ecyl2=holes*(E1+E2+E3)+E5-P*Vf;		// cylinder energy using zone energies
		Ttot=(holes*(T1m+T2m+T3m)+T5*ms5)/(holes*(m1+mf1+m2+mf2+m3)+ms5);
			// mass averaged temperature for all zones
		Ecyl=Ecyl-P*(Vf-Vi);	// cylinder energy using PdV
		fprintf(output,"%.2lf	%lf	%lf	%lf	%lf	%lf	%.10lf	%.10lf
			%.10lf	%.10lf\n",(S+j/4),T,P,Vf*1000,t,U2,Ecyl2,LL,LS,hc);		// print
								output file
		Vi=Vf;	// update volume
		printf("CA=%.2lf\n",S+j/4);		// print crank angle to the screen to watch progress
		fprintf(check,"%.10lf	%.10lf	%.10lf	%.10lf	%.10lf	%.10lf	%.10lf	%.10lf
			%.10lf	%.10lf	%.10lf\n",S+j/4,S+tid*.006*RPM,S+dur*.006*RPM,
			S+tstoic*.006*RPM,timepm*.006*RPM,LS,E1,E2,m3,tstoic,Tad);	// print
								check file
		j++;	// add for next theta calculation
	}while(phi>=1.1 || phi==0 || Qstep>0);		// end when all fuel is burned and all heat is released
	in1[0]=T;		// send variables to main
	in1[1]=P;
	in1[2]=Vf;
	in1[3]=(S+j/4);
	in1[6]=Ecyl2;
	in1[7]=Pm;
	in1[8]=Tm;
	fclose(check);
	return;		// return to main program
}
```

```
/*-------------------------------------------------------------*/
/* product enthalpy and Cp curve fits from Turns       */
/* units                                               */
/*      h=kJ/kmol                                       */
/*      Cp=kJ/kmol*K                                    */

void enthalpy(double T, double h[], double Cp[])
{
        int i;      // h fit=h0/Ru*T     Cp fit=Cp/Ru
        if (T < 1000)       // curve fits below 1000 K
        {
                h[0]= (2.50000000)+ (2.547162e+4)/T;       // H
                h[1]=  2.94642800 - (1.63816650e-3/2)*T + (2.42103100e-6/3)*T*T -
                        (1.60284310e-9/4)*T*T*T + (3.89069600e-13/5)*T*T*T*T +
                        2.91476400e+4/T;           // O
                h[2]=  2.50307100 - (2.18001800e-5/2)*T + (5.42052900e-8/3)*T*T -
                        (5.6475600e-11/4)*T*T*T + (2.09990400e-14/5)*T*T*T*T +
                        5.60989000e+4/T;           // N
                h[3]=  3.29812400 + (8.24944100e-4/2)*T - (8.14301500e-7/3)*T*T -
                        (9.4754340e-11/4)*T*T*T + (4.13487200e-13/5)*T*T*T*T -
                        1.01252090e+3/T;           // H2
                h[4]=  3.63726600 + (1.85091000e-4/2)*T - (1.67616460e-6/3)*T*T +
                        (2.38720200e-9/4)*T*T*T - (8.43144200e-13/5)*T*T*T*T +
                        3.60678100e+3/T;           // OH
                h[5]=  3.26245100 + (1.51194090e-3/2)*T - (3.88175500e-6/3)*T*T +
                        (5.58194400e-9/4)*T*T*T - (2.47495100e-12/5)*T*T*T*T -
                        1.43105390e+4/T;           // CO
                h[6]=  3.37654100 + (1.25306340e-3/2)*T - (3.30275000e-6/3)*T*T +
                        (5.21781000e-9/4)*T*T*T - (2.44626200e-12/5)*T*T*T*T +
                        9.81796100e+3/T;           // NO
                h[7]=  3.21293600 + (1.12748640e-3/2)*T - (5.75615000e-7/3)*T*T +
                        (1.31387730e-9/4)*T*T*T - (8.76855400e-13/5)*T*T*T*T -
                        1.00524900e+3/T;           // O2
                h[8]=  3.38684200 + (3.47498200e-3/2)*T - (06.3546960e-6/3)*T*T +
                        (6.96858100e-9/4)*T*T*T - (2.50658800e-12/5)*T*T*T*T -
                        3.02081100e+4/T;           // H2O
                h[9]=  2.27572400 + (9.92207200e-3/2)*T - (1.04091130e-5/3)*T*T +
                        (6.86668600e-9/4)*T*T*T - (2.11728000e-12/5)*T*T*T*T -
                        4.83731400e+4/T;           // CO2
                h[10]= 3.29867700 + (1.40824040e-3/2)*T - (3.96322200e-6/3)*T*T +
                        (5.64151500e-9/4)*T*T*T - (2.44485400e-12/5)*T*T*T*T -
                        1.02089990e+3/T;           // N2
                h[11]=20.786*(T-298);              // Ar
                h[12]= 3.68923622 + (2.21810144e-3/2)*T + (5.23238676e-6/3)*T*T -
                        (6.63176827e-9/4)*T*T*T + (2.60202183e-12/5)*T*T*T*T +
                        1.62901205e+4/T;           // CH3
                h[13]= 4.41637060 - (8.70015194e-3/2)*T + (3.76291670e-5/3)*T*T -
                        (3.77618177e-8/4)*T*T*T + (1.30862920e-11/5)*T*T*T*T -
                        1.01996733e+4/T;           // CH4
                h[14]=-.511756038 + (3.30314770e-2/2)*T - (6.05469976e-5/3)*T*T +
                        (5.56486251e-8/4)*T*T*T - (1.94703892e-11/5)*T*T*T*T +
                        2.63899141e+4/T;           // C2H2
                h[15]= 1.32064875 + (1.15017616e-2/2)*T + (9.19263758e-6/3)*T*T -
                        (1.90607146e-8/4)*T*T*T + (8.37926993e-12/5)*T*T*T*T +
                        5.35820211e+3/T;           // C2H4
```

```
h[16]= 1.28342106 + (2.13645232e-2/2)*T - (6.21251824e-5/3)*T*T +
       (8.20045112e-8/4)*T*T*T - (3.87470151e-11/5)*T*T*T*T +
       1.92381451e+4/T;          // NH2
h[17]= 3.80198217 - (1.28349450e-3/2)*T + (1.35478900e-5/3)*T*T -
       (1.46019990e-8/4)*T*T*T + (5.34854690e-12/5)*T*T*T*T -
       6.69042791e+3/T;          // NH3
h[18]= 1.53227086 + (1.55889172e-2/2)*T - (2.82817979e-5/3)*T*T +
       (2.70534804e-8/4)*T*T*T - (9.89830250e-12/5)*T*T*T*T +
       1.53063256e+4/T;          // HCN
h[19]= 4.82297645 - (7.66026577e-3/2)*T + (2.57241316e-5/3)*T*T -
       (2.76736722e-8/4)*T*T*T + (1.04823181e-11/5)*T*T*T*T +
       1.84213318e+2/T;          // HCO
h[20]= 2.46718716 + (1.02134320e-2/2)*T - (1.50930296e-5/3)*T*T +
       (1.30319012e-8/4)*T*T*T - (4.51091001e-12/5)*T*T*T*T +
       5.62901687e+4/T;          // C2H

Cp[0]=  2.5;        // H
Cp[1]=  2.94642800 - (1.63816650e-3)*T + (2.42103100e-6)*T*T -
        (1.60284310e-9)*T*T*T + (3.89069600e-13)*T*T*T*T;       // O
Cp[2]=  2.50307100 - (2.18001800e-5)*T + (5.42052900e-8)*T*T -
        (5.6475600e-11)*T*T*T + (2.09990400e-14)*T*T*T*T;       // N
Cp[3]=  3.29812400 + (8.24944100e-4)*T - (8.14301500e-7)*T*T -
        (9.4754340e-11)*T*T*T + (4.13487200e-13)*T*T*T*T;       // H2
Cp[4]=  3.63726600 + (185091000e-4)*T - (1.67616460e-6)*T*T +
        (2.38720200e-9)*T*T*T - (8.43144200e-13)*T*T*T*T;       // OH
Cp[5]=  3.26245100 + (1.51194090e-3)*T - (3.88175500e-6)*T*T +
        (5.58194400e-9)*T*T*T - (2.47495100e-12)*T*T*T*T;       // CO
Cp[6]=  3.37654100 + (1.25306340e-3)*T - (3.30275000e-6)*T*T +
        (5.21781000e-9)*T*T*T - (2.44626200e-12)*T*T*T*T;       // NO
Cp[7]=  3.21293600 + (1.12748640e-3)*T - (5.75615000e-7)*T*T +
        (1.31387730e-9)*T*T*T - (8.76855400e-13)*T*T*T*T;       // O2
Cp[8]=  3.38684200 + (3.47498200e-3)*T - (06.3546960e-6)*T*T +
        (6.96858100e-9)*T*T*T - (2.50658800e-12)*T*T*T*T;       // H2O
Cp[9]=  2.27572400 + (9.92207200e-3)*T - (1.04091130e-5)*T*T +
        (6.86668600e-9)*T*T*T - (2.11728000e-12)*T*T*T*T;       // CO2
Cp[10]= 3.29867700 + (1.4082 4040e-3)*T - (3.96322200e-6)*T*T +
        (5.64151500e-9)*T*T*T - (2.44485400e-12)*T*T*T*T;       // N2
Cp[11]=20.786;  // Ar
Cp[12]= 3.68923622 + (2.21810144e-3)*T + (5.23238676e-6)*T*T -
        (6.63176827e-9)*T*T*T + (2.60202183e-12)*T*T*T*T;       // CH3
Cp[13]= 4.41637060 - (8.70015194e-3)*T + (3.76291670e-5)*T*T -
        (3.77618177e-8)*T*T*T + (1.30862920e-11)*T*T*T*T;       // CH4
Cp[14]=-.511756038 + (3.30314770e-2)*T - (6.05469976e-5)*T*T +
        (5.56486251e-8)*T*T*T - (1.94703892e-11)*T*T*T*T;       // C2H2
Cp[15]= 1.32064875 + (1.15017616e-2)*T + (9.19263758e-6)*T*T -
        (1.90607146e-8)*T*T*T + (8.37926993e-12)*T*T*T*T;       // C2H4
Cp[16]= 1.28342106 + (2.13645232e-2)*T - (6.21251824e-5)*T*T +
        (8.20045112e-8)*T*T*T - (3.87470151e-11)*T*T*T*T;       // NH2
Cp[17]= 3.80198217 - (1.28349450e-3)*T + (1.35478900e-5)*T*T -
        (1.46019990e-8)*T*T*T + (5.34854690e-12)*T*T*T*T;       // NH3
Cp[18]= 1.53227086 + (1.55889172e-2)*T - (2.82817979e-5)*T*T +
        (2.70534804e-8)*T*T*T - (9.89830250e-12)*T*T*T*T;       // HCN
Cp[19]= 4.82297645 - (7.66026577e-3)*T + (2.57241316e-5)*T*T -
        (2.76736722e-8)*T*T*T + (1.04823181e-11)*T*T*T*T;       // HCO
Cp[20]= 2.46718716 + (1.02134320e-2)*T - (1.50930296e-5)*T*T +
        (1.30319012e-8)*T*T*T - (4.51091001e-12)*T*T*T*T;       // C2H
```

```
}
else    // curve fits above 1000 K
{
        h[0]= (2.5)     + (2.547162e+4)/T;  // H
        h[1]=  2.54205900 - (2.75506100e-5/2)*T - (3.10280300e-9/3)*T*T +
               (4.55106700e-12/4)*T*T*T - (4.36805100e-16/5)*T*T*T*T +
               2.92308000e+4/T;          // O
        h[2]=  2.45026800 + (1.06614580e-4/2)*T - (7.46533700e-8/3)*T*T +
               (1.87965200e-11/4)*T*T*T - (1.02598390e-15/5)*T*T*T*T +
               5.61160400e+4/T;          // N
        h[3]=  2.99142300 + (7.00064400e-4/2)*T - (5.63382800e-8/3)*T*T -
               (9.23157800e-12/4)*T*T*T + (1.58275190e-15/5)*T*T*T*T -
               8.35034000e+2/T;          // H2
        h[4]=  2.88273000 + (1.01397430e-3/2)*T - (2.27687700e-7/3)*T*T +
               (2.17468300e-11/4)*T*T*T - (5.12630500e-16/5)*T*T*T*T +
               3.88688800e+3/T;          // OH
        h[5]=  3.02507800 + (1.44268850e-3/2)*T - (5.63082700e-7/3)*T*T +
               (1.01858130e-10/4)*T*T*T - (6.91095100e-15/5)*T*T*T*T -
               1.42683500e+4/T;          // CO
        h[6]=  3.24543500 + (1.26913830e-3/2)*T - (5.01589000e-7/3)*T*T +
               (9.16928300e-11/4)*T*T*T - (6.27541900e-15/5)*T*T*T*T +
               9.80084000e+3/T;          // NO
        h[7]=  3.69757800 + (6.13519700e-4/2)*T - (1.25884200e-7/3)*T*T +
               (1.77528100e-11/4)*T*T*T - (1.13643540e-15/5)*T*T*T*T -
               1.23393010e+3/T;          // O2
        h[8]=  2.67214500 + (3.05629300e-3/2)*T - (8.73026000e-7/3)*T*T +
               (1.20099640e-10/4)*T*T*T - (6.39161800e-15/5)*T*T*T*T -
               2.98992100e+4/T;          // H2O
        h[9]=  4.45362300 + (3.14016800e-3/2)*T - (1.27841050e-6/3)*T*T +
               (2.39399600e-10/4)*T*T*T - (1.66903330e-14/5)*T*T*T*T -
               4.89669600e+4/T;          // CO2
        h[10]= 2.92664000 + (1.48797680e-3/2)*T - (5.68476000e-7/3)*T*T +
               (1.00970380e-10/4)*T*T*T - (6.75335100e-15/5)*T*T*T*T -
               9.22797700e+2/T;          // N2
        h[11]=20.786*(T-298);     // Ar
        h[12]= 2.22212302 + (7.39133106e-3/2)*T - (3.11253645e-6/3)*T*T +
               (6.38016845e-10/4)*T*T*T - (5.17254389e-14/5)*T*T*T*T +
               1.66655631e+4/T;          // CH3
        h[13]= .392678561 + (1.28941183e-2/2)*T - (5.77810964e-6/3)*T*T +
               (1.24667029e-9/4 )*T*T*T - (1.05358046e-13/5)*T*T*T*T -
               9.61781722e+3/T;          // CH4
        h[14]=90.13224340 - (1.73064598e-1/2)*T + (1.29168005e-4/3)*T*T -
               (3.99831826e-8/4 )*T*T*T + (4.36915115e-12/5)*T*T*T*T -
               5.24468020e+3/T;          // C2H2
        h[15]= 2.23392408 + (1.42680731e-2/2)*T - (6.47812640e-6/3)*T*T +
               (1.41470155e-9/4 )*T*T*T - (1.20851028e-13/5)*T*T*T*T +
               4.86652922e+3/T;          // C2H4
        h[16]= 2.31926713 + (4.10124638e-3/2)*T - (1.57101097e-6/3)*T*T +
               (2.89579692e-10/4)*T*T*T - (2.09174615e-14/5)*T*T*T*T +
               1.94327270e+4/T;          // NH2
        h[17]= 2.04193887 + (6.84131719e-3/2)*T - (2.51635499e-6/3)*T*T +
               (4.54017540e-10/4)*T*T*T - (3.31789419e-14/5)*T*T*T*T -
               6.32570110e+3/T;          // NH3
        h[18]= 3.42707507 + (3.95250318e-3/2)*T - (1.64192919e-6/3)*T*T +
               (3.34262947e-10/4)*T*T*T - (2.68528684e-14/5)*T*T*T*T +
               1.50552866e+4/T;          // HCN
```

```
        h[19]= 2.87342861 + (4.43037238e-3/2)*T - (2.08366110e-6/3)*T*T +
                (4.71108668e-10/4)*T*T*T - (4.16096466e-14/5)*T*T*T*T +
                4.26296671e+2/T;            // HCO
        h[20]= 3.88308949 + (3.48532441e-3/2)*T - (1.60351781e-6/3)*T*T +
                (4.19655298e-10/4)*T*T*T - (4.34906660e-14/5)*T*T*T*T +
                5.60013939e+4/T;            // C2H

        Cp[0]=  2.5;        // H
        Cp[1]=  2.54205900 - (2.75506100e-5)*T - (3.10280300e-9)*T*T +
                (4.55106700e-12)*T*T*T - (4.36805100e-16)*T*T*T*T;      // O
        Cp[2]=  2.45026800 + (1.06614580e-4)*T - (7.46533700e-8)*T*T +
                (1.87965200e-11)*T*T*T - (1.02598390e-15)*T*T*T*T;      // N
        Cp[3]=  2.99142300 + (7.00064400e-4)*T - (5.63382800e-8)*T*T -
                (9.23157800e-12)*T*T*T + (1.58275190e-15)*T*T*T*T;      // H2
        Cp[4]=  2.88273000 + (1.01397430e-3)*T - (2.27687700e-7)*T*T +
                (2.17468300e-11)*T*T*T - (5.12630500e-16)*T*T*T*T;      // OH
        Cp[5]=  3.02507800 + (1.44268850e-3)*T - (5.63082700e-7)*T*T +
                (1.01858130e-10)*T*T*T - (6.91095100e-15)*T*T*T*T;      // CO
        Cp[6]=  3.24543500 + (1.26913830e-3)*T - (5.01589000e-7)*T*T +
                (9.16928300e-11)*T*T*T - (6.27541900e-15)*T*T*T*T;      // NO
        Cp[7]=  3.69757800 + (6.13519700e-4)*T - (1.25884200e-7)*T*T +
                (1.77528100e-11)*T*T*T - (1.13643540e-15)*T*T*T*T;      // O2
        Cp[8]=  2.67214500 + (3.05629300e-3)*T - (8.73026000e-7)*T*T +
                (1.20099640e-10)*T*T*T - (6.39161800e-15)*T*T*T*T;      // H2O
        Cp[9]=  4.45362300 + (3.14016800e-3)*T - (1.27841050e-6)*T*T +
                (2.39399600e-10)*T*T*T - (1.66903330e-14)*T*T*T*T;      // CO2
        Cp[10]= 2.92664000 + (1.48797680e-3)*T - (5.68476000e-7)*T*T +
                (1.00970380e-10)*T*T*T - (6.75335100e-15)*T*T*T*T;      // N2
        Cp[11]=20.786;  // Ar
        Cp[12]= 2.22212302 + (7.39133106e-3)*T - (3.11253645e-6)*T*T +
                (6.38016845e-10)*T*T*T - (5.17254389e-14)*T*T*T*T;      // CH3
        Cp[13]= .392678561 + (1.28941183e-2)*T - (5.77810964e-6)*T*T +
                (1.24667029e-9 )*T*T*T - (1.05358046e-13)*T*T*T*T;      // CH4
        Cp[14]=90.13224340 - (1.73064598e-1)*T + (1.29168005e-4)*T*T -
                (3.99831826e-8 )*T*T*T + (4.36915115e-12)*T*T*T*T;      // C2H2
        Cp[15]= 2.23392408 + (1.42680731e-2)*T - (6.47812640e-6)*T*T +
                (1.41470155e-9 )*T*T*T - (1.20851028e-13)*T*T*T*T;      // C2H4
        Cp[16]= 2.31926713 + (4.10124638e-3)*T - (1.57101097e-6)*T*T +
                (2.89579692e-10)*T*T*T - (2.09174615e-14)*T*T*T*T;      // NH2
        Cp[17]= 2.04193887 + (6.84131719e-3)*T - (2.51635499e-6)*T*T +
                (4.54017540e-10)*T*T*T - (3.31789419e-14)*T*T*T*T;      // NH3
        Cp[18]= 3.42707507 + (3.95250318e-3)*T - (1.64192919e-6)*T*T +
                (3.34262947e-10)*T*T*T - (2.68528684e-14)*T*T*T*T;      // HCN
        Cp[19]= 2.87342861 + (4.43037238e-3)*T - (2.08366110e-6)*T*T +
                (4.71108668e-10)*T*T*T - (4.16096466e-14)*T*T*T*T;      // HCO
        Cp[20]= 3.88308949 + (3.48532441e-3)*T - (1.60351781e-6)*T*T +
                 (4.19655298e-10)*T*T*T -(4.34906660e-14)*T*T*T*T;      // C2H
}
for(i=0;i<21;i++)               // h0=h fit*Ru*T  Cp=Cpfit*Ru
{
        if(i!=11)
        {
                h[i]=h[i]*(Ru*T);
                Cp[i]=Cp[i]*Ru;
        }
}
```

```
	return;
}
/*-------------------------------------------------------------*/
/* adiabatic flame temperature calculation                     */

void adiabatic(double ad[], double P, double hr, double x[], double fuelcomp[], double phi, int flag1, int
flag2, double no1[])

{
	// declare variables
	double T1, T, q0, q1, T2, dq;

	// function prototypes
	double equil(double T, double P, double fuelcomp[], double phi, double x[], int flag1,
		double no1[]);
	if(ad[0]>3500 || ad[0]<1000)
		flag2=0;
	if(flag2==0)
	{
		T=2500;		// default temperature guesses
		T1=2700;
	}
	if(flag2==1)
	{
		T=ad[0];		// temperature guesses if previous guess calculated in injection function
		T1=T+10;
	}
	q0=equil(T,P,fuelcomp,phi,x,flag1,no1)-hr;  // secant method function using enthalpy from
                                                    equilibrium function at first temperature guess
	q1=equil(T1,P,fuelcomp,phi,x,flag1,no1)-hr;		// secant method function using enthalpy
                                                from equilibrium function at second temperature guess
	do
	{
		dq=(q1-q0);		// change in secant method function
		T2=T1-q1*((T1-T)/dq);	// new temperature guess
		T=T1;
		q0=q1;
		T1=T2;
		q1=equil(T2,P,fuelcomp,phi,x,flag1,no1)-hr;		// secant method function using
                                enthalpy from equilibrium function at subsequent temperature guess
	}while(fabs(q1)>100);
	ad[0]=T2;		// outputs to be used in other functions
	ad[1]=hr;
	return;
}
/*****************************************************************************/
/* Program name: Equilib.cpp                                                 */
/* Programmer:  Kenth Svensson                                               */
/* Project:     Rich's diesel code                                           */
/* Date:        1 March 2002                                                 */
/* Description: This program calculates the equilibrium composition of products */
/* from diesel combustion. The equation system is solved by LU decomposition. */
/* [mat][x]=[vec]=-[F]                                                       */
/*                                                                           */
/* dynamic memory allocation, handles neg. numbers, pivoting,                */
/* globally convergent                                                       */
```

```
/*****************************************************************************/

#define TINY 1.0e-20       // convergence criteria for equilibrium model
#define MAXITS 400                 // maximum iterations
#define TOLF 1.0e-4                // convergence on function values
#define TOLMIN 1.0e-6
#define TOLX 1.0e-7                // convergence on delta x
#define STPMX 100.0
#define ALPHA 1.0e-4
#define FREERETURN {  free(xold);free(p);free(indx);free(g);\
                                                        for(i=0; i<n; i++)\
                                                                free(fjac[i]);\
                                                        free(fjac);return;}            // free allocated
                                                                memory in equilibrium model

double equil(double T, double P, double fuelcomp[], double phi, double x[], int flag1, double no1[])
{
        /* variable definitions
           i ---------- loop counter
           row --------- number of rows in the matrix
           check ------- flag used in equation solver (newt function)
           flag -------- controls direction of equivalence ratio steps
           flag3 ------- when desired equivalence ratio is reached shuts off loop
           flag4 ------- also used to shut off loop
           flag5 ------- controls diminishing the step size
           tol --------- tolerance for phi step
           sum --------- sum of mole fractions
           molsum ------ total number of moles
           end --------- ending equivalence ratio
           step -------- phi  step size
           n ----------- # C atoms in fuel
           m ----------- # H atoms in fuel
           l ----------- # O atoms in fuel
           k ----------- # N atoms in fuel
          step1  ------- phi step size used when diminishing step size
          phi1 -------- equivalence ratio used when diminishing step size
           *vec -------- solution vector
           *no --------- mole fraction guesses
           *K ----------  equilibrium constants
           *molfrac ----  mole fractions of individual species
           r[4] -------- constants to simplify stoichiometry
           h[21] ------- enthalpies of individual species [kJ/kmol]
           hx[21] ------ enthalpies of individual species [kJ/kmol*mole fraction]
           hp ---------- enthalpy of the products
           Cp[21] ------ specific heats of individual species
           phis -------- starting equivalence ratio
           species[21][5] - */

        // declare variables
        int i, row=21, check, flag=0, flag3=0, flag4=0, flag5=0;
        double tol=1e-12, sum, molsum, end, step=0.1, n, m, l, k, step1, phi1;
        double *vec, *no, *K, *molfrac, r[4], h[21], hx[21], hp, Cp[21], phis=1.5;
        char species[21][5]={"H","O","N","H2","OH","CO","NO","O2","H2O","CO2","N2","Ar","CH3",
                "CH4","C2H2","C2H4","NH2","NH3","HCN","HCO","C2H"};

        /* function prototypes
```

```
   vector function calculates the solution vector
   newt function is the newton raphson method for non linear systems
   Kp function retrieves the equilibrium constants at a specified temperature
   enthalpy function retrieves individual species enthalpies
   guess function makes an initial guess of all mole fractions */
void vector(double vec[], double no[], double r[], double K[], int row, double *molsum,
        double fuelcomp[], double P);
void newt(double x[],int n,int *check,double fvec[],double K[],double P,double *molsum,
        double r[], double fuelcomp[]);
void Kp(double K[], double T);
void enthalpy(double T, double h[], double Cp[]);
void guess(double no[], double K[], double fuelcomp[], double P);

  //Dynamic memory allocation
vec=(double*)malloc(row*sizeof(double));  // allocate memory for solution guess mol efraction
                                             and equilibrium constant vectors
no=(double*)malloc(row*sizeof(double));
molfrac=(double*)malloc(row*sizeof(double));
K=(double*)malloc((row-5)*sizeof(double));

end=phi;            // end equivalence ratio is set to input value
n=fuelcomp[0];   // assign fuel composition
m=fuelcomp[1];
l=fuelcomp[2];
k=fuelcomp[3];
P=P/101.325;                // convert pressure into atmospheres for Kp values
Kp(K,T);            // get equilibrium constants
if(no1[11]==0)
{
        flag1=0;            // this insures a non-zero initial guess will be made
}
//Initialize guess vector
if(flag1==0)
{
        guess(no,K,fuelcomp,P);   // calls initial guess function if no guess exists
}else
{
        for(i=0;i<21;i++)
        {
                no[i]=no1[i];       // updates initial guess to previous solution (used when
                                       stepping through zone 3)
        }
}
if(flag1==1)
{
        phis=phi;           // starting and ending phi are the same if good guess available (used
                            when stepping through zone 3)
}
if(end>phis)
{
        step=-0.05;         // changes step direction if end phi is greater than start phi
        flag=1;
}
flag3=0;
do   //Range of phi
{                                           //Calculate stoichiometry simplifying constants
```

```
r[0]=(n+0.25*m-0.5*l)/phis;          //r0
r[1]=0.5*l+r[0];                     //r
r[2]=0.5*k+3.7274*r[0];              //r'
r[3]=0.0444*r[0];                    //r"
for(i=0;i<21;i++)
{
        if(no[i]<0)
        {
                flag5=1;
                no[i]=0.000000001;      // if any guess value is negative make it a
                                        small positive
        }
}
if(flag5==0)
{
        for(i=0;i<21;i++)
        {
                no1[i]=no[i];       // update guess vector
        }
}
vector(vec,no,r, K,row,&molsum,fuelcomp,P);          // Calculate F[]
newt(no,row,&check,vec,K,P,&molsum,r,fuelcomp);              // Solve system
for(i=0; i<row; i++)
{
        molfrac[i]=no[i]/molsum;            // Calculate mole fraction
}

for(sum=0, i=0; i<row; i++)
{
        sum=sum+molfrac[i];       // add mole fractions to check that the sum to 1
}
flag5=0;
for(i=0;i<21;i++)
{
        if(no[i]<0)
        {
                flag5=1;            // if any mole fractions are negative set flag to
                                    decrease step
        }
}
if(flag5==0)
{
        step1=step;         // save step size and phi for normal step size
        phi1=phis;
}
if(flag5==1 && step==step1)   // change step size if new step is the same as the old step
{
        step1=.5*step1;    // change step size
        for(i=0;i<21;i++)
        {
                no[i]=no1[i];       // update guess
        }
}
if(flag5==1 && step1<step)    // change step size if new step is smaller than the old step
{
        step1=.5*step1;    // change step size
```

```
                for(i=0;i<21;i++)
                {
                        no[i]=no1[i];       // update guess
                }
        }
        if(flag5==0)
        {
                phis=phis-step1;            // new equivalence ratio using normal step size
        }
        else
        {
                phis=phis-(phis-phi1)/2;    // new equivalence ratio for diminishing step size
        }
        if(flag==0)
        {

                if(phis>end || flag4==0)    // controls end of phi loop for decreasing phi
                {
                        flag3=0;
                }else
                {
                        flag3=1;
                }
                if(phis<end+tol)  // insures last loop is at target phi
                {
                        phis=end;
                        flag4=1;
                }
        }
        if(flag==1)
        {
                if(phis<end || flag4==0)    // controls end of phi loop for increasing phi
                {
                        flag3=0;
                }else
                {
                        flag3=1;
                }
                if(phis>end)        // insures last loop is at target phi
                {
                        phis=end;
                        flag4=1;
                }
        }
        if(flag1==1)
        {
                flag3=1;
        }
}while(flag3==0);           // ends phi loop

for(i=0;i<21;i++)
{
        no1[i]=no[i];       // update guess
}
enthalpy (T, h, Cp);        // calculate enthalpy of the products
hp=0;
```

```
	for (i=0;i<21;i++)
	{
		hx[i]=molfrac[i]*h[i];
		x[i]=molfrac[i];
		hp=hp+hx[i];
	}
	free(vec), (no), (molfrac), (K);	// Free up allocated memory

	return hp;		// return to calling program
}
/*-------------------------------------------------------------*/
/* equilibrium constant curve fits (Kp)                        */

void Kp(double K[], double T)

{
	// declare variables
	double Tr, logK[16];
	int i;

	Tr=.001*T;
		// reactions
	logK[0]=2*(  .432168  *log(Tr) - 11.2464  /Tr + 2.67269   - .0745744 *Tr + .00242484*Tr*Tr);
		// R1,  H2 <-> 2H
	logK[1]=2*(  .310805  *log(Tr) - 12.9540  /Tr + 3.21779   - .0738336 *Tr + .00344645*Tr*Tr);
		// R2,  O2 <-> 2O
	logK[2]=2*(  .389716  *log(Tr) - 24.5828  /Tr + 3.14505   - .0963730 *Tr + .00585643*Tr*Tr);
		// R3,  N2 <-> 2N
	logK[3]=2*(- .141784  *log(Tr) -  2.13308 /Tr +  .853461  + .0355015 *Tr - .00310227*Tr*Tr);
		// R4,  O2 + H2 <-> 2OH
	logK[4]=2*(  .0150879 *log(Tr) -  4.70959 /Tr +  .646096  + .00272805*Tr - .00154444*Tr*Tr);
		// R5,  N2 + O2 <-> 2NO
	logK[5]=2*(- .752364  *log(Tr) + 12.4210  /Tr  - 2.60286   + .259556  *Tr  - .0162687 *Tr*Tr);
		// R6,  2H2 + O2 <-> 2H2O
	logK[6]=2*(- .00415302*log(Tr) + 14.8627  /Tr  - 4.75746   + .124699  *Tr- .00900227*Tr*Tr);
		// R7,  2CO + O2 <-> 2CO2
	logK[7]=   -1.284553  *log(Tr)  - 15.76395 /Tr - 4.918345  + .6842445 *Tr - .05194576*Tr*Tr;
		// R8,  CO + 2H2 <-> CH3 + OH
	logK[8]=   - .9494526 *log(Tr) + 22.87447 /Tr - 7.256777  + .3787896 *Tr  - .02448167*Tr*Tr;
		// R9,  CH3 + H <-> CH4
	logK[9]=   - .2233993 *log(Tr)  - 23.31895 /Tr - 6.766712  + .3293027 *Tr - .01764268*Tr*Tr;
		// R10, 2CO + H2 <-> C2H2 + O2
	logK[10]=   2.066455  *log(Tr) + 10.77027 /Tr  - 6.962663  -1.308306  *Tr + .1346954 *Tr*Tr;
		// R11, C2H2 + H2 <-> C2H4
	logK[11]=  - .8377514 *log(Tr) + 15.78394 /Tr - 5.150921  + .2656375 *Tr - .01666026*Tr*Tr;
		// R12, N + H2 <-> NH2
	logK[12]=  -1.005302  *log(Tr) + 22.32473 /Tr - 6.636705  + .3923653 *Tr - .02239234*Tr*Tr;
		// R13, NH2 + H <-> NH3
	logK[13]=    .9494541 *log(Tr) + 25.60553 /Tr -  .5112236 - .5607905 *Tr + .04848174*Tr*Tr;
		// R14, CH3 + N <-> HCN + H2
	logK[14]=  -1.284553  *log(Tr) +  5.776046/Tr - 4.917345  + .5192443 *Tr - .03794574*Tr*Tr;
		// R15, CO + H <-> HCO
	logK[15]=  -1.005301  *log(Tr) - 25.49527 /Tr - 5.314705  + .4793645 *Tr  - .02839226*Tr*Tr;
		// R16, 2CO + H <-> C2H + O2

	for (i=0;i<16;i++)
```

```
        {
                K[i]=pow(10,logK[i]);
        }
        return;
}
/*-------------------------------------------------------------*/
void guess(double no[], double K[], double fuelcomp[], double P)
{
        /* variable definitions
          r[4] ------- constants to simplify stoichiometry
          Nsum ------- total number of moles guess
          nf --------- number of moles of fuel
          f1 --------- reaction equat ion
          f2 --------- reaction equation
          phig ------- equivalence ratio for the guess
          ox --------- updated oxygen guess
          oxrt ------- square root of oxygen guess
          fox -------- oxygen function
          dox -------- derivative of fox
          rat -------- ratio of fox to dox
          z ---------- calculation variable
          n ---------- # C atoms in fuel
          m ---------- # H atoms in fuel
          l ---------- # O atoms in fuel
          k ---------- # Natoms in fuel
          ind -------- keeps track of number of loops */
        double r[4], Nsum, nf=1, f1, f2, phig=1;
        double ox, oxrt, fox, dox, rat, z, n, m, l, k;
        int ind;

        n=fuelcomp[0];   // assign fuel composition
        m=fuelcomp[1];
        l=fuelcomp[2];
        k=fuelcomp[3];
        r[0]=(n+0.25*m-0.5*l)/phig;          //r0                  // calculate stoichiometry constants
        r[1]=0.5*l+r[0];                      //r
        r[2]=0.5*k+3.7274*r[0];               //r'
        r[3]=0.0444*r[0];                     //r"
        if (phig > 1)
        {
                Nsum=nf*(r[2]+r[3]+n+.5*m);      // guess of total number of moles for equivalence
                                                 ratios above 1
        }
        else
        {
                Nsum=nf*(r[1]+r[2]+r[3]+.25*m); // guess of total number of moles for equivalence
                                                 ratios below 1
        }
        ox=1;    // first guess of moles of oxygen
        oxrt=sqrt(ox);
        f1=sqrt(K[5]*P/Nsum);     // reaction rates
        f2=sqrt(K[6]*P/Nsum);
        fox=.5*m*f1*oxrt/(1+f1*oxrt)+(n+2*n*f2*oxrt)/(1+f2*oxrt)+2*ox/nf-r[1];
                // oxygen function
        ind=1;
        while (fabs(fox)>.000001)
```

```
	{
		if (fox > 0)
		{
			ox=.1*ox;		// new oxygen value if oxygen function greater than zero
			oxrt=sqrt(ox);
		}
		if (fox < 0)
		{
			dox=.25*m*f1/(oxrt*(1+f1*oxrt)*(1+f1*oxrt))+
				(n*f2-1)/(oxrt*(1+f2*oxrt)*(1+f2*oxrt))+2/nf;		// derivative of
												oxygen function
			rat=fox/dox;
			ox=ox-rat;		// new oxygen value if oxygen function less than zero
			oxrt=sqrt(ox);
			z=rat/ox;
		}
		fox=.5*m*f1*oxrt/(1+f1*oxrt)+(n+2*n*f2*oxrt)/(1+f2*oxrt)+2*ox/nf-r[1];
			// recalculate oxygen function
		ind++;
	}
	no[3]=.5*nf*m/(1+f1*oxrt);		// calculate all mole fraction guesses
	no[5]=nf*n/(1+f2*oxrt);
	no[7]=ox;
	no[10]=nf*r[2];
	no[11]=nf*r[3];
	no[8]=sqrt(K[5]*no[3]*no[3]*no[7]*P/Nsum);
	no[9]=sqrt(K[6]*no[5]*no[5]*no[7]*P/Nsum);
	no[0]=sqrt(K[0]*no[3]*P/Nsum);
	no[1]=sqrt(K[1]*no[7]*P/Nsum);
	no[2]=sqrt(K[2]*no[10]*P/Nsum);
	no[4]=sqrt(K[3]*no[3]*no[7]);
	no[6]=sqrt(K[4]*no[7]*no[10]);
	no[12]=K[7]*no[3]*no[3]*no[5]/no[4]*P/Nsum;
	no[13]=K[8]*no[0]*no[12]*P/Nsum;
	no[14]=K[9]*no[3]*no[5]*no[5]/no[7]*P/Nsum;
	no[15]=K[10]*no[3]*no[14]*P/Nsum;
	no[16]=K[11]*no[2]*no[3]*P/Nsum;
	no[17]=K[12]*no[0]*no[16]*P/Nsum;
	no[18]=K[13]*no[2]*no[12]/no[3];
	no[19]=K[14]*no[0]*no[5]*P/Nsum;
	no[20]=K[15]*no[0]*no[5]*no[5]/no[7]*P/Nsum;

	return;	// return to equilibrium function
}
/*****************************************************************************
Calculates solution vector G[]-F[]=-F[]
*****************************************************************************/
void vector(double vec[], double no[], double r[], double K[], int row, double *molsum, double fuelcomp[],
double P)
{
	int i;
	double nf=1, n, m, l, k;

	n=fuelcomp[0];
	m=fuelcomp[1];
	l=fuelcomp[2];
```

```
	k=fuelcomp[3];

	//Calculate mole sum
	for(*molsum=0, i=0; i<row; i++)
		*molsum+=no[i];

	vec[0] =-1*(no[0]*no[0]*(P/(*molsum))-K[0]*no[3]);		//R1, H2 <=> 2H
	vec[1] =-1*(no[1]*no[1]*(P/(*molsum))-K[1]*no[7]);		//R2, O2 <=> 2O
	vec[2] =-1*(no[2]*no[2]*(P/(*molsum))-K[2]*no[10]);		//R3, N2 <=> 2N
	vec[3] =-1*(no[8]*no[8]-K[5]*no[3]*no[3]*no[7]*(P/(*molsum)));	//R6, 2 H2 + O2 <=> 2 H2O
	vec[4] =-1*(no[4]*no[4]-K[3]*no[3]*no[7]);		//R4, H2 + O2 <=> 2 OH
	vec[5] =-1*(no[9]*no[9]-K[6]*no[5]*no[5]*no[7]*(P/(*molsum)));	//R7, 2 CO + O2 <=> 2 CO2
	vec[6] =-1*(no[6]*no[6]-K[4]*no[7]*no[10]);		//R5, O2 + N2 <=> 2 NO
	vec[7] =-1*(no[7]*no[20]*(*molsum/P)-K[15]*no[0]*no[5]*no[5]);	//R16, 2 CO + H <=> C2H
									+ O2
	vec[8] =-1*(no[0]+2*no[3]+no[4]+2*no[8]+3*no[12]+4*no[13]+2*no[14]+
		4*no[15]+2*no[16]+3*no[17]+no[18]+no[19]+no[20]-m*nf);	//Hydrogen balance
	vec[9] =-1*(no[1]+no[4]+no[5]+no[6]+2*no[7]+no[8]+2*no[9]+no[19]-2*r[1]*nf-l);
		//Oxygen balance
	vec[10]=-1*(no[2]+no[6]+2*no[10]+no[16]+no[17]+no[18]-2*r[2]*nf-k);
		//Nitrogen balance
	vec[11]=-1*(no[11]-r[3]*nf);		//Argon balance
	vec[12]=-1*(no[4]*no[12]*(*molsum/P)-K[7]*no[3]*no[3]*no[5]);	//R8, CO + 2 H2 <=> CH3
									+ OH
	vec[13]=-1*(no[13]*(*molsum/P)-K[8]*no[0]*no[12]);		//R9, CH3 + H <=> CH4
	vec[14]=-1*(no[7]*no[14]*(*molsum/P)-K[9]*no[3]*no[5]*no[5]);	//R10, 2 CO + H2 <=>
									C2H2 + O2
	vec[15]=-1*(no[15]*(*molsum/P)-K[10]*no[3]*no[14]);		//R11, C2H2 + H2 <=> C2H4
	vec[16]=-1*(no[16]*(*molsum/P)-K[11]*no[2]*no[3]);		//R12, N + H2 <=> NH2
	vec[17]=-1*(no[17]*(*molsum/P)-K[12]*no[0]*no[16]);		//R13, NH2 + H <=> NH3
	vec[18]=-1*(no[3]*no[18]-K[13]*no[2]*no[12]);		//R14, CH3 + N <=> HCN + H2
	vec[19]=-1*(no[19]*(*molsum/P)-K[14]*no[0]*no[5]);		//R15, CO + H <=> HCO
	vec[20]=-1*(no[5]+no[9]+no[12]+no[13]+2*no[14]+2*no[15]+no[18]+no[19]+ 2*no[20]-n*nf);
		//Carbon balance
}
/*****************************************************************************
From numerical Recipes in C
*****************************************************************************/
void newt(double x[],int n,int *check,double fvec[],double K[],double P,double *molsum,double r[],
double fuelcomp[])
{
	int i,its,j,*indx;
	double d,den,f,fold,stpmax,sum,temp,test,**fjac,*g,*p,*xold;

	void vector(double vec[], double no[], double r[], double K[], int row, double *molsum,
		double fuelcomp[], double P);
	double fmin(int n,double fvec[]);
	void lnsrch(int n,double xold[],double fold,double g[],double p[],double x[],
		double *f,double stpmax,int *check,double fvec[],double r[],double K[],
		double *molsum, double fuelcomp[], double P);
	void ludcmp(double *a[],int n,int *indx,double *d);
	void lubksb(double *a[],int n,int *indx,double b[]);
	void Kp(double K[], double T);
	void enthalpy(double T,  double h[], double Cp[]);

	//Dynamic memory allocation
```

```
indx=(int*)malloc(n*sizeof(int));
fjac=(double**)malloc(n*sizeof(double*));		//Set up the rows of fjac
for(i=0; i<n; i++)				//fjac is the Jacobian matrix
	fjac[i]=(double*)malloc(n*sizeof(double));		//Set up the columns of fjac
g=(double*)malloc(n*sizeof(double));
p=(double*)malloc(n*sizeof(double));
xold=(double*)malloc(n*sizeof(double));

vector(fvec,x,r,K,n,molsum,fuelcomp,P);
//Calculate fvec based on latest estimate
f=fmin(n,fvec);		//Find fmin=0.5*F*F

test=0.0;		//Test for initial guess being a root. Use more stringent
for(i=0; i<n; i++)		//test than simply TOLF.
	if(fabs(fvec[i]) > test)
		test=fabs(fvec[i]);
if(test<0.01*TOLF)
	FREERETURN

for(sum=0.0, i=0; i<n; i++)
	sum = sum + x[i]*x[i];	//Calculate stpmax for line searches, sum = sum + SQR(x[i]);
stpmax=STPMX*__max(sqrt(sum),(double)n);	//Compared variables must be of same type.
for(its=1; its<=MAXITS; its++)
	{
	for(i=0; i<n; i++)
		for(j=0; j<n; j++)
			fjac[i][j]=0.0;	//Initialize Jacobian to zero
	//Fill in Jacobian matrix
	fjac[0][0]=2*x[0]*(P/(*molsum));		//R1
	fjac[0][3]=-K[0];
	fjac[1][1]=2*x[1]*(P/(*molsum));		//R2
	fjac[1][7]=-K[1];
	fjac[2][2]=2*x[2]*(P/(*molsum));		//R3
	fjac[2][10]=-K[2];
	fjac[3][3]=-K[5]*2*x[3]*x[7]*(P/(*molsum));	//R6
	fjac[3][7]=-K[5]*x[3]*x[3]*(P/(*molsum));
	fjac[3][8]=2*x[8];
	fjac[4][3]=-K[3]*x[7];	//R4
	fjac[4][4]=2*x[4];
	fjac[4][7]=-K[3]*x[3];
	fjac[5][5]=-K[6]*2*x[5]*x[7]*(P/(*molsum));	//R7
	fjac[5][7]=-K[6]*x[5]*x[5]*(P/(*molsum));
	fjac[5][9]=2*x[9];
	fjac[6][6]=2*x[6];		//R5
	fjac[6][7]=-K[4]*x[10];
	fjac[6][10]=-K[4]*x[7];
	fjac[7][0]=-K[15]*x[5]*x[5];	//R16
	fjac[7][5]=2*K[15]*x[0]*x[5];
	fjac[7][7]=x[20]*(*molsum/P);
	fjac[7][20]=x[7]*(*molsum/P);
	fjac[8][0]=1;		//Hydrogen balance
	fjac[8][3]=2;
	fjac[8][4]=1;
	fjac[8][8]=2;
	fjac[8][12]=3;
	fjac[8][13]=4;
```

```
fjac[8][14]=2;
fjac[8][15]=4;
fjac[8][16]=2;
fjac[8][17]=3;
fjac[8][18]=1;
fjac[8][19]=1;
fjac[8][20]=1;
fjac[9][1]=1;           //Oxygen balance
fjac[9][4]=1;
fjac[9][5]=1;
fjac[9][6]=1;
fjac[9][7]=2;
fjac[9][8]=1;
fjac[9][9]=2;
fjac[9][19]=1;
fjac[10][2]=1;          //Nitrogen balance
fjac[10][6]=1;
fjac[10][10]=2;
fjac[10][16]=1;
fjac[10][17]=1;
fjac[10][18]=1;
fjac[11][11]=1;             //Argon balance
fjac[12][3]=-K[7]*2*x[3]*x[5];              //R8
fjac[12][4]=x[12]*(*molsum/P);
fjac[12][5]=-K[7]*x[3]*x[3];
fjac[12][12]=x[4]*(*molsum/P);
fjac[13][0]=-K[8]*x[12];                //R9
fjac[13][12]=-K[8]*x[0];
fjac[13][13]=(*molsum/P);
fjac[14][3]=-K[9]*x[5]*x[5];                    //R10
fjac[14][5]=-K[9]*2*x[5]*x[3];
fjac[14][7]=x[14]*(*molsum/P);
fjac[14][14]=x[7]*(*molsum/P);
fjac[15][3]=-K[10]*x[14];               //R11
fjac[15][14]=-K[10]*x[3];
fjac[15][15]=(*molsum/P);
fjac[16][2]=-K[11]*x[3];                //R12
fjac[16][3]=-K[11]*x[2];
fjac[16][16]=(*molsum/P);
fjac[17][0]=-K[12]*x[16];               //R13
fjac[17][16]=-K[12]*x[0];
fjac[17][17]=(*molsum/P);
fjac[18][2]=-K[13]*x[12];               //R14
fjac[18][3]=x[18];
fjac[18][12]=-K[13]*x[2];
fjac[18][18]=x[3];
fjac[19][0]=-K[14]*x[5];                //R15
fjac[19][5]=-K[14]*x[0];
fjac[19][19]=(*molsum/P);
fjac[20][5]=1;              //Carbon balance
fjac[20][9]=1;
fjac[20][12]=1;
fjac[20][13]=1;
fjac[20][14]=2;
fjac[20][15]=2;
fjac[20][18]=1;
```

```
		fjac[20][19]=1;
		fjac[20][20]=2;

		for(i=0; i<n; i++)		//Compute f-gradient for line search.
			{
			for(sum=0.0, j=0; j<n; j++)
				sum = sum + fjac[j][i]*fvec[j];
			g[i]=sum;
			}
		for(i=0; i<n; i++)
			xold[i]=x[i];			//Store x[] in xold[]
		fold=f;
	//Store f=fmin in fold
		for(i=0; i<n; i++)			//RHS for linear equations.
			p[i] = fvec[i];
		ludcmp(fjac,n,indx,&d);		//Solve using LUD
		lubksb(fjac,n,indx,p);		//Back substitution
		lnsrch(n,xold,fold,g,p,x,&f,stpmax,check,fvec,r,K,molsum,fuelcomp,P); //lnsrch returns
                                                                        new x[] and f. Calculate fvec at new x[].
		test=0.0;		//Test for convergence on function values.
		for(i=0; i<n; i++)
			if(fabs(fvec[i]) > test)
				test=fabs(fvec[i]);
		if(test < TOLF)
			{
			check=0;		//check is false (0) on a normal exit
			FREERETURN
			}
		if(*check)		//check is true (1) when x is too close to xold
			{		//Check for gradient of f zero, spurious convergence.
			test=0.0;
			den=__max(f,0.5*n);
			for(i=0; i<n; i++)
				{
				temp=fabs(g[i])*__max(fabs(x[i]),1.0)/den;
				if(temp > test)
					test=temp;
				}
			*check=(test < TOLMIN ? 1 : 0);
			FREERETURN
			}
		test=0.0;		//Test for convergence on delta x
		for(i=1; i<=n; i++)
			{
			temp=(fabs(x[i-1]-xold[i-1]))/__max(fabs(x[i-1]),1.0);
			if(temp > test)
				test=temp;
			}
		if(test < TOLX)
			FREERETURN
		}
	//printf("\n MAXITS exceeded in newt");
}
/*************************************************************************
fmin calculates (1/2)*FdotF
*************************************************************************/
```

```
double fmin(int n, double fvec[])
{
	int i;
	double sum;

	for(sum=0.0, i=0; i<n; i++)
		sum = sum + fvec[i]*fvec[i];
	return 0.5*sum;
}
/*************************************************************************
From Numerical recipes in C
*************************************************************************/
void lnsrch(int n,double xold[],double fold,double g[],double p[],double x[],double *f,
	double stpmax,int *check,double fvec[],double r[],double K[],double *molsum,
	double fuelcomp[], double P)
{
	int i;
	double a,alam,alam2,alamin,b,disc,f2,fold2,rhs1,rhs2;
	double slope,sum,temp,test,tmplam;

	void vector(double vec[], double no[], double r[], double K[], int row, double *molsum,
		double fuelcomp[], double P);

	*check=0;			//Normal exit
	for(sum=0.0, i=0; i<n; i++)
		sum += p[i]*p[i];
	sum=sqrt(sum);

	if(sum > stpmax)
		{
		for(i=0; i<n; i++)
			p[i] *= stpmax/sum;
		}
	//Scale if attempted step is too big.
	for(slope=0.0, i=0; i<n; i++)
		slope += g[i]*p[i];
	test=0.0;		//Compute lambda_min
	for(i=0; i<n; i++)
		{
		temp=fabs(p[i]) / __max(fabs(xold[i]),1.0);   //temp=fabs(p[i])/FMAX(fabs(xold[i]),1.0);
		if(temp > test)
			test=temp;
		}
	alamin=TOLX/test;		//alamin is lambda_min
	alam=1.0;		//Try full step first, alam is lambda
	for(;;)
		{
		for(i=0; i<n; i++)
			x[i]=xold[i]+alam*p[i];		//This loop updates x[]
		vector(fvec,x,r,K,n,molsum,fuelcomp,P);
		*f=fmin(n,fvec);
		if(alam < alamin)
			{			//Convergence on delta_x
			for(i=0; i<n; i++)
				x[i]=xold[i];
			*check=1;
```

```
                return;
                }
        else if(*f <= fold+ALPHA*alam*slope)       //Eqn. 9.7.7
                {
                return;                         //Sufficient function decrease
                }
        else
                {               //Backtrack
                if(alam == 1.0)
                        tmplam = -slope/(2.0*(*f-fold-slope));          //First time
                else
                        {       //Subsequent backtracks
                        rhs1=*f-fold-alam*slope;
                        rhs2=f2-fold2-alam2*slope;
                        a=(rhs1/(alam*alam)-rhs2/(alam2*alam2))/(alam-alam2);
                        b=(-alam2*rhs1/(alam*alam)+alam*rhs2/(alam2*alam2))/
                                (alam-alam2);
                        if(a == 0.0)
                                tmplam = -slope/(2.0*b);
                        else
                                {
                                disc=b*b-3.0*a*slope;
                                if(disc<0.0)
                                {
                                        //printf("\nRoundoff problem in lnsrch.");
                                }
                                else
                                        tmplam=(-b+sqrt(disc))/(3.0*a);
                                }
                        if(tmplam>0.5*alam)
                                tmplam=0.5*alam;
                        }
                }
        alam2=alam;
        f2 = *f;
        fold2=fold;
        alam=__max(tmplam,0.1*alam);
        }
}
/*************************************************************************
From Numerical recipes in C
*************************************************************************/
void ludcmp(double *a[], int n, int *indx, double *d)
{
        int i, imax, j, k;
        double big, dum, sum, temp;
        double *vv;
        //vv stores the implicit scaling of each row

        vv=(double*)malloc(n*sizeof(double));
        *d=1.0;
        //No row interchanges yet
        for(i=1; i<=n; i++)         //Loops over the rows to get the implicit scaling information
                {
                big=0.0;
                for(j=1; j<=n; j++)
```

```
		{
		if((temp=fabs(a[i-1][j-1])) > big)
			big=temp;
		}
	if(big == 0.0)
	{
		printf("\nSingular matrix");
	}	//No no-zero largest element
	vv[i-1]=1.0/big;		//Save scaling
	}
for(j=1; j<=n; j++)		//This is the loop over columns of Crout's method
	{
	for(i=1; i<j; i++)		//Eqn. 2.3.12 except for i=j
		{
		sum=a[i-1][j-1];
		for(k=1; k<i; k++)
			sum-=a[i-1][k-1]*a[k-1][j-1];
		a[i-1][j-1]=sum;
		}
		big=0.0;		//Initialize for the search for largest pivot element
		for(i=j; i<=n; i++)	//This is i=j of eqn. 2.3.12 and i=j+1...N of eqn. 2.3.13
			{
			sum=a[i-1][j-1];
			for(k=1; k<j; k++)
				sum-=a[i-1][k-1]*a[k-1][j-1];
			a[i-1][j-1]=sum;
			if((dum=vv[i-1]*fabs(sum)) >= big)		//Is the figure of merit for
									the pivot better than the best so far?
				{
				big=dum;
				imax=i;
				}
			}
		if(j != imax)		//Do we need to interchange rows?
			{
			for(k=1; k<=n; k++)		//Yes, do so...
				{
				dum=a[imax-1][k-1];
				a[imax-1][k-1]=a[j-1][k-1];
				a[j-1][k-1]=dum;
				}
			*d=-(*d);		//...and change the parity of d
			vv[imax-1]=vv[j-1];		//Also interchange the scale factor
			}
		indx[j-1]=imax;
		if(a[j-1][j-1] == 0.0)		//If the pivot element is zero the matrix is singular
						(at least to the precision of
			a[j-1][j-1]=TINY;		//the algorithm). For some applications on
						singular matrices, it is desirable
						//to substitute TINY for zero.
		if(j != n)		//Now, finally, divide by the pivot element
			{
			dum=1.0/(a[j-1][j-1]);
			for(i=j+1; i<=n; i++)
				a[i-1][j-1] *= dum;
			}
```

```
		}			//Go back for the next column in the reduction.
		free(vv);
}
/*************************************
From Numerical recipes in C
*************************************/
void lubksb(double *a[], int n, int *indx, double b[])
{
	int i, ii=0, ip, j;
	double sum;

	for(i=1; i<=n; i++)			//When ii is set to a positive value, it will
		{
	//become the index of the first non-vanishing
		ip=indx[i-1];
	//element of b. We now do the forward substitution,
		sum=b[ip-1];			//eqn. 2.3.6. The only new wrinkle is to unscramble
		b[ip-1]=b[i-1];			//the permutation as we go.
		if(ii)
			for(j=ii; j<=i-1; j++)
				sum -= a[i-1][j-1]*b[j-1];
		else if(sum)			//A non-zero element was encountered, so from now
			ii=i;			//on we will have to do the sums in the loop above.
		b[i-1]=sum;
		}
	for(i=n; i>=1; i--)			//Now we do the back substitution, eqn. 2.3.7.
		{
		sum=b[i-1];
		for(j=i+1; j<=n; j++)
			sum -= a[i-1][j-1]*b[j-1];
		b[i-1]=sum/a[i-1][i-1];			//Store a component of the solution vector X.
		}
	//All done!
}

/*-----------------------------------------------------------------*/
/* this function returns fuel properties given a fuel name    */

void properties(int name, double fuel[], double fuelcomp[])
{
	// retrieve data

	/* fuel[0]=Cpfuel   kJ/kg*K                    */
	/* fuel[1]=Tboil    K                          */
	/* fuel[2]=hfg      kJ/kg                      */
	/* fuel[3]=rhofuel  kg/m^3                     */
	/* fuel[4]=cetane #                            */
	/* fuel[5]=heat of formation kJ/kmol           */
	/* fuel[6]=heating value  kJ/kg                */
	/* fuel[7]=molecular weight kg/kmol            */
	/* fuelcomp[0]=# carbon                        */
	/* fuelcomp[1]=# hydrogen                      */
	/* fuelcomp[2]=# oxygen                        */
	/* fuelcomp[3]=# nitrogen                      */

	switch (name)
```

```
{
	case 1:// n-heptane
		fuel[0]=2.242;
		fuel[1]=361;
		fuel[2]=364.9;
		fuel[3]=683.76;
		fuel[4]=50;
		fuel[5]=-224050;
		fuel[6]=44556;
		fuel[7]=94;
		fuelcomp[0]=7;
		fuelcomp[1]=16;
		fuelcomp[2]=0;
		fuelcomp[3]=0;
		break;
	case 2:// n-hexadecane
		fuel[0]=3.158;
		fuel[1]=560;
		fuel[2]=227;
		fuel[3]=770;
		fuel[4]=100;
		fuel[5]=-456140;
		fuel[6]=43946;
		fuel[7]=226.44;
		fuelcomp[0]=16;
		fuelcomp[1]=34;
		fuelcomp[2]=0;
		fuelcomp[3]=0;
		break;
	case 3:// heptamethylnonane
		fuel[0]=3.424;
		fuel[1]=520;
		fuel[2]=194;
		fuel[3]=811;
		fuel[4]=15;
		fuel[5]=-413200;
		fuel[6]=43852;
		fuel[7]=226.44;
		fuelcomp[0]=16;
		fuelcomp[1]=34;
		fuelcomp[2]=0;
		fuelcomp[3]=0;
		break;
	case 4:// methanol
		fuel[0]=2.824;
		fuel[1]=338;
		fuel[2]=1100;
		fuel[3]=790;
		fuel[4]=5;
		fuel[5]=-239100;
		fuel[6]=19917;
		fuel[7]=32.04;
		fuelcomp[0]=1;
		fuelcomp[1]=4;
		fuelcomp[2]=1;
		fuelcomp[3]=0;
```

```
            break;
case 5:// diethyl ether
            fuel[0]=2.369;
            fuel[1]=307.6;
            fuel[2]=365.6;
            fuel[3]=713.8;
            fuel[4]=115;
            fuel[5]=-279400;
            fuel[6]=33775;
            fuel[7]=72;
            fuelcomp[0]=4;
            fuelcomp[1]=10;
            fuelcomp[2]=1;
            fuelcomp[3]=0;
            break;
case 6:// HCG
            fuel[0]=2.235;
            fuel[1]=421;
            fuel[2]=335;
            fuel[3]=737;
            fuel[4]=50;
            fuel[5]=-350000;
            fuel[6]=44000;
            fuel[7]=110;
            fuelcomp[0]=100;
            fuelcomp[1]=187;
            fuelcomp[2]=0;
            fuelcomp[3]=0;
            break;
case 7:// FTD
            fuel[0]=3.197;
            fuel[1]=611;
            fuel[2]=224;
            fuel[3]=782;
            fuel[4]=50;
            fuel[5]=-456140;     // # for cetane
            fuel[6]=43200;
            fuel[7]=170;
            fuelcomp[0]=10;
            fuelcomp[1]=18;
            fuelcomp[2]=0;
            fuelcomp[3]=0;
            break;
case 8:// DF2
            fuel[0]=3.197;
            fuel[1]=576;
            fuel[2]=224;
            fuel[3]=712;//845;
            fuel[4]=50;
            fuel[5]=-180974;     // # calculated
            fuel[6]=43200;
            fuel[7]=151;
            fuelcomp[0]=11;
            fuelcomp[1]=19;
            fuelcomp[2]=0;
            fuelcomp[3]=0;
```

```
                        break;
                default:
                        break;
        }
}
/*-------------------------------------------------------------------------------*/
/* this function calculates the stoichiometric air fuel ratio of the selected    */
/* fuel based on fuel and air compositions                                       */

double airfuel(double fuelcomp[])
{
        // declare vari ables
        double AFs;     // calculate molar stoichiometric air fuel ratio
        AFs=(fuelcomp[0]+fuelcomp[1]*.25-fuelcomp[2]*.5)*(1+3.7274+.0444);
        return AFs;     // return air fuel ratio
}
/*-----------------------------------------------------------------------*/
/* liquid length model                                                   */

void liquidl(double ll[], double fuel[], double fuelcomp[], double Tinj, double T, double rhoa, double AFs,
double t, double Ufuel, int inj, double MWa, double Cpa, double te, double dh, double angle, double l1)
{
        /* variable definitions
          B ---------- liquid length energy ratio
          c ---------- constant used to find spread angle
          tplus ------ non dimensionalizing time constant
          ttilda ----- non dimensional time for spray model at a given time
          xtilda ----- non dimensional penetration length for spray model at a given time
          x ---------- spray penetration length at a given time
          tL --------- time to reach liquid length
          tx1 -------- time to reach given length
          tetilda ---- non dimensional time for spray model at a given time for back end of spray
          xe --------- length to back end of spray
          xetilda ---- non dimensional penetration length for spray model at a given time for back end of
                       spray
          rhotilda --- non dimensionalized density (fuel to air density ratio)
          spreadangle  spray spread angle
          xplus ------ non dimensionalizing length constant
          Ca --------- nozzle area contraction coefficient
          dorifice --- nozzle orifice diameter
          phix ------- equivalence ratio at the spray length
          a ---------- spread angle multiplier
          L ---------- liquid length
          k ---------- liquid length correlation constant
          alpha ------ liquid length correlation constant
          beta ------- liquid length correlation constant
          Ltilda ----- non dimensional liquid length
          z ---------- spray model variable used to find equivalence ratios
          phiL ------- equivalence ratio at the liquid length
          ttildaL ---- non dimensional time for liquid length
          ttildax1 --- non dimensional time for given spray length
          injdata[3] - injector parameters retrieved from the injector function
          phix1 ------ equivalence ratio at a given length
          x1tilda ---- non dimensional penetration length for given spray lengt h
          xtilda0 ---- first guess of non dimensional penetration length for solving spray correlation using
                       the secant method
```

```
  xtilda1 ---- second guess of non dimensional penetration length for solving spray correlation
               using the secant method
  xtilda2 ---- subsequent guesses of non dimensional penetration length for solving spray
               correlation using the secant method
  q0 --------- secant method function
  q1 --------- secant method function
  dq --------- change in secant method function */

// declare variables
double B, c=.31, tplus, ttilda, xtilda, x, tL, tx1, tetilda, xe, xetilda;
double rhotilda, spreadangle, xplus, Ca, dorifice, phix;
double a=.66, L, k=10.5, alpha=.58, beta=.59, Ltilda, z, phiL, ttildaL, t tildax1;
double injdata[3], phix1, x1tilda, xtilda0=50, xtilda1=0, xtilda2, q0, q1, dq;

/* function prototypes
   injector function holds data on different injector nozzles
   airfuel function retrieves the molar stoichiometric air fuel ratio */
void injector(int inj, double injdata[]);
double airfuel(double fuelcomp[]);

injector(inj,injdata);          // retrieve injector data
dorifice=injdata[0]/1000;  // change units to mm
Ca=injdata[2];
AFs=airfuel(fuelcomp);               // get molar stoichiometric air fuel ratio
AFs=AFs*MWa/fuel[7];   // change air fuel ratio to mass basis

// liquid length calculations
if(dh==0)
{
          Cpa=Cpa/MWa;               // calculate specific heat of air
          dh=Cpa*(T-fuel[1]);        // change in enthalpy for liquid length model
}
B=(fuel[0]*(fuel[1]-Tinj)+fuel[2])/dh;         // liquid length energy ratio
rhotilda=fuel[3]/rhoa;        // density ratio
if(angle==0)
{
          spreadangle=c*(pow(1/rhotilda,.19)-.0043*sqrt(rhotilda));      // spray spread angle if
                                                                         angle not previously calculated
}
else
{
          spreadangle=angle;          // spray spread angle if previously calculated (insures angle
                                      doesnt change)
}
xplus=sqrt(rhotilda)*((sqrt(Ca)*dorifice)/(a*spreadangle));
// calculate non dimensionalizing length constant
L=dorifice*k*(pow(rhotilda,alpha))*(pow(B,beta));
// liquid length calculation
Ltilda=L/xplus;   // non dimensional liquid length
z=1+16*(Ltilda*Ltilda);               // equivalence ratio constant
phiL=(sqrt(z)-1)/2;           // equivalence ratio at liquid length
tplus=xplus/(Ufuel);          // calculate non dimensionalizing time constant
ttilda=t/tplus;       // non dimensionalized time
ttildaL=Ltilda/2+(Ltilda/4)*sqrt(1+16*Ltilda*Ltilda)+(1/16)*log(4*Ltilda+
          sqrt(1+16*Ltilda*Ltilda)); // non dimensional time to reach liquid length
tL=ttildaL*tplus;             // time to reach liquid length
```

```
// calculate spray penetration length and phi at that length given the time included in the
        function prototype
q0=xtilda0/2+(xtilda0/4)*sqrt(1+16*xtilda0*xtilda0)+(1/16)*log(4*xtilda0+
        sqrt(1+16*xtilda0*xtilda0))-ttilda; // secant function at first penetration length guess for
                                            a given time
q1=xtilda1/2+(xtilda1/4)*sqrt(1+16*xtilda1*xtilda1)+(1/16)*log(4*xtilda1+
        sqrt(1+16*xtilda1*xtilda1))-ttilda; // secant function at second penetration length guess
                                            for a given time
do
{
        dq=(q1-q0);     // change in secant function
        xtilda2=xtilda1-q1*((xtilda1-xtilda0)/dq);  // new non dimensional length guess
        xtilda0=xtilda1;
        q0=q1;
        xtilda1=xtilda2;
        q1=xtilda2/2+(xtilda2/4)*sqrt(1+16*xtilda2*xtilda2)+(1/16)*log(4*xtilda2+
                sqrt(1+16*xtilda2*xtilda2))-ttilda; // secant function at subsequent penetration
                                                    length guess for a given time
}while(fabs(q1)>.0001);
xtilda=xtilda2;  // non dimensional length at a given time
x=xtilda*xplus;          // length at a given time
z=1+16*(xtilda*xtilda);  // equivalence ratio constant
phix=(sqrt(z)-1)/2;      // equivalence ratio at tip at given time

// calculate time to reach length given in function prototype and equivalence ratio at that length
x1tilda=l1/xplus;
z=1+16*(x1tilda*x1tilda);          // equivalence ratio constant
phix1=2*AFs/(sqrt(z)-1); // equivalence ratio at given length
ttildax1=x1tilda/2+(x1tilda/4)*sqrt(1+16*x1tilda*x1tilda)+(1/16)*log(4*x1tilda+
        sqrt(1+16*x1tilda*x1tilda));      // non dimensional time to reach given length
tx1=ttildax1*tplus;      // time to reach given length
tetilda=te/tplus;        // non dimensional time for back end of spray

// calculate spray penetration length to the back end of the spray given the time after EOI included
        in the function prototype
xtilda0=50;
xtilda1=0;
q0=xtilda0/2+(xtilda0/4)*sqrt(1+16*xtilda0*xtilda0)+(1/16)*log(4*xtilda0+
        sqrt(1+16*xtilda0*xtilda0))-tetilda; // secant function at first penetration length guess for
                                            back end of spray
q1=xtilda1/2+(xtilda1/4)*sqrt(1+16*xtilda1*xtilda1)+(1/16)*log(4*xtilda1+
        sqrt(1+16*xtilda1*xtilda1))-tetilda; // secant function at second penetration length guess
                                            for back end of spray
do
{
        dq=(q1-q0);     // change in secant function
        xtilda2=xtilda1-q1*((xtilda1-xtilda0)/dq);  // new non dimensional length guess for
                                                    back end of spray
        xtilda0=xtilda1;
        q0=q1;
        xtilda1=xtilda2;
        q1=xtilda2/2+(xtilda2/4)*sqrt(1+16*xtilda2*xtilda2)+(1/16)*log(4*xtilda2+
                sqrt(1+16*xtilda2*xtilda2))-tetilda; // secant function at subsequent penetration
                                                    length guess for back end of spray
}while(fabs(q1)>.0001);
```

```
	xetilda=xtilda2;		// non dimensional length to back end of spray
	xe=xetilda*xplus;		// length to back end of spray
	ll[0]=L;		// variables returned to other functions
	ll[1]=phiL*fuel[7]/MWa;
	ll[2]=x;
	ll[3]=phix*fuel[7]/MWa;
	ll[4]=tL;
	ll[5]=spreadangle;
	ll[6]=tx1;
	ll[7]=phix1;
	ll[8]=xe;

	return;		// return to injection function
}
/*--------------------------------------------------------------------*/
/* injector parameters                                                */

void injector(int inj, double injdata[])
{
	/*injdata[0]=orifice diameter(um)  */
	/*injdata[1]=Cd                                                */
	/*injdata[2]=Ca                                                */

	switch (inj)
	{
		case 1:
			injdata[0]=100;
			injdata[1]=.80;
			injdata[2]=.86;
			break;
		case 2:
			injdata[0]=194;
			injdata[1]=.77;
			injdata[2]=.82;
			break;
		case 3:
			injdata[0]=251;
			injdata[1]=.79;
			injdata[2]=.79;
			break;
		case 4:
			injdata[0]=246;
			injdata[1]=.78;
			injdata[2]=.81;
			break;
		case 5:
			injdata[0]=267;
			injdata[1]=.77;
			injdata[2]=.82;
			break;
		case 6:
			injdata[0]=363;
			injdata[1]=.81;
			injdata[2]=.85;
			break;
		case 7:
```

```
                injdata[0]=498;
                injdata[1]=.84;
                injdata[2]=.88;
                break;
        case 8:
                injdata[0]=71;
                injdata[1]=.86;
                injdata[2]=.86;
                break;
        case 9:
                injdata[0]=180;
                injdata[1]=.77;
                injdata[2]=.82;
                break;
        default:
                break;
    }
}
```

APPENDIX B. PREMIXED BURN MODEL

This appendix describes how the heat release takes place from ignition until the lift-off length is reached. This has been termed the premixed burn duration.

The heat release rate from the initial premixed burn was calculated by spreading the total heat release from the available fuel at the time of ignition over a premixed burn time. To do this a premixed burn time must be determined as well as the shape of the heat release rate. A global reaction rate published by Turns [33] was used to determine the premixed burn time. The global reaction seen in Equation B.1 gives the change in concentration of diesel fuel, modeled as $C_{10}H_{22}$.

$$d[C_{10}H_{22}] = 3.8 \cdot 10^{11} \exp\left(\frac{15098}{T}\right)[C_{10}H_{22}]^{.25}[O_2]^{1.5}dt \qquad (B.1)$$

The fuel available for the reaction is all the fuel beyond the liquid length just prior to ignition or all the fuel in zone 2. The initial fuel and air concentrations were calculated using the equivalence ratio given in the spray model. A time step is taken and a change in fuel concentration is calculated. Equation B.1 is numerically integrated taking finite time steps ΔT, to determine successive fuel concentrations. The fuel concentration at the next time step is given by determining $\Delta[C_{10}H_{22}]$ from Equation B.2 and then the new

concentration of fuel from Equation B.3. Time steps are taken until the fuel concentration is zero as seen in Figure B.1.

$$\Delta[C_{10}H_{22}] = 3.8 \cdot 10^{11} \exp\left(\frac{15098}{T}\right)[C_{10}H_{22}]^{.25}[O_2]^{1.5}\Delta t \qquad \text{(B.2)}$$

$$[C_{10}H_{22}]_N = [C_{10}H_{22}]_O - d[C_{10}H_{22}] \qquad \text{(B.3)}$$

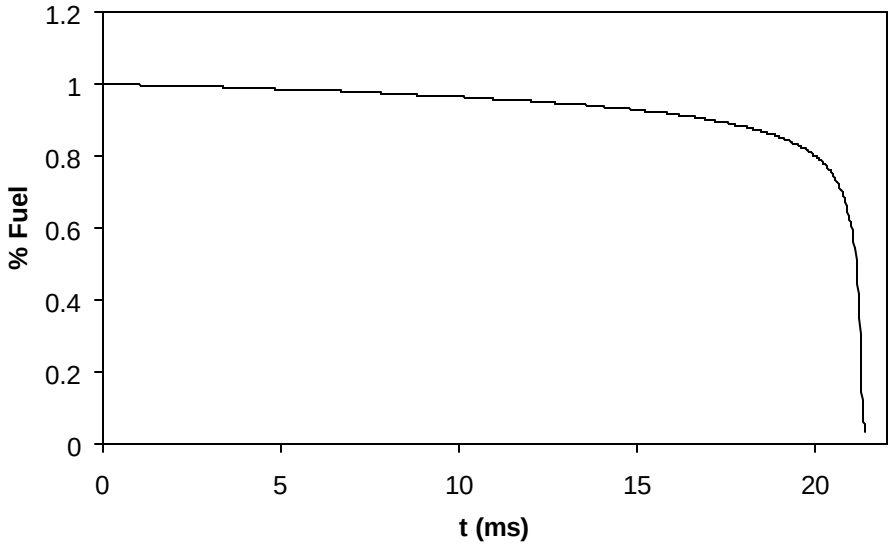


Figure B.1. Fuel Burn vs. Time Using Equation B.1.

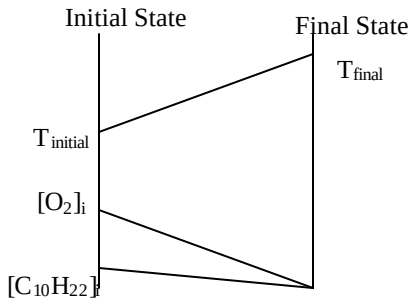


Figure B.2. Premixed Burn Concentrations and Temperature Variations

At each new time step the values of $[O_2]$ and temperature are changing. These are updated by recognizing that 1) the oxygen concentration reduces proportionally as the fuel concentration is reduced and 2) the temperature increases from the initial temperature to the adiabatic flame temperature as the fuel is burned. The temperature was assumed to increase linearly with decreasing fuel and O_2 concentration from the initial temperature in zone 2 to the adiabatic flame temperature. A schematic demonstrating the change in temperature and O_2 concentration as fuel concentration is changed is shown in Figure B.2 with the Equations B.4 and B.5 showing how the calculation in the program proceeded.

$$[O_2]_N = [C_{10}H_{22}]_N \left(\frac{[O_2]_2 - [O_2]_3}{[C_{10}H_{22}]_2 - [C_{10}H_{22}]_3} \right) \tag{B.4}$$

$$T_N = T_2 + ([O_2]_2 - [O_2]_N) \left(\frac{T_3 - T_2}{[O_2]_2 - [O_2]_3} \right) \tag{B.5}$$

The total time to consume the fuel for this reaction is much longer than a typical diesel premixed burn. The reaction rate starts and remains low for 19 - 20 ms before

increasing rapidly at the end as seen in Figure B.3. One reason for the longer reaction time is that the model being used assumes the products to be perfectly mixed after each time step. In reality, numerous hot spots would likely develop in the premixed mixture producing the higher temperature needed to increase the reaction rate locally. In order to scale the model to a realistic time, the "heat up" time was removed from the total reaction time. The heat up time was defined as the time required to produce a heat release greater than the enthalpy of vaporization of the fuel. The justification for selecting the heat of vaporization is that the ignition delay in an engine is typically defined as the time between injection and a positive value of apparent heat release. The apparent heat release can only be positive if the heat release from the fuel exceeds the heat absorbed by the evaporating fuel. When the energy from the reaction rate is greater than the energy required to vaporize the fuel, time accumulates until all of the fuel is consumed. The time between the end of the heat up period and the end of fuel concentration is the premixed burn duration t_{PM} as shown in Figure B.3.

Knowing the time for the premixed burn we must now determine the shape of the heat release during this time. The heat release from the initial premixed burn volume Q_{ID} is spread out over the premixed burn duration. In order to smooth the shape of this heat release the cosine function in Equation B.6 was used. In this equation t_{AI} is the time after ignition of the fuel and t_{PM} is the time for the premixed burn. This function multiplies Q_{ID} by a number between 0 and 1 calculated using the cosine function to determine how

much of the premixed heat release to add in a particular time step. This amount is used to calculate the heat release rate for those crank angles.

$$Q_{PM} = Q_{ID}\left(1-\left(.5\cos\left(p\frac{t_{AI}}{t_{PM}}\right)+.5\right)\right) \tag{B.6}$$

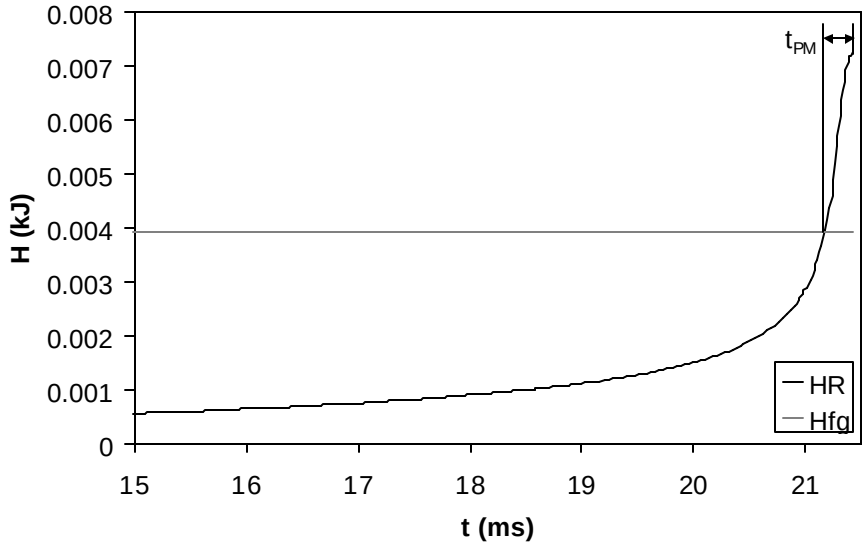


Figure B.3 Time for the Premixed Burn.

Knowing the time for the premixed burn we must now determine the shape of the heat release during this time. The heat release from the initial premixed burn volume Q_{ID} is spread out over the premixed burn duration. In order to smooth the shape of this heat release the cosine function in Equation B.6 was used. In this equation t_{AI} is the time

after ignition of the fuel and t_{PM} is the time for the premixed burn. This function multiplies Q_{ID} by a number between 0 and 1 calculated using the cosine function to determine how much of the premixed heat release to add in a particular time step. This amount is used to calculate the heat release rate for those crank angles.

$$Q_{PM} = Q_{ID}\left(1-\left(.5\cos\left(p\frac{t_{AI}}{t_{PM}}\right)+.5\right)\right) \tag{B.6}$$

The new total heat release in region A described in Chapter 4 at any crank angle including the premixed burn $Q_{A,N}$ is shown in Equation B.7. To account for the premixed burn the total heat release available to the premixed burn Q_{ID} is subtracted from the total available at the current crank angle $Q_{A,O}$ and a portion for the premixed burn calculated in Equation B.6 Q_{PM} is added back in as shown in Figure B.4.

$$Q_{A,N} = Q_{A,O} - Q_{ID} + Q_{PM} \tag{B.7}$$

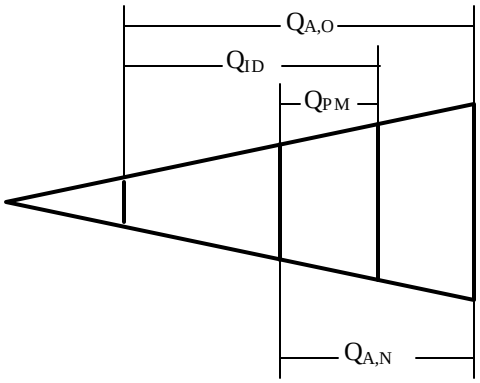


Figure B.4. Premixed Burn Heat Release

The heat release rate then becomes Equation B.8 This equation takes into account the vaporization of the fuel and the premixed burn duration. It also takes into account the ignition delay by setting $Q_{Tot,t+?t}$ and $Q_{Tot,t}$ to zero until the start of combustion.

$$\dot{Q}_{HR} = Q_{Tot,t+\Delta t,N} - Q_{Tot,t} \tag{B.8}$$

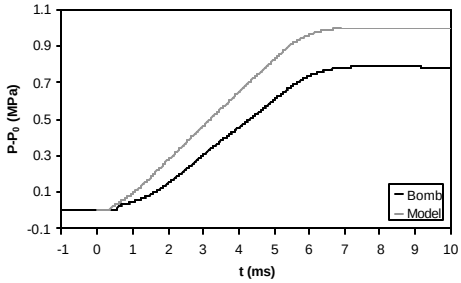
APPENDIX C. DATA SETS

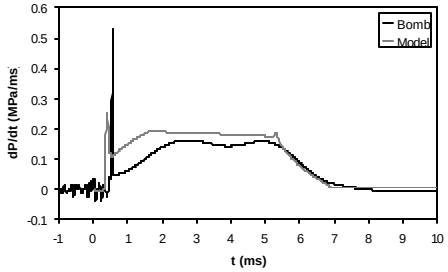
Sandia National Laboratories Constant Volume Combustion Vessel Results

Pressure Rise

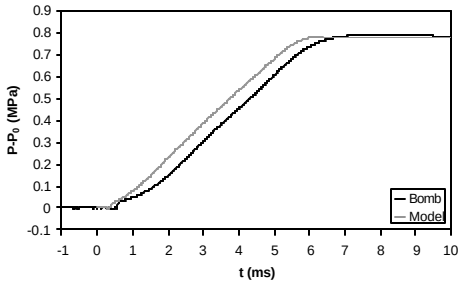
Heat Release Rate

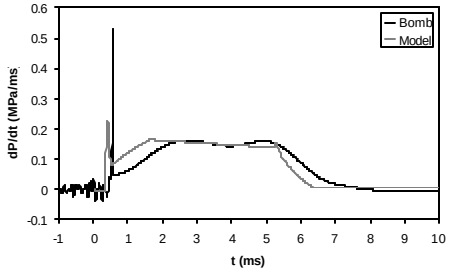
246 μm nozzle





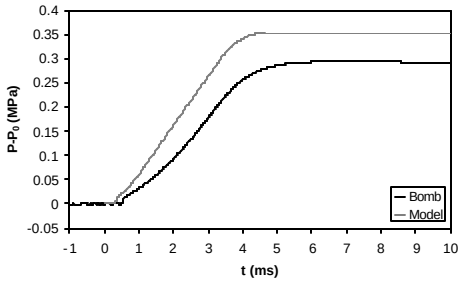
No heat transfer

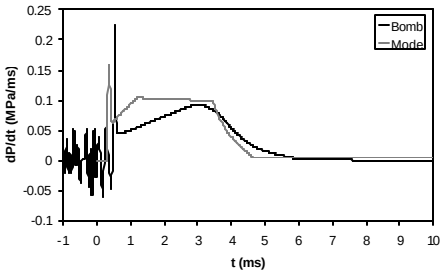




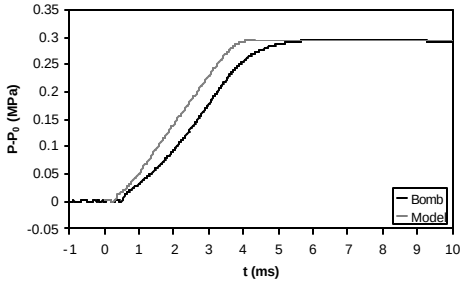
Heat transfer

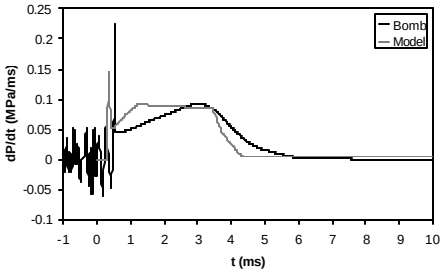
180 µm nozzle





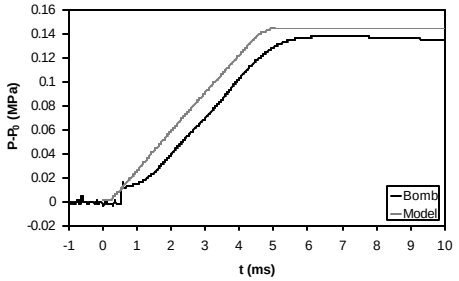
No heat transfer

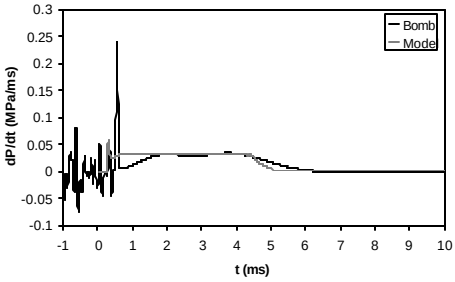




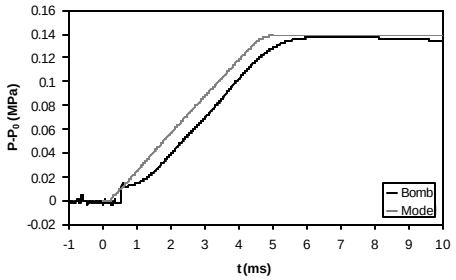
Heat transfer

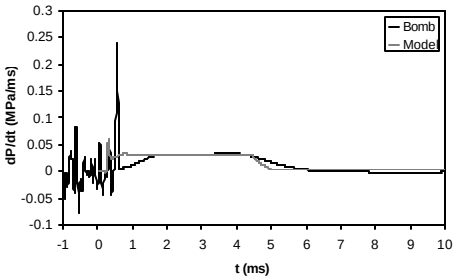
100 µm nozzle





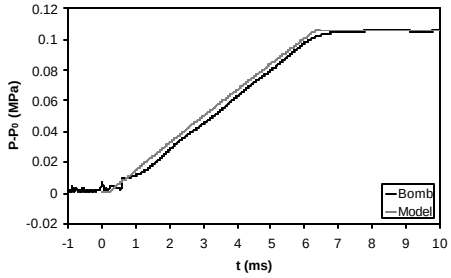
No heat transfer

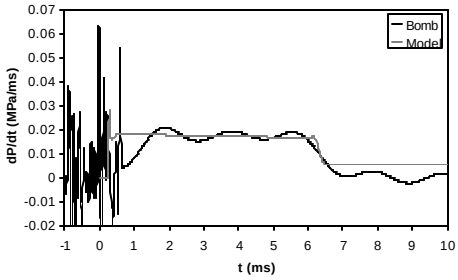




Heat transfer

71 µm nozzle



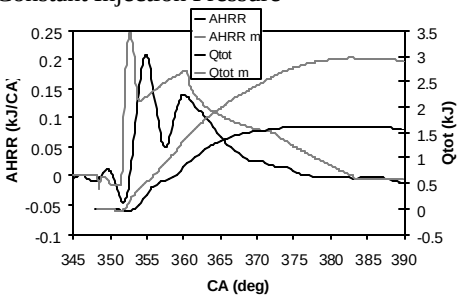


No heat transfer

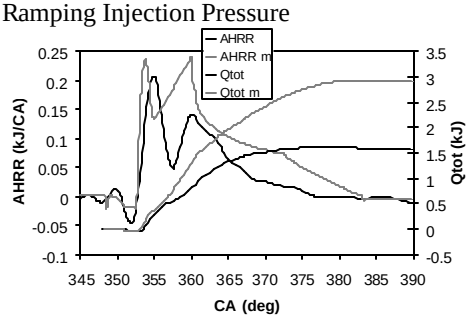
Sandia National Laboratories Research Engine Heat Release Rate Results

1200 RPM Low Fuel Load Normal Injection Timing

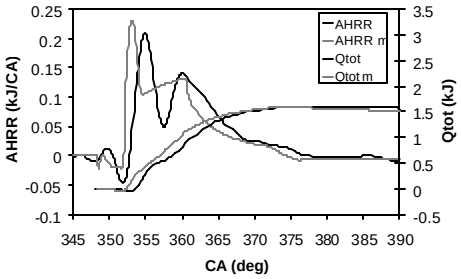
Constant Injection Pressure

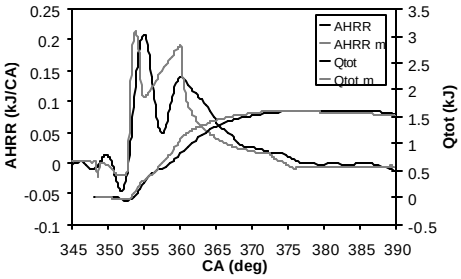


Ramping Injection Pressure



Normal Heat Transfer

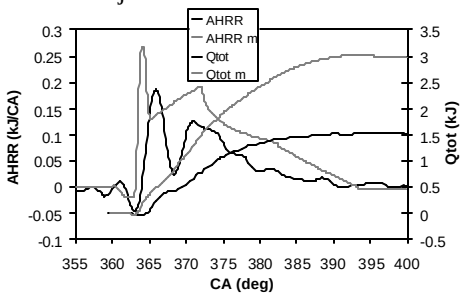




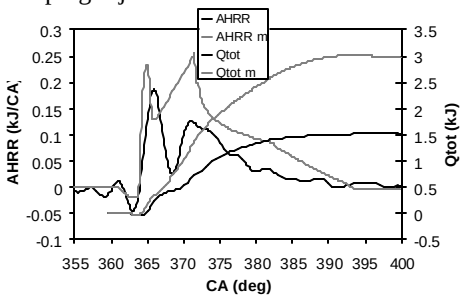
Extra Heat Transfer

1200 RPM Low Fuel Load Retarded Injection Timing

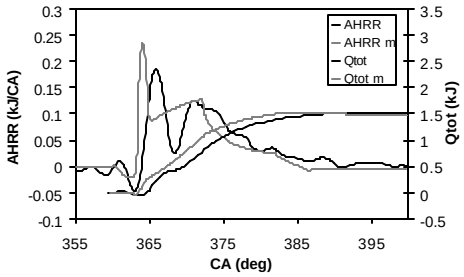
Constant Injection Pressure

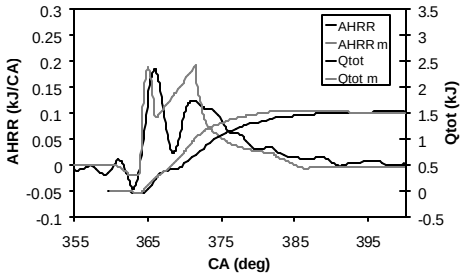


Ramping Injection Pressure



Normal Heat Transfer

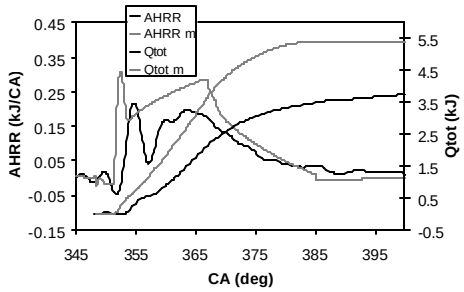




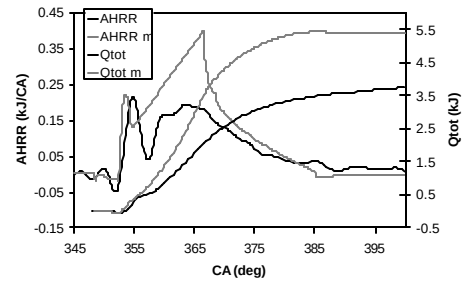
Extra Heat Transfer

1200 RPM High Fuel Load Normal Injection Timing

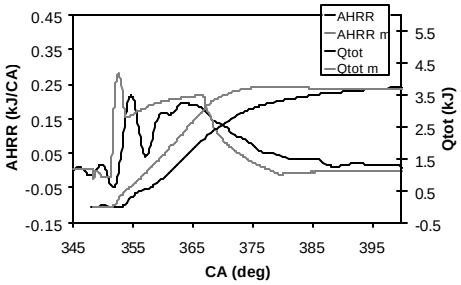
Constant Injection Pressure

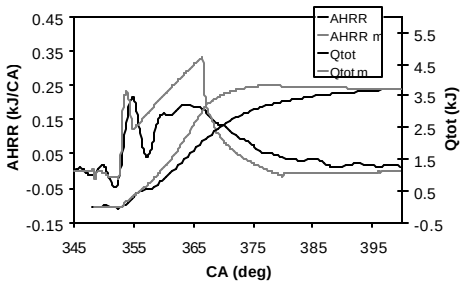


Ramping Injection Pressure



Normal Heat Transfer

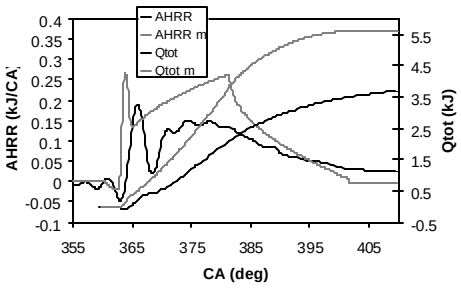


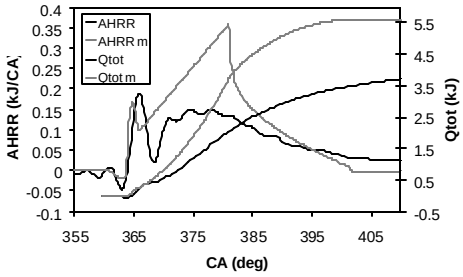


Extra Heat Transfer

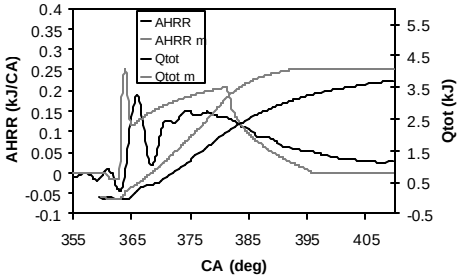
1200 RPM High Fuel Load Retarded Injection Timing

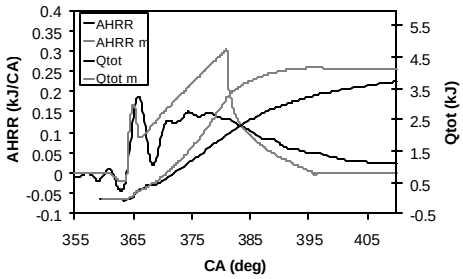
Constant Injection Pressure Ramping Injection Pressure





Normal Heat Transfer

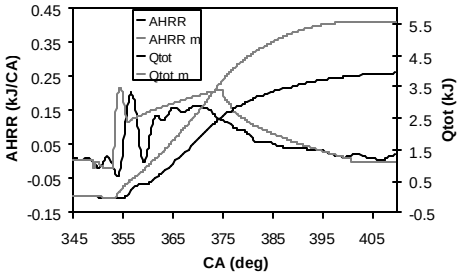




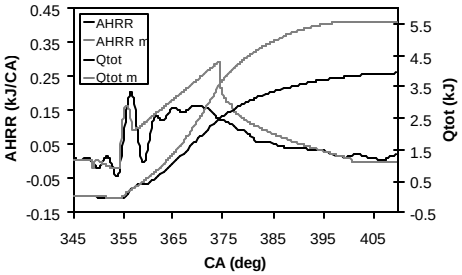
Extra Heat Transfer

1680 RPM High Fuel Load Normal Injection Timing

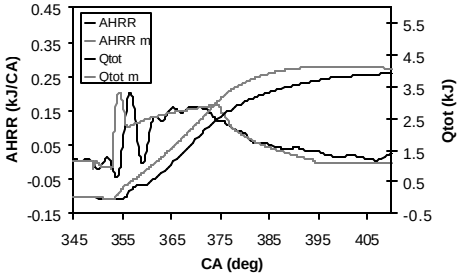
Constant Injection Pressure

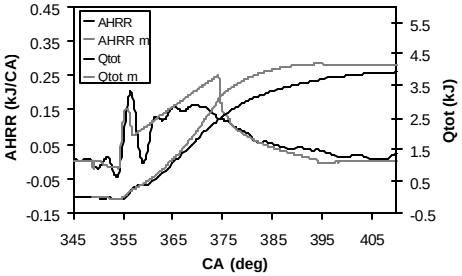


Ramping Injection Pressure



Normal Heat Transfer

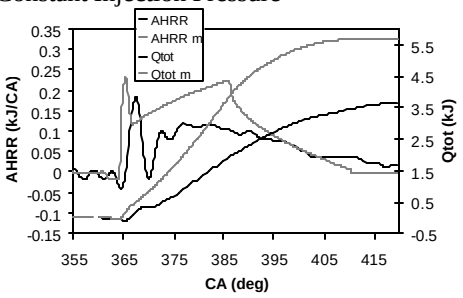




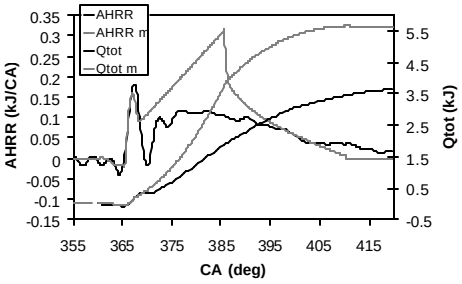
Extra Heat Transfer

1680 RPM High Fuel Load Retarded Injection Timing

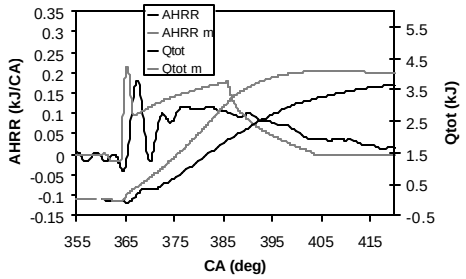
Constant Injection Pressure

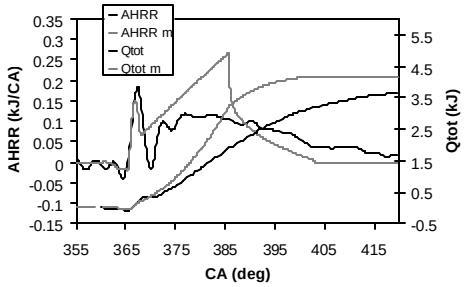


Ramping Injection Pressure



Normal Heat Transfer



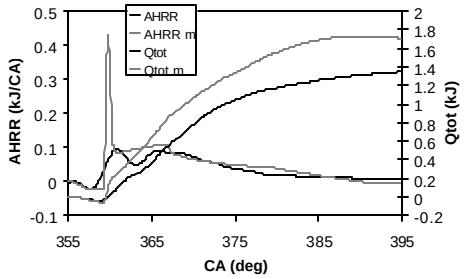


Extra Heat Transfer

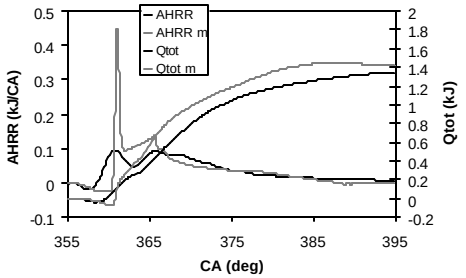
BYU Research Engine Heat Release Rate Results

1500 RPM $\phi = .3$

Constant Injection Pressure

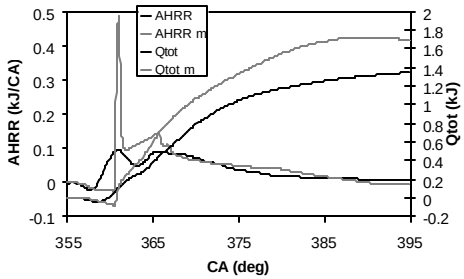


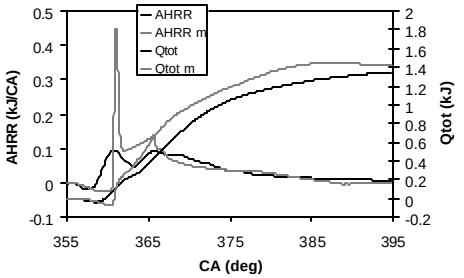
Normal Heat Transfer



Extra Heat Transfer

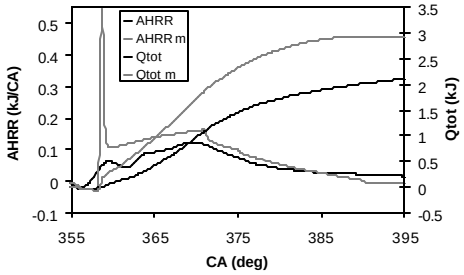
Ramping Injection Pressure



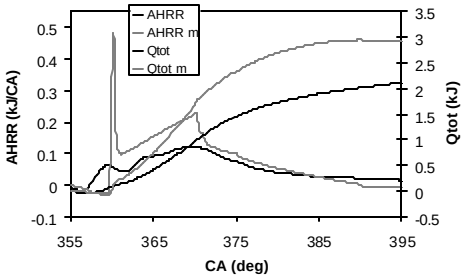


1500 RPM ϕ = .5

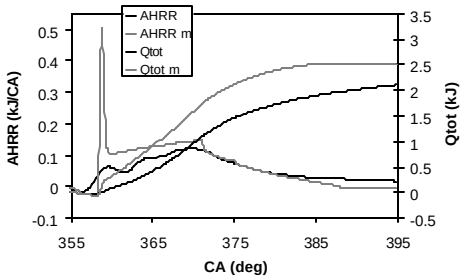
Constant Injection Pressure

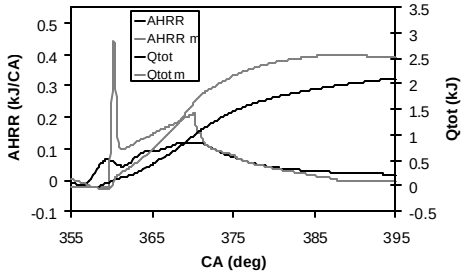


Ramping Injection Pressure



Normal Heat Transfer

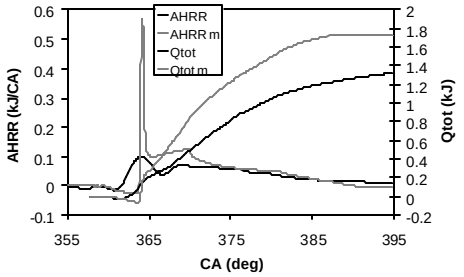




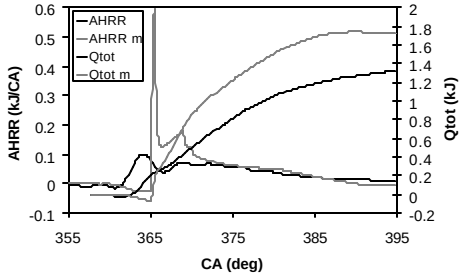
Extra Heat Transfer

2000 RPM ϕ = .3

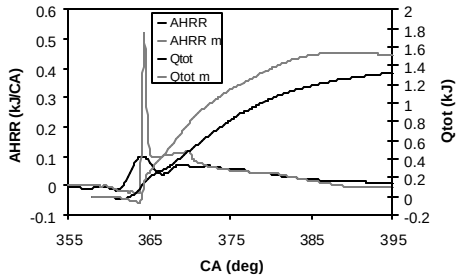
Constant Injection Pressure

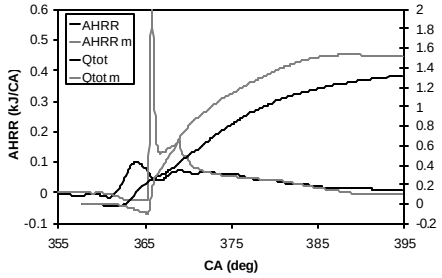


Ramping Injection Pressure



Normal Heat Transfer

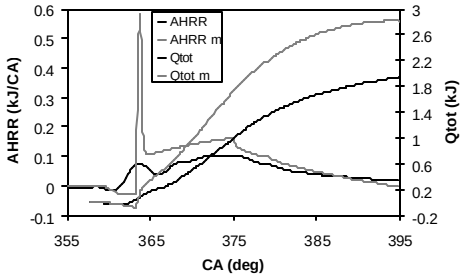




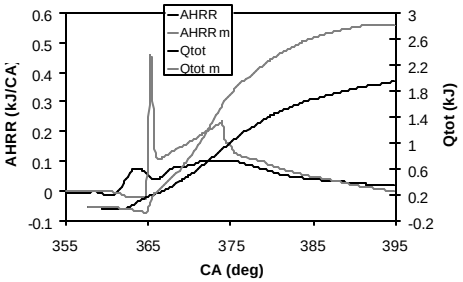
Extra Heat Transfer

2000 RPM $\phi = .5$

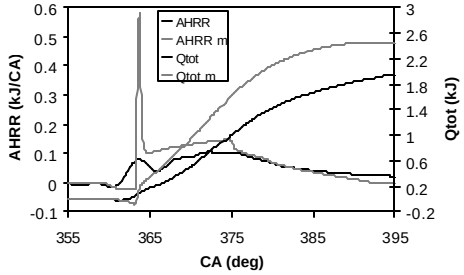
Constant Injection Pressure

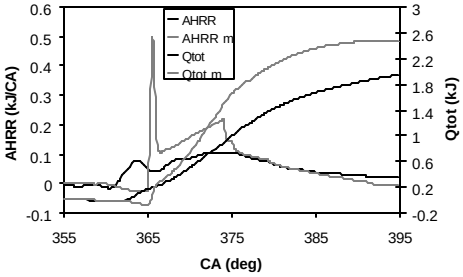


Ramping Injection Pressure



Normal Heat Transfer





Extra Heat Transfer