To the Graduate Council:

I am submitting herewith a thesis written by David Christopher Thames entitled "A Computational Model of Meteor Entry Into Earth's Atmosphere." I have examined the final copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Aerospace Engineering.

Dr. J. Evans Lyne, Major Professor

We have read this thesis and
recommend its acceptance:

Dr. H. Joe Wilkerson

Dr. Charles R. Collins

Accepted for the Council:

Associate Vice Chancellor and
Dean of The Graduate School

A COMPUTATIONAL MODEL OF METEOR ENTRY INTO EARTH'S ATMOSPHERE

A Thesis
Presented for the
Master of Science
Degree
The University of Tennessee, Knoxville

David Christopher Thames

August 1997

DEDICATION

I dedicate this thesis to the memory of my cousin and dear friend

Edward Kyle Ruhl

who inspired me to imagine going places I never dreamt possible.

ACKNOWLEDGMENTS

I wish to thank my advisor, Dr. Evans Lyne, for his guidance and for giving me the opportunity to work in an area that is almost as much fun as it is work. I owe a debt of gratitude to K. G. Sewell and the 1961 group at Chance Vought for producing the document that was the basis for my research. I also want to thank the members of my graduate committee, Dr. Joe Wilkerson and Dr. Charles Collins, for their assistance and encouragement. Their suggestions and ideas were invaluable. I am indebted to the Mechanical and Aerospace office staff in the department, especially Marva Anderson, for always having the answer to my questions, no matter how obscure.

There are many other people whom I wish to thank for their patience and assistance in helping me get through this hectic time. First and foremost, I wish to thank my wife, Susan, for her patience and willingness to put her dreams aside while I have pursued two degrees, for her unfaltering support of my dreams, for the tireless hours that she read through revision after revision of this thesis, and for being there when things got rough. I wish to thank my parents for encouraging me to go after my dreams and helping me to accomplish them. I also want to thank my mother- and father-in-law for their support and encouragement. I owe thanks to the rest of my family and to all of my in-laws for their constant encouragement.

Finally, I include a thank you to the many people whom I should have included in this short list, but somehow overlooked.

ABSTRACT

This work improves an existing model by Evans Lyne, Michael Tauber, and Rich Fought of meteor entry through Earth's atmosphere to allow calculations that are more accurate for a broader range of entry bodies than previously possible. An extensive high temperature thermodynamic model for air is used. This model was devised by a group of researchers at the Chance Vought Research Center and provides more than a threefold increase in the allowable temperature range over current comparable thermodynamic models. This corresponds to more than a doubling of the allowable entry velocity of meteors into Earth's atmosphere from approximately 20 km/s to 40 km/s and above. This program includes a detailed fragmentation model and an ablation model that accounts for both cooling and blowing effects within the shock layer. The model allows for analysis of nearly all Earth-crossing asteroids and virtually all non-retrograde comets. Other minor improvements have also been made over the previous model, including an improved pressure and enthalpy calculation behind the shock and an improved method of using the Green-Nicolet approximation to account for the blowing of ablated material blocking some of the radiative heat transfer to the body.

TABLE OF CONTENTS

Chapter 1	Introduction	1
1.1	Historical Background	1
1.2	Meteor Trajectory Models	4
1.3	Meteor Fragmentation Models	4
1.4	Meteor Ablation Models	6
1.5	Applicable Research	9
Chapter 2	Original Trajectory Model	10
2.1	Model Considerations	10
2.2	Entry Aerodynamics	11
2.3	Body Mechanics	12
2.4	Aerodynamic Heating	14
Chapter 3	Modified Trajectory Model	18
3.1	Objectives	18
3.2	Chance Vought Research	19
3.3	Curve Fits Using ITPLBV	22
3.4	Curve Fitting Chance Vought with ITPLBV	25
3.5	Utilizing the Curve Fits in the METEOR Program	28
3.6	Solving Differential Equations with DVERK	29
3.7	Green-Nicolet Adjustment	30
3.8	FORTRAN 77 to Fortran 90	32
Chapter 4	Error Analysis	34
4.1	Thermodynamic Curves: Vought vs. Tannehill	34

4.2	Thermodynamic Curves: Vought vs. Fought	38
4.3	Trajectory Comparisons	42
4.4	Green-Nicolet Comparison	48
Chapter 5	Trajectory Analysis	50
5.1	Analysis Approach	50
5.2	Trajectory Analysis: Entry Parameters	52
5.3	Trajectory Analysis: Body Parameters	70
5.4	Trajectory Analysis: Body Types	89
5.5	Airburst Altitude	96
5.6	Radiative Heat Flux Across the Face	105
Chapter 6	Conclusion	108
6.1	Current Status	108
6.2	Future Work	108
References		110
Appendices		115
A.	Chance Vought Graphs	116
B.	Chance Vought Data Tables	124
C.	METEOR Code (Fortran 90)	128
Vita		207

LIST OF TABLES

4.1	Enthalpy (pressure, density) comparison	38
4.2	Temperature (pressure, density) comparison	38
4.3	Temperature (pressure, enthalpy) comparison	39
4.4	Fought's physical characteristics of entry bodies	47
5.1	Body Physical Parameters	51
5.2	Entry Parameters	52

LIST OF FIGURES

3.1	ITPLBV derivative grid	23
3.2	Green-Nicolet Approximation	31
3.3	Modified Green-Nicolet Approximation	31
4.1	Vought vs. Tannehill temperature (pressure, density)	36
4.2	Vought vs. Tannehill enthalpy (pressure, density)	37
4.3	Vought vs. Fought temperature (pressure, enthalpy)	40
4.4	Vought vs. Fought low pressure thermodynamic curves	41
4.5	METEOR vs. T10 time (Comet 50 m diam.)	43
4.6	METEOR vs. T10 velocity (Comet 50 m diam.)	43
4.7	METEOR vs. T10 stagnation point temperature (Comet 50 m diam.)	44
4.8	METEOR vs. T10 stagnation pressure (Comet 50 m diam.)	44
4.9	METEOR vs. T10 body mass (Comet 50 m diam.)	45
4.10	METEOR vs. T10 kinetic energy (15 km/s, 50 m diam.)	46
4.11	METEOR vs. T10 body radius (15 km/s, 50 m diam.)	46
4.12	Green-Nicolet Approximation (Comet 15 km/s, 50 m diam.)	49
5.1	Altitude vs. Time (Weak median body, 10 m radius, 45°)	53
5.2	Altitude vs. Time (Weak median body, 10 m radius, 25 km/s)	54
5.3	Trajectory Velocity (Weak median body, 45°)	56
5.4	Trajectory Velocity (Weak median body, 10 m radius, 25 km/s)	57
5.5	Trajectory Kinetic Energy (Weak median body, 45°)	58
5.6	Trajectory Kinetic Energy (Weak median body, 10 m radius, 25 km/s)	59
5.7	Trajectory Temperature (Weak median body, 45°)	61

5.8	Trajectory Temperature (Weak median body, 10 m radius, 25 km/s)	62
5.9	Trajectory Pressure (Weak median body, 45°)	63
5.10	Trajectory Pressure (Weak median body, 10 m radius, 25 km/s)	64
5.11	Trajectory Mass (Weak median body, 45°)	66
5.12	Trajectory Mass (Weak median body, 10 m radius, 25 km/s)	67
5.13	Trajectory Radius (Medium median body, 45°)	68
5.14	Trajectory Radius (Weak median body, 10 m radius, 25 km/s)	69
5.15	Trajectory Velocity vs. Density (Weak median body, 50 m radius, 35 km/s, 45°)	71
5.16	Trajectory Kinetic Energy vs. Density (Weak median body, 50 m radius, 35 km/s, 45°)	72
5.17	Trajectory Temperature vs. Density (Weak median body, 50 m radius, 35 km/s, 45°)	73
5.18	Trajectory Pressure vs. Density (Weak median body, 50 m radius, 35 km/s, 45°)	74
5.19	Trajectory Mass vs. Density (Weak median body, 50 m radius, 35 km/s, 45°)	75
5.20	Trajectory Radius vs. Density (Weak median body, 50 m radius, 35 km/s, 45°)	76
5.21	Trajectory Heat Flux vs. Density (Weak median body, 50 m radius, 35 km/s, 45°)	77
5.22	Trajectory Mass vs. Heat of Ablation (Weak median body, 50 m radius 35 km/s, 45°)	79
5.23	Trajectory Kinetic Energy vs. Heat of Ablation (Weak median body, 50 m radius, 35 km/s, 45°)	80
5.24	Trajectory Velocity vs. Heat of Ablation (Weak median body, 50 m radius, 35 km/s, 45°)	81
5.25	Trajectory Temperature vs. Heat of Ablation (Weak median body, 50 m radius, 35 km/s, 45°)	82

5.26	Trajectory Pressure vs. Heat of Ablation (Weak median body, 50 m radius, 35 km/s, 45°)	83
5.27	Trajectory Heat Flux vs. Heat of Ablation (Weak median body, 50 m radius, 35 km/s, 45°)	84
5.28	Trajectory Radius vs. Yield Strength (Weak median body, 50 m radius, 35 km/s, 45°)	85
5.29	Trajectory Mass vs. Yield Strength (Weak median body, 50 m radius, 35 km/s, 45°)	86
5.30	Trajectory Pressure vs. Yield Strength (Weak median body, 50 m radius, 35 km/s, 45°)	87
5.31	Trajectory Heat Flux vs. Yield Strength (Weak median body, 50 m radius, 35 km/s, 45°)	88
5.32	Trajectory Velocity (Median body, 50 m radius, 45°)	90
5.33	Trajectory Kinetic Energy (Median body, 50 m radius, 45°)	92
5.34	Trajectory Temperature (Median body, 50 m radius, 45°)	93
5.35	Trajectory Pressure (Median body, 50 m radius, 45°)	94
5.36	Trajectory Mass (Median body, 50 m radius, 45°)	95
5.37	Trajectory Radius (Median body, 50 m radius, 45°)	97
5.38	Trajectory Heat Flux (Median body, 50 m radius, 45°)	98
5.39	Airburst altitude for various entry parameters (Weak median body)	100
5.40	Airburst altitude for various body types (Median body, 15°)	101
5.41	Airburst Height vs. Density (10 m radius, 35 km/s, 15°)	102
5.42	Airburst Height vs. Heat of Ablation (10 m radius, 35 km/s, 15°)	103
5.43	Airburst Height vs. Yield Strength (10 m radius, 35 km/s, 15°)	104
5.44	Temperature vs. Body Position (Weak median body, 50 m radius, 35 km/s, 45°)	106

5.45 Heat Flux vs. Body Position (Weak median body, 25 m radius, 30 km/s, 45°) 107

NOMENCLATURE

A	cross-sectional area
C_d	drag coefficient
C_H	heat transfer coefficient
dt	time step
g	gravitational acceleration
H	altitude
h_1	enthalpy outside of shock layer
h_2	enthalpy behind shockwave
m	mass
$\dot{m}$	mass flow rate
P_1	pressure outside of shock layer
P_2	pressure behind shockwave
P_{stag}	stagnation pressure
Q	heat of ablation
q_R	radiative heat transfer rate
q_{Rad}	adiabatic radiative heat transfer rate
R	distance from center of the Earth
r	effective radius of cross-section
S_{GN}	Green-Nicolet scale factor
T	temperature
T_1	temperature outside of shock layer

T_2	temperature behind shockwave
t	time
V	velocity
β	local shock angle
δ	shock layer thickness
ε	ablation parameter
μ	average effective absorption coefficient per unit length
γ	flight path angle
ρ_a	local atmospheric density
ρ_m	meteor density
ρ_1	density outside of shock layer
ρ_2	density behind shockwave
σ	Stefan-Boltzmann constant (5.670×10^{-8} W/m^2·K^4)

CHAPTER 1

INTRODUCTION

1.1 Historical Background

In 1980, Luis and Walter Alvarez proposed the possibility that 65 million years ago dinosaurs were completely annihilated by the aftermath of a massive meteor's collision with the Earth. Many scientists ridiculed the concept at the time. However, in 1989 the 150 to 180 mile wide crater of the Alvarez's meteor was found in the southern Gulf of Mexico, and in 1996 it was confirmed to be the right age and composition. While there is still debate over whether the meteor was solely responsible for the extinction of the dinosaurs, there is little argument over the significance of the meteor impact and its aftermath. It was a body large enough to cause waves that crossed over the Florida peninsula and to spread particles from the meteor in the atmosphere around the entire globe (Seeds 1990).

On June 30, 1908 in the far reaches of Siberia, local villagers witnessed a ball of fire crossing the sky and then heard and felt a blast that threw people to the ground and completely leveled about 2200 km of the Tunguska forest. Seismic waves were detected to have circled the globe twice, and a mysterious glow could be seen in the skies across Europe and Siberia over the next several days. Because of turmoil in the region and the isolation of Siberia, it was several years before anyone investigated the incident, and they could find no cause; only a huge circular area of fallen and charred

trees similar to what would be expected from a nuclear explosion above the surface of the Earth (Stone 1996). There was speculation that the cause was a spaceship that exploded trying to land, a floating miniature black hole, or a piece of antimatter that collided with matter. It is more commonly believed to have been a meteor that exploded several kilometers above the ground. Since a meteor is any comet or asteroid that enters Earth's atmosphere, the only serious debate is over which type of body the meteor was and how large it was when it entered Earth's atmosphere (Seeds 1990).

In 1994, the scientific community was treated to a front row seat of the comet Shoemaker-Levy 9 colliding with Jupiter. This event also sparked the general public's interest in meteors. Articles appeared in virtually every major and minor news source and science magazine. A coordinated worldwide observational effort using both ground and space-based instruments amassed a wealth of data.

Then in 1996, a group from NASA announced the discovery of possible micro-fossils on a meteor from Mars that suggest microscopic life could have existed on the red planet or even that life on Earth could have evolved from these alien components.

Over the last forty years, these incidents have led to a growing scientific and public interest in meteors and their effects on the Earth. The past year has seen numerous articles about meteors in major science and news magazines, documentaries about what has happened and what could happen, and a two part mini-series on network television about a massive asteroid on a collision course with Earth.

More importantly, the growing interest has led to more serious research about the possibilities. NASA has funded two studies that both concluded the only defense against a catastrophic collision would be early warning. This would allow time to consider feasible alternatives and to consider the possibility of changing the trajectory of large bodies. They recommended that funding should be put forth to catalogue the estimated 2000 comets and asteroids of one kilometer or larger size that currently cross Earth's orbit. Currently, NASA is funding three very small groups that are attempting to catalogue as many objects as possible. Additionally, many studies have been done on the local and global consequences of certain size and mass bodies impacting different areas of the globe.

While knowing which bodies could collide with Earth and what the results would be when the body impacts are important, it is only useful if the dynamics of these bodies entering the atmosphere are understood. The Tunguska meteor would have had devastating results over a much smaller area if the meteor had hit the ground rather than exploding above it. It could have killed hundreds or thousands if it had landed in a populated area. Therefore, an understanding of the physical processes which the meteor undergoes as it passes through Earth's atmosphere is necessary to determine the trajectory of the body, whether the meteor will strike the surface, and how large the body will be at impact.

1.2 Meteor Trajectory Models

Over the last decade and a half, a large amount of effort has been put forth by researchers and scientists to model the entry of meteors into Earth's atmosphere. This analysis requires a detailed calculation of the body's kinetic energy including losses due to aerodynamic drag and due to mass loss as a result of ablation. Additional consideration must be given to the pressure build up on the windward face of the body. These pressures can lead to stresses greater than the meteor's yield strength and cause fragmentation or possibly even an explosion, or airburst, of the meteor. Very early models were crude approximations and often neglected significant factors.

Recent work by many researchers including Chyba *et al.* (1993), Hills and Goda (1993), Svetsov *et al.* (1995), and Thomas *et al.* (1994) have offered many variations and improvements on different aspects of meteor entry modeling. However, all of these models seem to be lacking in some area of the analysis, either with regard to the mechanical fragmentation of the body or the aerothermal ablation. The approaches by different researchers and the current status of the fragmentation and ablation models are described in the following sections.

1.3 Meteor Fragmentation Models

Prior to 1993, several models of meteor entry neglected fragmentation. Models by Fay *et al.* (1964) and Biberman *et al.* (1972, 1978a, 1978b, 1980) limited their research to small bodies with ablation only and included no fragmentation model. A

model by Baldwin and Shaeffer (1971) assumed that once the yield strength was reached, the body fractured into a specific number of individual fragments. Research published by Chyba *et al.* (1993) and Hills and Goda (1993) suggested a simple analytical model. These models proposed that the body fractured when the windward face pressure reaches twice the yield strength, at which time the pressure in the center of the body should be approximately equal to the yield strength. The individual fragments continue to move as a single deformable body that begins to pancake and spread radially. Chyba derived equation [1.1] to calculate the radius of a fragmented body as it grows.

$$r\frac{d^2r}{dt^2} = \frac{C_d\rho_a V^2}{2\rho_m}, \qquad [1.1]$$

where r is body radius, C_d is the drag coefficient, ρ_a is the atmospheric density, V is the velocity of the body, and ρ_m is the density of the meteor. This model was later used by Lyne *et al.* (1996, 1997).

Later models by Thomas *et al.* (1994) and Svetsov *et al.* (1995) using complex, numerical hydrodynamic models have compared reasonably well with Chyba's simple analytical model (Chyba 1993). The fact that calculations using vastly different approaches have yielded similar results lends credibility to the findings of these various models of mechanical failure.

1.4 Meteor Ablation Models

While the fragmentation models have generally produced similar results, recent ablation models show substantial disagreement. Svetsov *et al.* (1995) concluded that "for large bodies above several meters in size...we can consider only mechanical processes." To account for ablation, Hills and Goda used an equation from Bronshten (1983)

$$\frac{dm}{dt} = -\frac{1}{2}\varepsilon A \rho_a V^3, \qquad [1.2]$$

where m is the mass of the body, ε is the ablation parameter, and A is the cross-sectional area of the windward face. This model and that by Sekanina (1983) assumed a constant coefficient of heat transfer throughout the trajectory of the body. Both Baldwin and Shaeffer (1971) and Chyba *et al.* (1993) use a similar approach defined by the equation

$$Q\frac{dm}{dt} = -\frac{1}{2}AC_H\rho_a V^3, \qquad [1.3]$$

where Q is the heat of ablation and C_H is the coefficient of heat transfer. Equation [1.3] has split the ablation parameter from equation [1.2] into a constant heat of ablation and a variable coefficient of heat transfer. Chyba's model (Chyba 1993) calculates the heat transfer using the well-known Stefan-Boltzmann equation

$$Q\frac{dm}{dt} = -\sigma A T^4, \qquad [1.4]$$

where σ is the Stefan-Boltzmann constant and T is the temperature behind the shockwave. For Chyba's model, this temperature behind the shockwave is assumed to be uniform and constant at 25,000 K. These various models have resulted in drastically different results. While the model of Svetsov *et al.* (1995) completely neglects the ablation, the models of Chyba *et al.* (1993) and Sekanina *et al.* (1983) have been shown to substantially over predict ablation (Lyne 1997). An example of the enormous differences that can be produced by the various models can be seen by examining an icy body (ρ = 1 g/cm^3) entering the atmosphere vertically. If the body has a 100 meter radius and enters the atmosphere at 50 km/s, "the body loses half its initial kinetic energy at an altitude about 4 km," according to analysis by Svetsov *et al.* (1995). However, the model by Hills and Goda estimates that a body of this type would lose half of its energy before it even reached 20 km. This would make a huge difference considering that the maximum energy release occurs when the body reaches approximately fifty-percent energy loss.

A major source of error in several previous ablation models has been their failure to account for non-adiabatic processes and radiation blockage within the shock layer. Notable exceptions have been the studies by Fay *et al.* (1964) and Biberman *et al.* (1972, 1978a, 1978b, 1980). However, as was previously mentioned, these works considered only small, non-fragmenting bolides.

A recent model by Lyne *et al.* (1997) has attempted to integrate both a fragmentation model and a relatively sophisticated aerothermal model which accounts

for both radiative cooling (non-adiabatic effects) and radiative blockage within the shock layer. This model calculates the thermodynamic state of the gas in the shock layer as a function of both time and space during the meteor's trajectory and uses this information to determine the heating distribution over the meteor's surface. A detailed description of the approach is given in the following chapter.

A major limitation to the method of Lyne *et al.* (1997) has been the availability of thermodynamic data for high temperature air. Two computational models are widely available to determine the thermodynamic state within the shock layer. The first was originally written by Harry Bailey (1967) and has been improved upon by Tannehill *et al.* (1974), and more recently by Srinivasan *et al.* (1987). This current version of this model is valid within a range of temperatures from 300 K to 25,000 K and within a pressure range of 10^{-7} atm to 1,000 atm. The second thermodynamic model was originally written by Gupta (1987) and has continued to be improved upon with two later versions by Gupta *et al.* (1990, 1991). However, the 1991 version of this model was restricted to a temperature range of 500 K to 30,000 K and a pressure range of 10^{-4} atm to 100 atm. Further work with this model by Fought *et al.* (1997) has extended the maximum temperature from 30,000 K to 32,600 K and the maximum pressure from 100 atm to 1,000 atm. This gives a model that matches the upper pressure limit of the Srinivasan model and exceeds the maximum temperature by over 7,000 K. However, even this improved model is restrictive on meteor study. This temperature limit of Fought *et al.* (1997) still limits the maximum speed of meteors analyzed to approximately 20 km/s. This is due to the temperatures reached inside

the shock layer of meteors entering the Earth's atmosphere. According to the data regarding the known Earth orbit crossing asteroids (Gehrels 1994) and studies of existing Earth and lunar impacts (Chyba 1991), approximately 20% of all Earth-crossing asteroids and about 70% of all Earth-crossing cometary bodies travel at speeds greater than 20 km/s.

1.5 Applicable Research

During the 1960's, a group at the Chance Vought Research Center anticipated the need for high temperature and high pressure data with the growing interest in hypersonic flight, the increase in aircraft speeds, and the beginning of the age of space flight. In 1961, this group completed a report calculating the properties of air at temperatures from 3,000 K to 100,000 K for pressures from 0.00001 to 100,000 atm. However, this report was apparently never published outside of the company and it virtually disappeared, except for a reference in an early AIAA paper (Fay 1964) attempting to analyze some aspects of meteor entry.

The present research makes use of the results of the Chance Vought study to improve upon the current meteor trajectory model of Lyne *et al.* (1997). The atmospheric entry model of Lyne *et al.* (1997) is explained in the following chapter and the changes made to that model for the purposes of this research are explained in Chapter 3. The results of these modifications are then discussed in Chapters 4 and 5.

CHAPTER 2

ORIGINAL TRAJECTORY MODEL

2.1 Model Considerations

This chapter is a summary of the original code that was developed by Evans Lyne and Rich Fought to model meteor atmospheric entry. There are three main areas of consideration during entry. These are aerodynamic forces, aerodynamic heating, and body mechanics. The first of these, aerodynamic forces, involves the atmospheric drag slowing the body as gravity pulls the object towards the surface. As the body moves lower into the atmosphere, the density increases and, therefore, the amount of atmospheric drag increases. The aerodynamic heating involves heating of the air as it crosses the hypersonic shockwave. This hot air then radiates heat to the body of the meteor resulting in a gradual ablation of the surface. The third consideration, body mechanics, involves the pressure buildup on the windward side of the meteor as it passes through increasingly dense layers of the atmosphere. Once this pressure reaches twice the yield strength of the meteor, then the body will begin to fracture. The approach taken by Lyne *et al.* (1997) to handle each of these effects will be discussed in the next three sections. A more detailed analysis of these approaches can be found in Rich Fought's thesis listed in the references (Fought 1995) and in a soon to be published paper by Lyne *et al.* (1997).

2.2 Entry Aerodynamics

The motion of a meteor, ignoring heating effects, is simply determined by its initial motion, the force of Earth's gravity attracting it, and the counter drag force that slows its descent. With no lifting forces involved, this is referred to as a ballistic entry. The basic equation of motion for a ballistic entry object is

$$m\frac{dV}{dt} = -\frac{1}{2}C_d\rho_a AV^2 - mg\sin\gamma \qquad [2.1]$$

where m is mass of the body, V is the body's velocity, C_d is the coefficient of drag, ρ_a is the local atmospheric density, A is the body's cross-sectional area, g is Earth's gravitational acceleration, and γ is the flight path angle of the body (Vinh 1980). Note that the atmospheric model used for the atmospheric density and the gravitational acceleration at all altitudes was the 1976 NOAA Standard Atmosphere. The flight path angle is measured from zero degrees parallel to the surface to minus ninety degrees perpendicular to the surface. Since meteors generally travel at hypersonic speeds, the drag coefficient is assumed to be dependent on the shape of the windward side of the body only. This assumption comes from Newtonian theory for bodies traveling at hypersonic speeds (Anderson 1989). A second related assumption is that ablation will keep the shape, although not the size, relatively uniform and, therefore, the coefficient of drag will remain constant. For this model, a drag coefficient of 1.2 is used.

The altitude of the meteor is governed by the equation

$$\frac{dH}{dt} = V \sin \gamma \qquad [2.2]$$

where H is the altitude (Vinh 1980). This basic equation of motion simply shows that the altitude is dependent on the downward velocity. Likewise, the flight path angle is determined from

$$V \frac{d\gamma}{dt} = -\left(g - \frac{V^2}{R} \right) \cos \gamma \qquad [2.3]$$

where R is the distance from the center of the Earth (Vinh 1980). This equation accounts for the change in the curvature of the body's path due to the force of gravity.

Since all three of these equations, [2.1] through [2.3], are differential equations and are dependent on the velocity, they must be solved simultaneously. The Lyne *et al.* (1997) code accomplished this using a library of mathematical functions called the International Mathematical Subroutine Library (IMSL). The specific function used was IVPRK, a Runge-Kutta-Verner fifth-order and sixth-order differential equation solver for an initial value problem.

2.3 Body Mechanics

The mechanical model used by Lyne *et al.* (1997) was adopted from work done by Chyba *et al.* (1993). This model assumes the meteor body shape to be a cylinder with spherical caps on each end. The radius of these spherical caps is equal to the length of the center cylindrical section. The body is considered oriented such that one of the spherical caps acts as the windward face. As the meteor passes through the

increasingly dense atmosphere, the pressures on this face build. The body is considered to remain intact until the stagnation pressure on the windward face reaches twice the yield strength of the body. At this point, the pressures at the center of the body are assumed to have reached the yield strength. The entire body then fractures simultaneously, but continues to move as a single body. The stagnation pressure on the body is calculated using equation

$$P_{stag} = \rho_1 (V \sin \beta)^2 \left(1 - \frac{\rho_1}{\rho_2}\right), \qquad [2.4]$$

where P_{stag} is the stagnation pressure, ρ_1 is the density outside of the shock, V is the velocity of the body, β is the local shock angle, and ρ_2 is the density immediately behind the shock (Anderson 1990).

Without the cohesion of the pre-fragmented body, the pressures on the windward face of the body begin to flatten it and the resulting debris starts spreading radially. Chyba *et al.* (1993) formulated the following equation to approximate the change in the radius during this process:

$$r \frac{d^2 r}{dt^2} = \frac{C_D \rho_a V^2}{2 \rho_m} \qquad [2.5]$$

where r is the effective radius of the cross-section and ρ_m is the density of the meteor (which is assumed to remain constant). This increase in the cross-sectional area leads to an increased drag on the body and a larger surface area to absorb the radiated heat.

As the body grows radially, it is assumed that ablation continues to maintain the body's general shape.

2.4 Aerodynamic Heating

The aerodynamic heating and ablation model of Lyne *et al.* (1997) involves a series of calculations that account for many interacting elements that affect the radiative heat transfer during atmospheric entry. The two types of heating that occur during meteor entry are convective heating, from the air passing around the meteor, and radiative heating, from the heat radiating off the hot gases surrounding the body. Studies have shown that the dominant heating for such high-speed entries is due to radiative heat transfer and the convective heating is comparably negligible in these types of problems (Fought 1995).

The method used by Lyne *et al.* (1997) first calculates the pressure and enthalpy downstream of the shock from the equations

$$P_2 = P_1 + \rho_1 (V \sin\beta)^2 \left(1 - \frac{\rho_1}{\rho_2}\right), \text{ and} \qquad [2.6]$$

$$h_2 = h_1 + \frac{1}{2}(V \sin\beta)^2 \left[1 - \left(\frac{\rho_1}{\rho_2}\right)^2\right], \qquad [2.7]$$

where P_1 is the atmospheric pressure and h_1 is calculated from the atmospheric temperature. Note that the atmospheric pressure and temperature at all altitudes was calculated using the 1976 NOAA Standard Atmosphere. The density behind the

shock, ρ_2, is also an unknown in these equations. However, the density ratio (ρ_1/ρ_2) is expected to be very small and obviously the square of that value is also very small, so the density terms have previously been neglected. Using the thermodynamic model of Fought *et al.* (1997), the temperature behind the shock layer could then be determined. This temperature was necessary to determine the radiative heat transfer.

The thermodynamic properties are calculated at several points along the shockwave and vary depending on the shock angle, β, at each location as seen in equations [2.6] and [2.7]. The radiative heating calculated at each of these points is assumed to transfer directly to the body. This results in varied heating across the face of the meteor with the maximum heating at the stagnation point and the minimum heating along the edges.

Basic radiative heating can be approximated using the Stefan-Boltzmann Law for an optically thin gas,

$$q_R = 2\delta\mu\sigma(T_2)^4. \qquad [2.8]$$

where q_R is the radiative heat transfer rate, δ is the shock layer thickness, μ is the average effective absorption coefficient, σ is the Stefan-Boltzmann constant, and T_2 is the temperature of the gas immediately behind the shock. For an optically thin gas, the emissivity can be calculated as $2\delta\mu$, where $\delta\mu$ is much less than one. For a large meteor, the shock layer can be assumed to be optically thick for which $2\delta\mu$ is replaced by an emissivity of one. Therefore, equation [2.8] can be simplified to

$$q_{Rad} = 5.67\left(\frac{T_2}{1000}\right)^4 \qquad \left[\frac{\mathrm{W}}{\mathrm{cm}^2}\right], \qquad [2.9]$$

where q_{Rad} is the adiabatic radiative heat transfer rate. The assumption of optical thickness is reasonable because for large bodies such as those considered in this study, the shock layer is several meters thick and has stagnation pressures on the order of a thousand atmospheres (Lyne 1997).

Once the adiabatic radiative heat transfer rate was determined, it was necessary to account for the nonadiabatic phenomena in the shock layer. The gas cools as it moves downstream from the shock; therefore, the gas temperatures drop from the values calculated using equations [2.6] and [2.7]. An approximation to account for this cooling effect was derived by Robert Goulard (1964) and modified by Tauber (1969) as described by Lyne and Tauber (1995). The adjusted radiative heat transfer is calculated from the equation

$$q_{R2} = \frac{q_{R_{ad}}}{1 + 3.33\Gamma^{0.7}}, \qquad [2.10]$$

where

$$\Gamma = \frac{4q_{R_{ad}}}{\rho V^3}, \qquad [2.11]$$

and Γ is less than or equal to one.

Finally, a portion of the radiative heat transfer is blocked by material that is being ablated off the surface of the meteor. This phenomenon is known as blowing. This was modeled in previous studies by Lyne *et al.* (1996) by using an approximation of results developed by Green and Nicolet (1978) for silicon dioxide (SiO_2) heat shields. This was considered a good choice for the meteor analysis because chondrites are approximately 60% silicon and oxygen by mass (Dodd 1981). The Green-Nicolet approximation correlated the radiation blockage due to blowing to a ratio of the mass flow rate of the ablated material and the mass flow rate in the freestream. A scale factor was multiplied by the radiative heat transfer in the equation

$$q_{R3} = S_{GN} q_{R2}, \qquad [2.12]$$

where the scale factor, S_{GN}, was equal to 0.5 for a mass flow rate ratio greater than 0.677, 1.0 for a ratio of less than 0.1, and determined by the following equation for ratios between 0.1 and 0.677:

$$S_{GN} = 1 - \frac{0.26}{0.3}\left(\frac{\dot{m}}{\rho_1 V} - 0.1\right), \qquad [2.13]$$

where $\dot{m}$ is the mass flow rate of the ablated mass. This approximation is graphically shown in the following chapter in **Figure 3.2**.

CHAPTER 3

MODIFIED TRAJECTORY MODEL

3.1 Objectives

It is the intent of this thesis to improve upon the previously described code written by Evans Lyne and Rich Fought to analyze meteor entry. This chapter describes the changes that were made. The primary objective of this research is to incorporate high temperature thermodynamic data from a Chance Vought report (Anonymous 1961) into the existing meteor entry code to enable the analysis of higher velocity entries than previously possible. The existing code is limited to bodies where the temperature of the gases behind the shock can be no greater than 30,000 K with a pressure of up to 1,000 atm. The new Vought data can extend the temperature range to as high as 100,000 K for all pressures between 0.005 and 1,000 atm and within more limited temperature ranges for pressures as low as 0.00002 atm and as high as 100,000 atm. Because of the temperature constraints, the original code is limited to analyzing meteors traveling at speeds no faster than 20 km/s. The additional data allows analysis of bodies traveling at speeds up to 50 km/s. This will increase the range to include most Earth-crossing asteroids and comets (Chyba 1991).

The secondary objectives include more accurately calculating the pressure and enthalpy equations [2.6] and [2.7] to account for density changes across the shock, replacing the differential equation solver, improving the curve fit of the Green-Nicolet

blowing factor, and converting the program to Fortran 90. Each of these changes has been made for independent reasons. Previously the density ratio across the shock was not included in the pressure and enthalpy equations (the ratio was assumed to be zero in equations [2.6] and [2.7]) because the thermodynamic data was not available, and the value was believed to be negligible. The differential equation solver, DVERK, is incorporated into the code to replace an IMSL routine that was machine dependent. The modified curve fit for the Green and Nicolet correction improves upon a convergence problem with the original curve. Finally, several modifications are made to convert the program from FORTRAN 77 to Fortran 90.

3.2 Chance Vought Research

In the early 1960's, a small group at the Chance Vought Research Center in Dallas recognized the current and future need for high temperature thermodynamic data for air. With the advancement of supersonic aircraft and the beginning of the space race, some research was already being done in this area. However, it was limited to the specific situations that were being dealt with in each case. The Chance Vought group decided to take a more broad approach. They chose several thermodynamic properties, namely temperature, pressure, density, enthalpy, entropy, molar concentration, speed of sound, specific heat at constant pressure, and specific heat at constant volume, and solved them over a broad range. In 1961, they published an internal report of their results (Anonymous 1961). However, the report appears to have never been published externally and rarely used in future research outside of the company.

The biggest difference in the Chance Vought report over similar works was the number of species considered in the calculations. As a gas is heated, the distance between the atoms in a molecule increases. However, the molecule stays intact due to intermolecular forces. As long as these molecules stay intact, the gas is considered "perfect." If the heating continues, eventually the intermolecular forces will be overcome and the atom separates, or dissociates, from the molecule. At that point, the composition of the gas begins to change since the atom is now an individual element within the gas. In addition, the thermodynamic properties of the gas begin to change. A gas undergoing these changes is referred to as a "real" gas. As heating continues, dissociation will continue until all of the atoms have been freed. Another change that begins to happen is ionization of atoms. This generally occurs at higher temperatures than dissociation and involves the electrons within an atom breaking free and adding an additional element to the gas. The temperature ranges over which dissociation and ionization occur are strongly dependent on the type of molecule or atom and on the ambient pressure or density.

Most aerodynamic researchers such as Tannehill *et al.* (1974), Srinivasan *et al.* (1987), Gupta *et al.* (1987, 1990, 1991) and Fought *et al.* (1997) have limited their thermodynamic analysis to the two most common elements in air, nitrogen (N) and oxygen (O), and their molecules and ions. This has generally lead to analysis using no more than 11 species (N_2, O_2, N, O, NO, NO^+, e^-, N^+, O^+, N^{+2}, and O^{+2}). The group at Chance Vought decided to expand their analysis to include a third reference species, Argon (Ar), and a much greater number of ions. In all, their analysis included

30 species (N_2, N_2^+, O_2, O_2^+, N, O, Ar, NO, NO^+, O^-, e^-, N^+, O^+, Ar^+, N^{+2}, O^{+2}, Ar^{+2}, N^{+3}, O^{+3}, Ar^{+3}, N^{+4}, O^{+4}, Ar^{+4}, N^{+5}, O^{+5}, Ar^{+5}, O^{+6}, Ar^{+6}, Ar^{+7}, and Ar^{+8}). While a few other researchers had included Argon, their analysis was limited to a very small range of temperatures and, therefore, to a limited number of the ions (Anonymous 1961).

The Vought group performed their analysis over a range of temperatures from 3,000 K to 100,000 K and a range of density ratios based on the standard atmospheric density from 10^{-6} to 100 (Anonymous 1961). A series of differential thermodynamic equations for enthalpy, entropy, specific heat at constant pressure, specific heat at constant volume, speed of sound, and pressure were used. To solve these equations a partition function and the molar concentrations of the gas were needed. The molar concentrations were solved using a series of chemical equilibrium equations for a reacting gas. To assure equilibrium, a chemical concentration equation, a mass conservation equation, and a charge conservation equation were solved iteratively. For the specific heat equations, two additional sets of equations were required to handle derivatives within the partition function. The partition function used was a modified version of one given in a previous Chance Vought report by P. B. Brandt, one of the major contributors to the later Vought report (Brandt 1959).

The entire approach was a straightforward statistical thermodynamics problem. The most significant differences lie in the partition function used and the large number of equations required to solve for all of the molecules and ions investigated. The complication of these factors was overcome by using 1960's state of the art digital

computers. They allowed the researchers to solve a series of equations that would have been impractical to solve by hand (Anonymous 1961).

The results of this work are found in a series of graphs. Some of these graphs have been included in **Appendix A**. To utilize this data, a series of curve fits had to be written to approximate all of the values between the curves.

3.3 Curve Fits Using ITPLBV

Curve fitting was accomplished by using an existing bivariate interpolation routine written by Hiroshi Akima (1974) and published in the Journal of the ACM. The routine, called ITPLBV, requires a series of data points for a bivariate function, $z = z(x, y)$. The data points need not be equally spaced. The routine then fits a smooth surface to the data using a set of bicubic polynomials in a piecewise fashion. This method requires three partial derivatives, $\frac{\partial z}{\partial x}$, $\frac{\partial z}{\partial y}$, and $\frac{\partial^2 z}{\partial x \partial y}$, at each data point. This was accomplished by using a total of thirteen grid points with the point in question in the center as shown in **Figure 3.1** (Akima 1974).

Each bicubic polynomial corresponds to a rectangle of data points and uses the value of each point and the three partial differential equations of each of the four corner nodes. When an edge point is evaluated, the necessary data points outside of the grid are estimated using the boundary point, the three points inside the grid past the boundary point, and the two adjacent boundary points (Akima 1974).

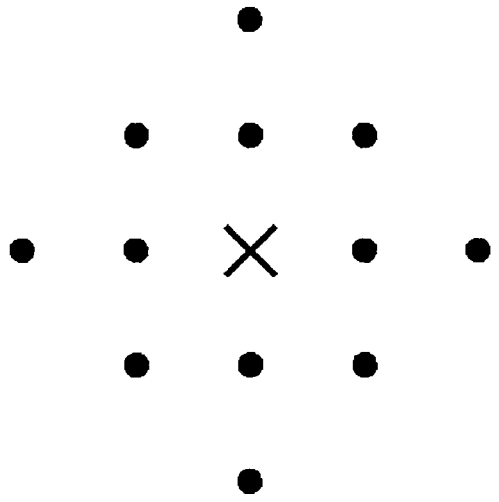


Figure 3.1 - ITPLBV derivative grid

The format of the ITPLBV routine is

```
ITPLBV(IU,LX,LY,X,Y,Z,N,U,V,W),
```

where the inputs are defined as follows:

IU is the logical unit number for the error output,

LX is the number of grid points in the x-dimension,

LY is the number of grid points in the y-dimension,

X is an array of the x-dimension data,

Y is an array of the y-dimension data,

Z is a two-dimensional array of the value at each X and Y coordinate,

N is the number of points to be interpolated in each call of the routine,

U is an array of the x-coordinate values to interpolate from,

V is an array of the y-coordinate values to interpolate from,

and the output, W, is the array of interpolated Z values corresponding to U and V (Akima 1974).

The main advantage of this routine is that it does not use iterative solutions. There are no problems of computational stability or convergence. The main limitation of the routine is that the data must be input in a complete, rectangular grid. For every X from zero to LX and every Y from zero to LY, there must be a corresponding Z value. Also, the data must fit into a smooth, bivariate function (Akima 1974).

3.4 Curve Fitting Chance Vought with ITPLBV

To curve fit the Chance Vought data, the first step was to read approximately 1500 data points from the Chance Vought curves. This number was determined to be sufficient to allow ITPLBV to produce accurate curve fits through trial and error. The data points were separated into two tables of data. One group contains pressure values at given enthalpy and density values. This data was used to interpolate the density. The second group was enthalpy values at given pressures and temperatures. This group was used to interpolate temperatures. Both tables of the Vought data points are given in **Appendix B**. Note that this data is in SI units although most of the data on the Vought graphs is non-dimensional. As seen from the tables, two problems existed in curve fitting this data with ITPLBV. First, not every point on the tables has a value. Second, the value to be interpolated is not the `Z` value in either table.

To resolve the existing problems, separate routines were written to prepare the data for ITPLBV. The `TEMP_FIT` routine was written to interpolate the temperature from a given pressure and enthalpy, and the `DENS_FIT` routine was written to interpolate the density for a given pressure and temperature. Each routine determines where on the table of data the point to be interpolated lies and determines the largest complete, rectangular grid of data containing this value. If the value is outside of the table, the routines flag an error, but then choose the largest possible grid of data on the table and allow the routine to attempt to continue. Next, the `TEMP_FIT` routine takes the given pressure and ten equally spaced temperature

values across the entire range of possible temperatures and lets ITPLBV interpolate enthalpy values for each. The interpolated enthalpy values are compared to the given enthalpy to determine which two points the given enthalpy is between, and the allowable temperature range is adjusted to match. Ten equally spaced temperatures are then determined from the new temperature range, and the process is repeated. During each iteration, all ten enthalpy values are compared with the given enthalpy. When an interpolated enthalpy is within 0.1% of the given enthalpy, it is considered a match, and the corresponding temperature is accepted.

For example, assume a pressure of 20 MPa and an enthalpy of 330 MJ/kg are calculated using equations [2.6] and [2.7], respectively. The routine looks at the table and decides that the largest rectangular grid would be for temperatures from 3,000 K to 100,000 K and for pressures from 506.625 Pa to 101.325 MPa. A set of ten equally spaced temperatures are determined and stored in an array as

```
V(1) =   3,000 K
V(2) =  13,778 K
V(3) =  24,556 K
V(4) =  35,334 K
V(5) =  46,112 K
V(6) =  56,889 K
V(7) =  67,667 K
V(8) =  78,445 K
V(9) =  89,223 K
```

V(10) = 100,000 K

When the enthalpy values from these interpolations are compared with the given enthalpy of 330 MJ/kg, none of them are within 0.1%, and it is found that the value lies between W(4) and W(5). A new set of temperatures is chosen between 35,334 K, V(4), and 46,112 K, V(5), and the process repeats.

An identical procedure is used for the DENS_FIT routine by using the density instead of the temperature and using the given enthalpy with the density to interpolate pressures.

Although not needed for this particular trajectory code, four similar routines were written to interpolate enthalpy from pressure and temperature (ENTH_FIT), pressure from enthalpy and density (PRES_FIT), enthalpy from pressure and density (ENTH_FIT2), and pressure from temperature and enthalpy (PRES_FIT2). These routines also allow other possible solutions by combining two of them. For example, density can be solved from pressure and temperature by first solving ENTH_FIT for enthalpy and then using the pressure and enthalpy in DENS_FIT to determine the density.

All of these routines are included in a listing of the entire METEOR program in **Appendix C**.

A comparison of this thermodynamic model with the most recent Tannehill model by Srinivasan *et al.* (1987) and the most recent Gupta model by Fought *et al.* (1997) is discussed in Chapter 4.

3.5 Utilizing the Curve Fits in the METEOR Program

There are two parts of the original program that require the thermodynamic curve fits. First, to calculate the temperature inside the shock from the local pressure and enthalpy, the `TEMP_FIT` routine is needed. Secondly, to add the density ratio term to the pressure equation [2.6] and the enthalpy equation [2.7], the `DENS_FIT` routine is needed to determine the density behind the shock.

The `TEMP_FIT` routine is a replacement for the `TEMPNEW` routine by Rich Fought. The `TEMPNEW` routine used a modified version of Gupta's curve fits (Gupta 1991) to extend the temperature range to 30,000 K at 1,000 atm (Fought 1995). As mentioned earlier, the new routine allows interpolation of thermodynamic data at temperatures as high as 100,000 K to 100,000 atm. However, the Vought data only extends down to 3,000 K, and meteor trajectories commonly require some interpolations below this value. Therefore, the Fought routine has been left in the program to handle interpolations at enthalpies below 7.44580512 MJ/kg. The cutoff is in terms of the enthalpy, because the nature of the program does not allow a temperature cutoff to be feasible. The enthalpy value used insures that all interpolations under 3,000 K will be performed by `TEMPNEW`, and that it will not have to handle any interpolations above 5,500 K.

The use of the DENS_FIT routine allowed the calculation of a term previously assumed negligible. It is used to determine the density inside the shock, ρ_2, for equations [2.6] and [2.7]. It also has the same limitation as TEMP_FIT of a minimum temperature of 3,000 K. For temperatures below that, the normal shock equations are used to determine the density (Anderson 1990).

A comparison of the changes will be discussed in Chapter 4.

3.6 Solving Differential Equations with DVERK

One of the major limitations of the T10 code was its dependence on the IMSL routine, IVPRK, to solve the differential equations of motion. This is a package of mathematical routines that must be purchased for specific machines. Therefore, this dependence restricted the program to running on those machines. The DVERK routine that replaced it is simply a Fortran routine that can be compiled with the code on any machine.

The IVPRK routine is an ordinary differential equation solver based on the Runge-Kutta-Verner fifth-order and sixth-order method. A nearly identical routine, DVERK, written by Jackson *et al.* (1976) was used to replace IVPRK. The only significant difference in calling the two routines is that DVERK requires a pre-allocated workspace be passed as a two-dimensional array.

Because of the similarity with the previous routine, no detailed testing was done. Instead comparisons done for other purposes were relied upon to verify the DVERK routine.

3.7 Green-Nicolet Adjustment

The Green-Nicolet approximation was previously added to the code in an attempt to account for the blowing effects of the ablated material that blocks radiative heat transfer to the body. The approximation is correlated with the ratio of the mass flow rate of the ablated material and the mass flow rate of the freestream. It was defined as 50% blockage for a ratio greater than 0.677, no blockage for a ratio of less than 0.1, and according to the following equation for the range between 0.1 and 0.677:

$$1-\frac{0.26}{0.3}\left(\frac{\dot{m}}{\rho_1 V}-0.1\right) \qquad [2.13]$$

where $\dot{m}$ is the mass flow rate of the ablated mass, ρ_1 is the atmospheric density, and V is the velocity of the body. This can be seen graphically in **Figure 3.2** along with a modified curve in **Figure 3.3**.

The blockage has to be solved iteratively, because the mass flow rate of ablated material is dependent on the radiative heating and the radiative heating is dependent on the mass flow rate. Since the curve of **Figure 3.2** is not smooth, a solution near one of the two corners would jump back and forth around the corner and fail to

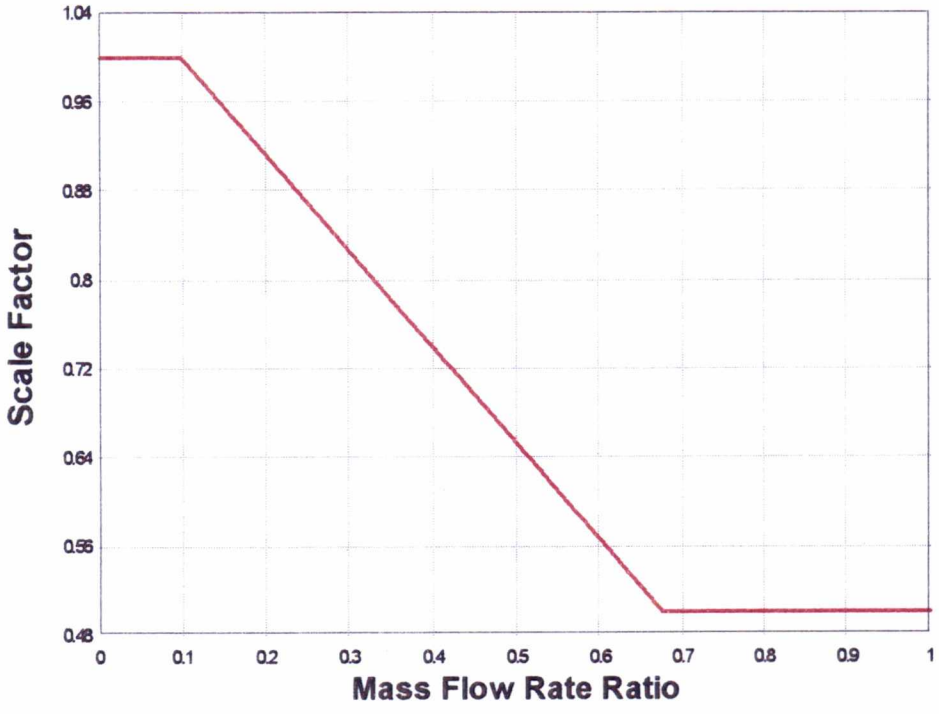


Figure 3.2 - Green-Nicolet Approximation

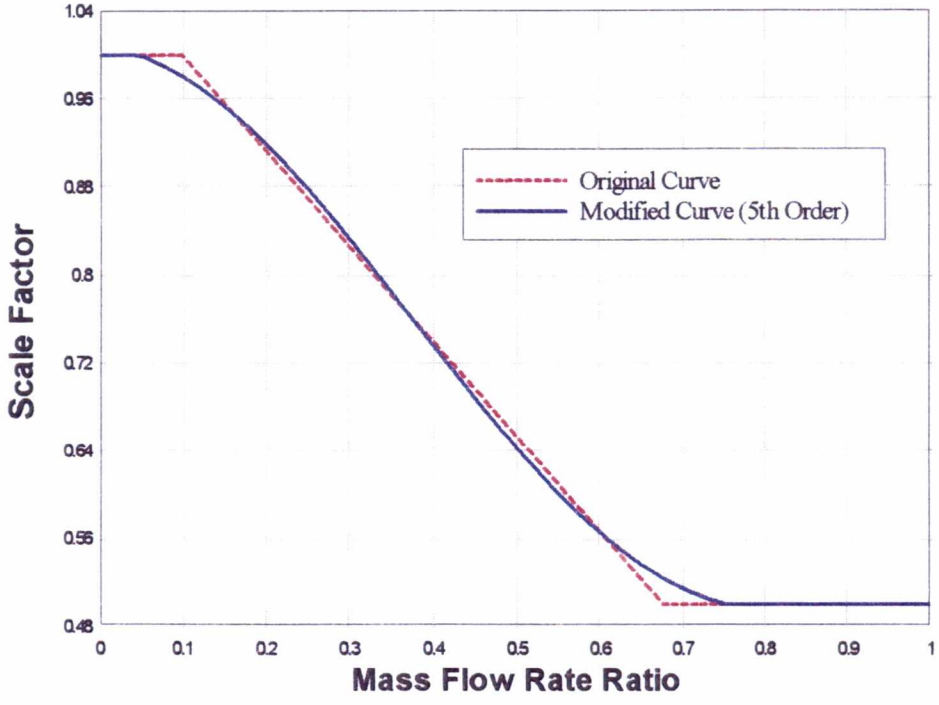


Figure 3.3 - Modified Green-Nicolet Approximation

converge on a solution. After a set number of iterations, the code simply assumed, correctly, that no convergence was possible and accepted the current scale factor that would be either too high or too low.

To improve upon this approach, a fifth-order curve fit was calculated to replace the transitional part of the curve. The new curve assumes 50% blockage for a mass flow rate ratio above 0.754793198787, no blockage for ratios below 0.04802734866, and the following equation between these values:

$$1.0095555 - 0.11552317\frac{\dot{m}}{\rho_1 V} - 1.7206431\left(\frac{\dot{m}}{\rho_1 V}\right)^2 - 0.53616033\left(\frac{\dot{m}}{\rho_1 V}\right)^3 + 4.1602124\left(\frac{\dot{m}}{\rho_1 V}\right)^4 - 2.2932833\left(\frac{\dot{m}}{\rho_1 V}\right)^5 \quad [3.1]$$

This modified Green-Nicolet approximation can be seen in relation to the original curve in **Figure 3.3**.

This modification still leaves a curve that is not smooth, but it greatly reduces the discrepancy at the transition between curves. A comparison of the modification is given in Chapter 4.

3.8 FORTRAN 77 to Fortran 90

The primary reason for the conversion of the program from FORTRAN 77 to Fortran 90 was the addition of some array enhancements. Since the Vought data was in large, incomplete tables and the ITPLBV routine required complete, rectangular

grids of data, it was necessary to use different sections of the array at different times. The only way to do this in FORTRAN 77 was to create entirely new arrays of the desired size and copy the data over for each different section needed. Fortran 90 allows referencing sections of the array as independent arrays.

For example, a 4x4 array would be declared as

```
DIMENSION ARRAY(4,4).
```

To utilize an array consisting of only the center four nodes, a reference can be made as

```
ARRAY(2:3,2:3).
```

This is treated as a 2x2 array consisting of the four central nodes in ARRAY. This greatly simplified programming the curve fits.

CHAPTER 4

ERROR ANALYSIS

4.1 Thermodynamic Curves: Vought vs. Tannehill

To confirm the validity of both the Chance Vought curves and the routines written to fit them, comparisons were made with other current thermodynamic models. The first comparison was made with the most recent version of Tannehill's curve fits (1974) by Srinivasan *et al.* (1987). Routines were written using the equations and constants derived by Tannehill. However, Tannehill's work was limited to routines accepting density and pressure, internal energy and density, or pressure and entropy as inputs. Internal energy was not included in the Vought report, so no comparison could be made with those Tannehill routines, and entropy was not completed at the time of this report because it was not necessary for meteor entry analysis. Therefore, only the two of Tannehill's routines that accepted pressure and density could be used for comparison. These routines returned temperature and enthalpy.

Comparisons were made by randomly selecting a pressure and a density, within the valid range of Vought and Tannehill, for 10,000 points and determining the difference between the values returned by the two thermodynamic models at each point. The points were separated into groups according to the percent difference between each model's solution. The groups were divided as follows: below 1%, 1%

to 2%, 2% to 3%, 3% to 4%, above 4%. These points were then plotted in **Figures 4.1** and **4.2** according to the percent difference. The edges of the data are bound vertically by the minimum and maximum temperatures of 3,000 K and 23,000 K and horizontally by the minimum and maximum densities of 1.2251×10^{-6} kg/m^3 and 68.892 kg/m^3. These limits were set to stay within the Vought minimum temperature, the Tannehill maximum temperature, and the Vought density ranges. The Tannehill curves are said to be valid up to 25,000 K, but analysis was limited to 23,000 K because of the limited number of ions included in the Tannehill model.

Besides the limiting boundaries, which were mentioned above and can clearly be seen in the graphs, all empty space within the limits are areas of error below 1%. The number of points used gave thorough coverage of the entire range. The average error for the enthalpy data was 0.84% and for the temperature data was 1.05%. The maximum error for the enthalpy data was 6.02% and for the temperature data was 11.46%. However, all errors above 5% fell in the upper temperature ranges and are most likely a result of the species neglected in the Tannehill model. Of the 10,000 points compared, less than 0.83% were above 5% in either graph. The breakdown of the error data can be seen in **Table 4.1** for the enthalpy data and **Table 4.2** for the temperature data.

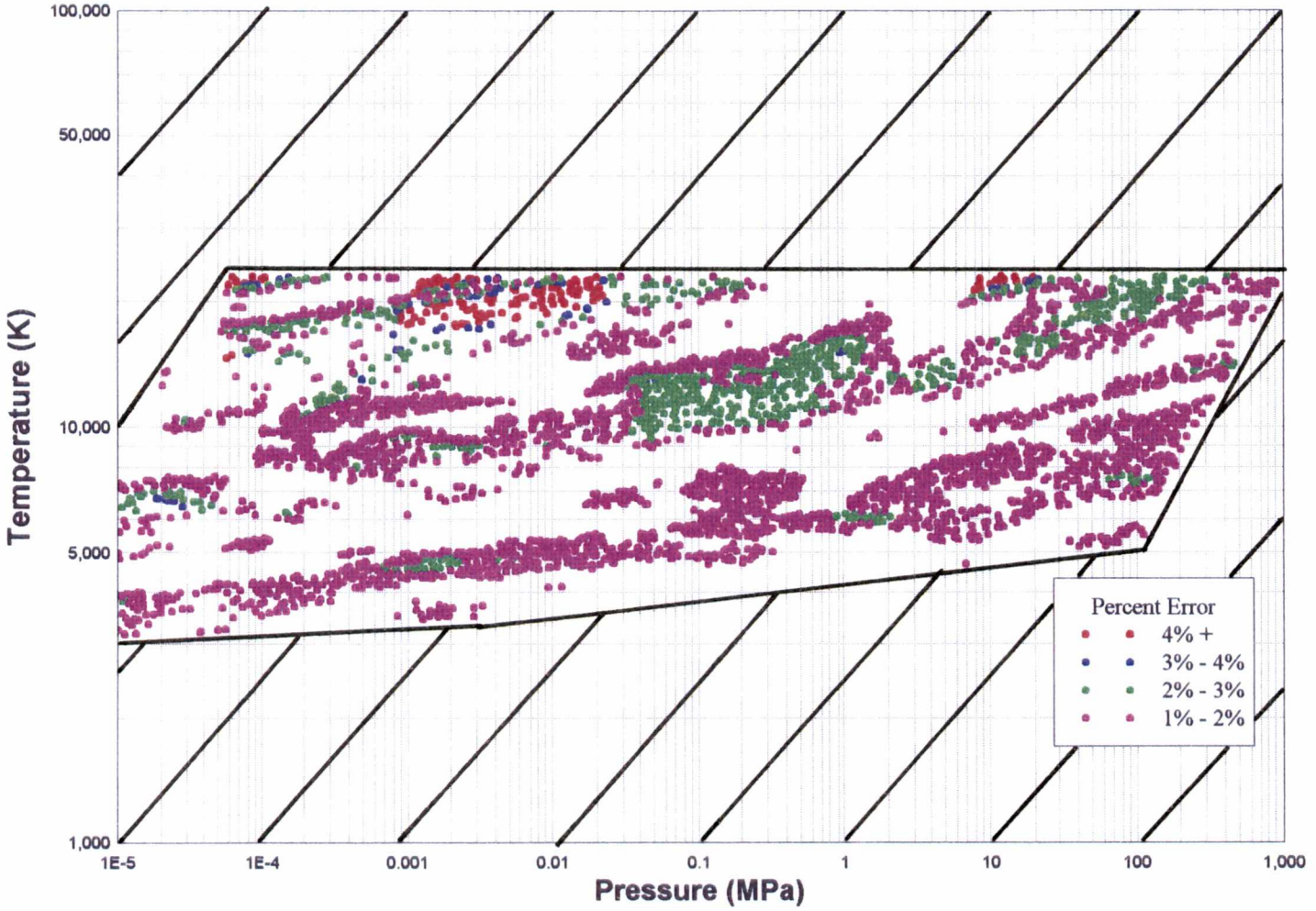


Figure 4.1 - Vought vs. Tannehill temperature (pressure, density)

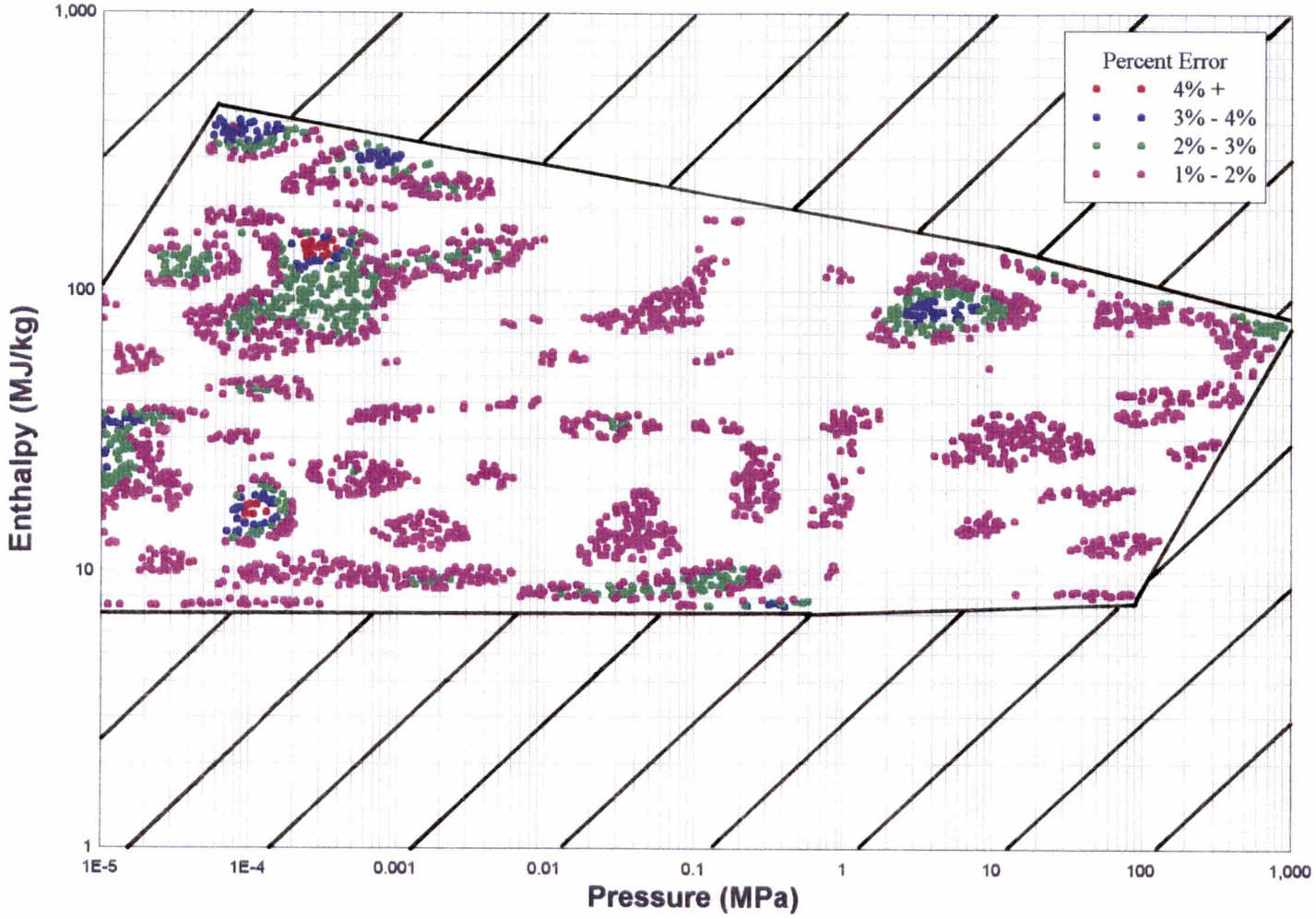


Figure 4.2 - Vought vs. Tannehill enthalpy (pressure, density)

Table 4.1 - Enthalpy (pressure, density) comparison

Range	Percent of 10,000 Points
0% - 1%	69.88%
1% - 2%	22.76%
2% - 3%	5.59%
3% - 4%	1.50%
4% - 5%	0.19%
5% +	0.08%

Table 4.2 - Temperature (pressure, density) comparison

Range	Percent of 10,000 Points
0% - 1%	55.72%
1% - 2%	33.72%
2% - 3%	8.36%
3% - 4%	0.67%
4% - 5%	0.70%
5% +	0.83%

4.2 Thermodynamic Curves: Vought vs. Fought

The second comparison for the Chance Vought curve fits was made with the thermodynamic model used in the T10 code and written by Rich Fought (1995). This routine was based on work done by Fought *et al.* (1997) to expand a previous Gupta model (Gupta 1991). Fought's work was focused on improving a previous meteor trajectory code and outputs temperature from a pressure and enthalpy input. This is the routine that the TEMP_FIT code, described in Chapter 3, was written to replace.

A procedure identical to that used to compare the Tannehill curves in section 4.1 was used with the Fought curves. The limits were adjusted to a maximum temperature of 29,000 K and a maximum pressure of 100 MPa. The maximum temperature was changed because the Fought curves are valid at higher temperatures than the Tannehill curves. The pressure bounds were set by limitations in the Fought curves. A graph of the data is shown in **Figure 4.3**.

The same ranges were used in dividing the results, and the overall error was even smaller than the Tannehill results. The average error was 0.73% and the maximum error was 6.52%. The breakdown of the error is in **Table 4.3**.

Table 4.3 - Temperature (pressure, enthalpy) comparison

Range	Percent of 10,000 Points
0% - 1%	76.14%
1% - 2%	19.19%
2% - 3%	3.10%
3% - 4%	1.00%
4% - 5%	0.39%
5% +	0.18%

As an additional comparison, some constant pressure curves were calculated using the Vought and Fought models to graphically compare the two models. A section of these plots is shown in **Figure 4.4**. It can clearly be seen that both

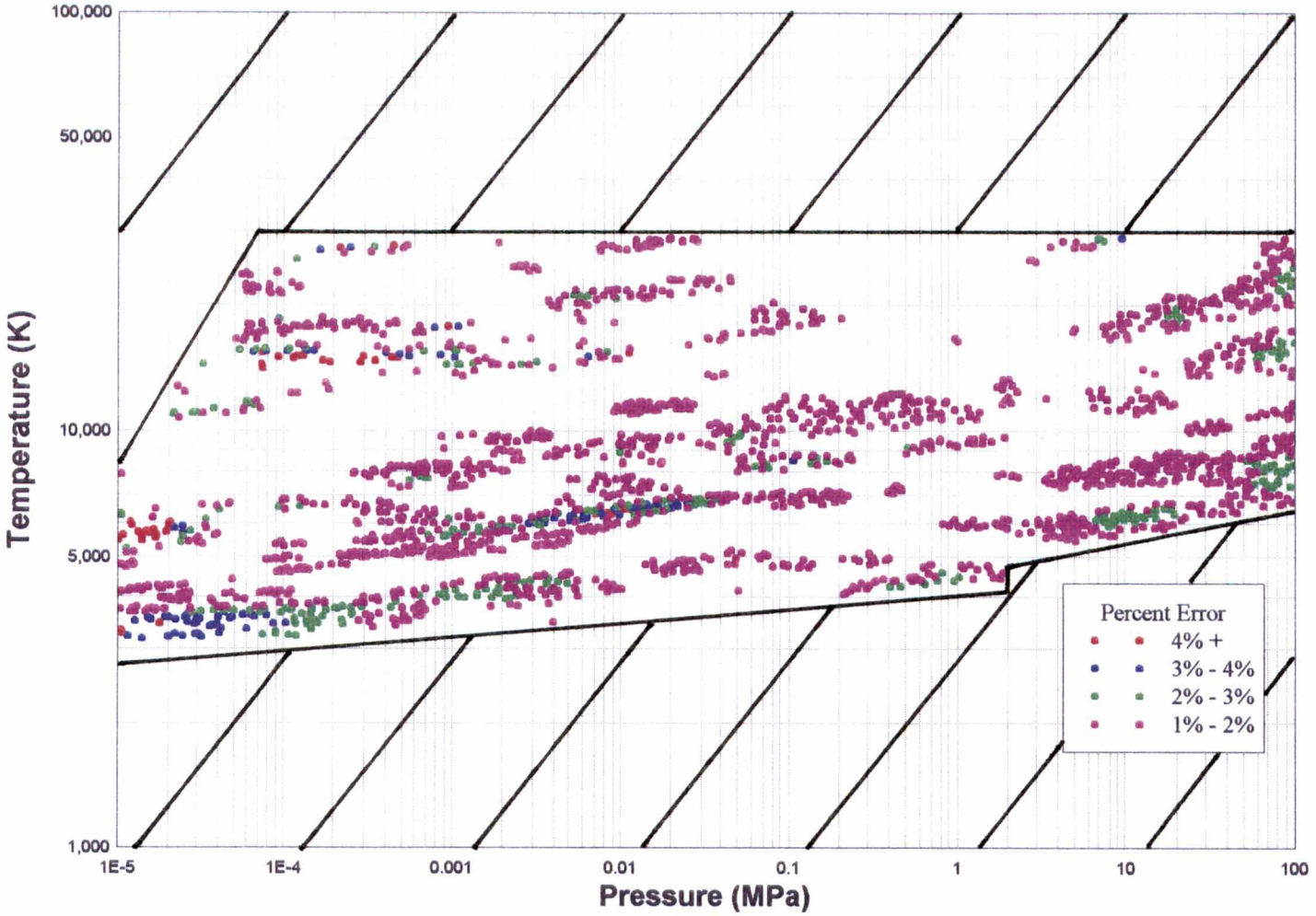


Figure 4.3 - Vought vs. Fought temperature (pressure, enthalpy)

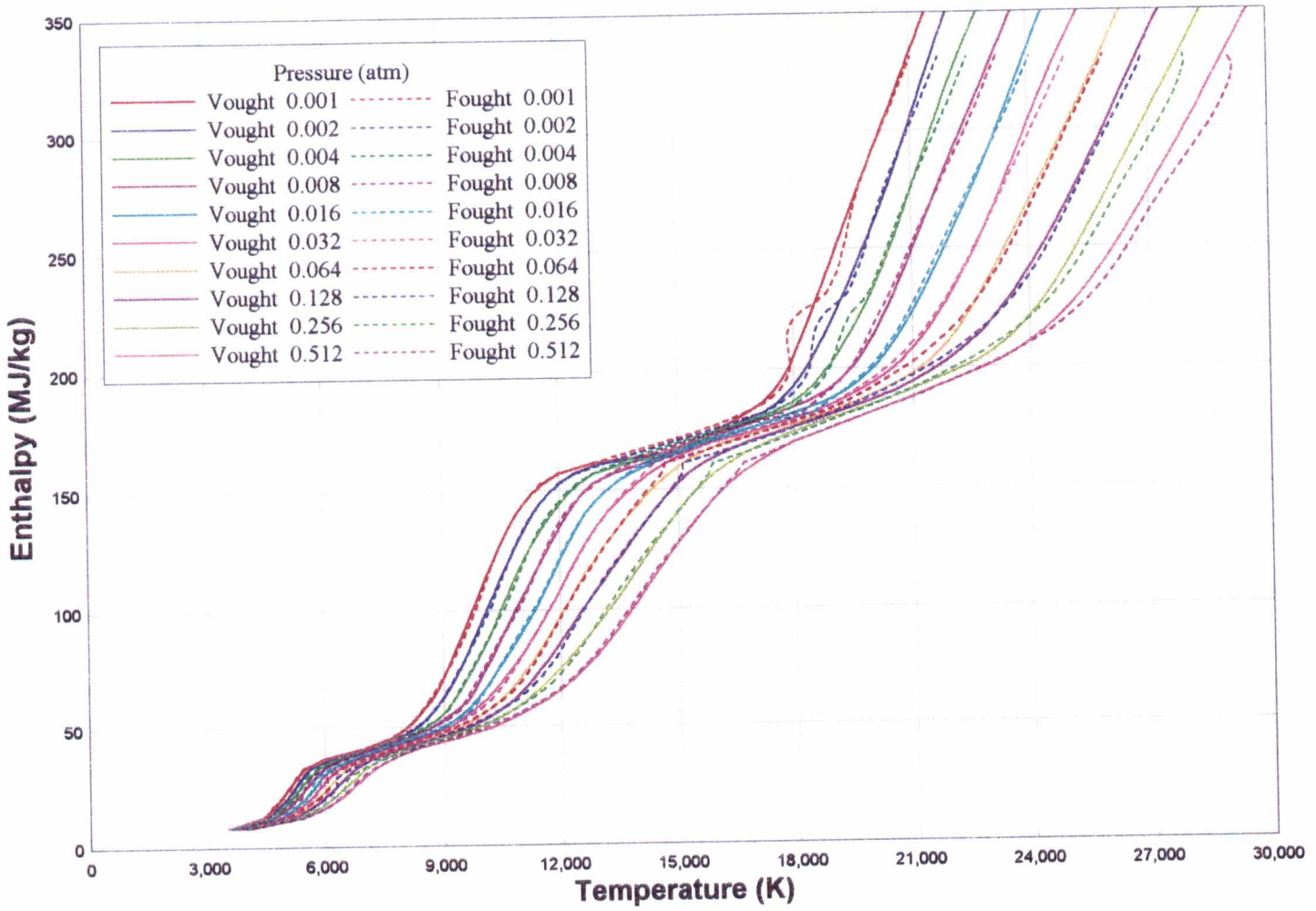


Figure 4.4 - Vought vs. Fought low pressure thermodynamic curves

follow the same basic trend, but the Vought curves are smoother. This is most likely because the Vought curves account for argon ionization and additional nitrogen and oxygen ionizations, while the Fought curves do not and ultimately must catch up to stay with the trend.

Although comparisons could only be done on three of the thermodynamic routines, the additional routines were assumed valid because they use the same data and methods as the validated routines.

4.3 Trajectory Comparisons

With the thermodynamic routines validated, the next check was to compare trajectories calculated using the new code, METEOR, with those calculated using the old, T10. These comparisons could only be done within the velocity limits of T10 of approximately 19 km/s. The only noticeable differences expected within this range were due to the added density term in the pressure and enthalpy equations. The thermodynamic routines used in T10 and METEOR were shown in section 4.2 to match closely and the density effect on the pressure and enthalpy equations were expected to be small. Therefore, the curves were expected to match very closely.

The physical properties of the bodies used for comparisons were taken from Fought's thesis and are shown in **Table 4.4** (Fought 1995). Some of the trajectories are shown in **Figures 4.5** through **4.11**, and, as expected, show nearly identical

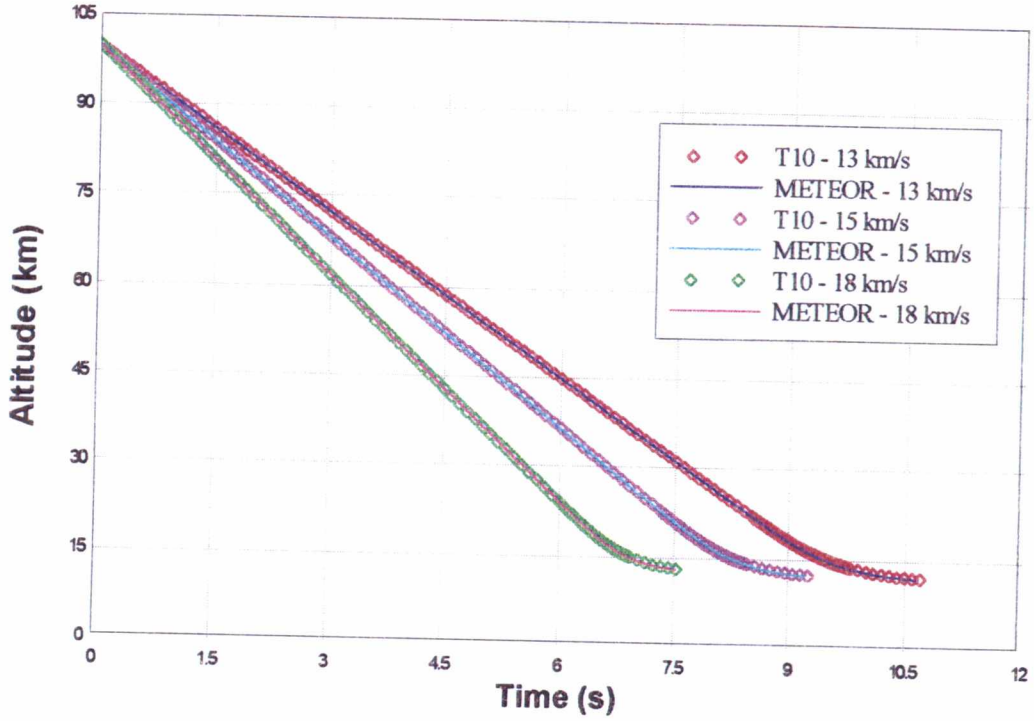


Figure 4.5 - METEOR vs. T10 time (Comet 50 m diam.)

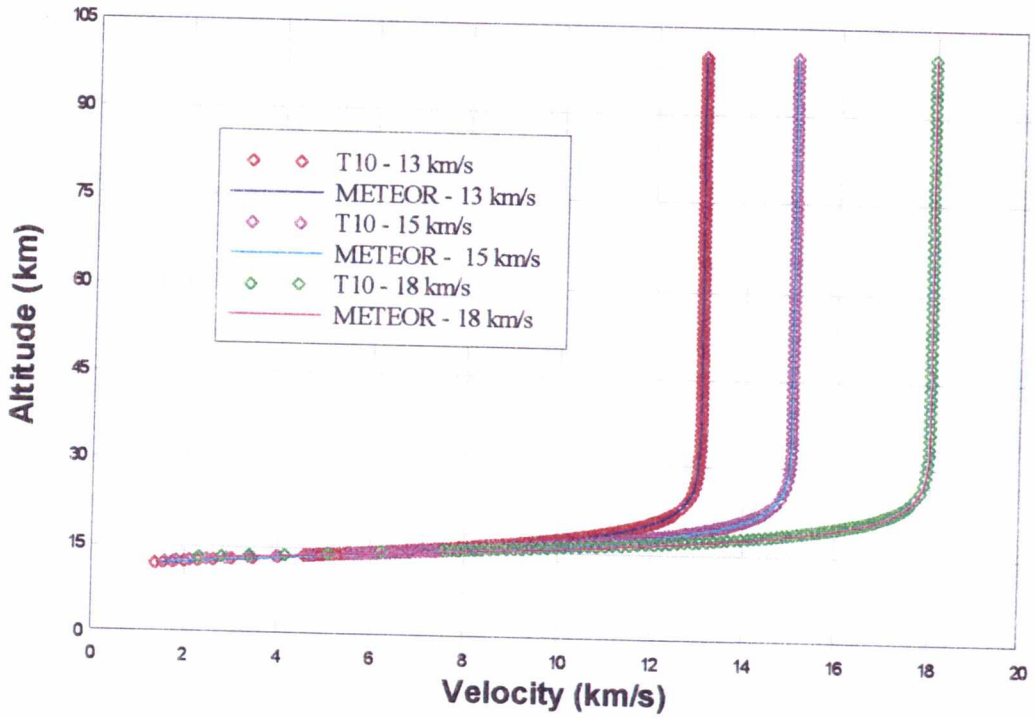


Figure 4.6 - METEOR vs. T10 velocity (Comet 50 m diam.)

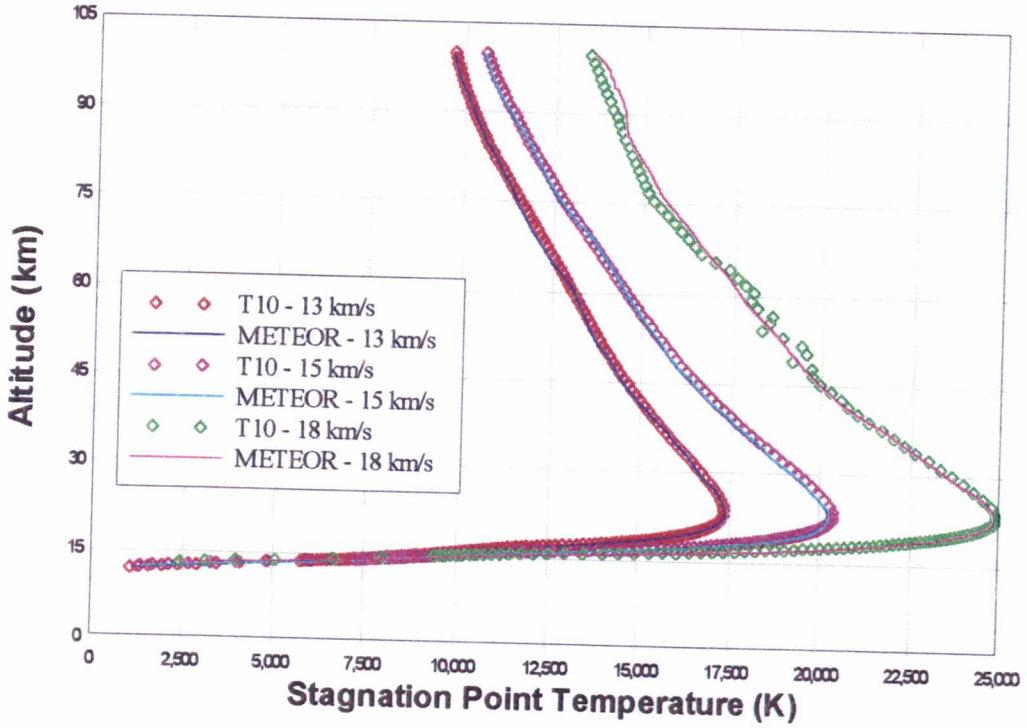


Figure 4.7 - METEOR vs. T10 stagnation point temperature (Comet 50 m diam.)

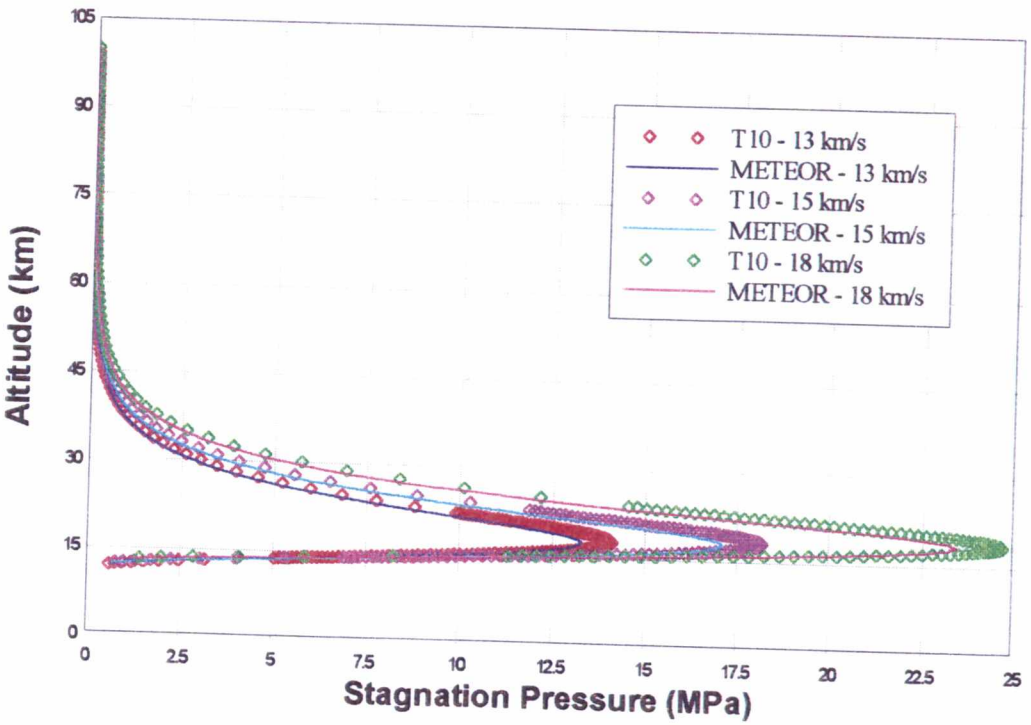


Figure 4.8 - METEOR vs. T10 stagnation pressure (Comet 50 m diam.)

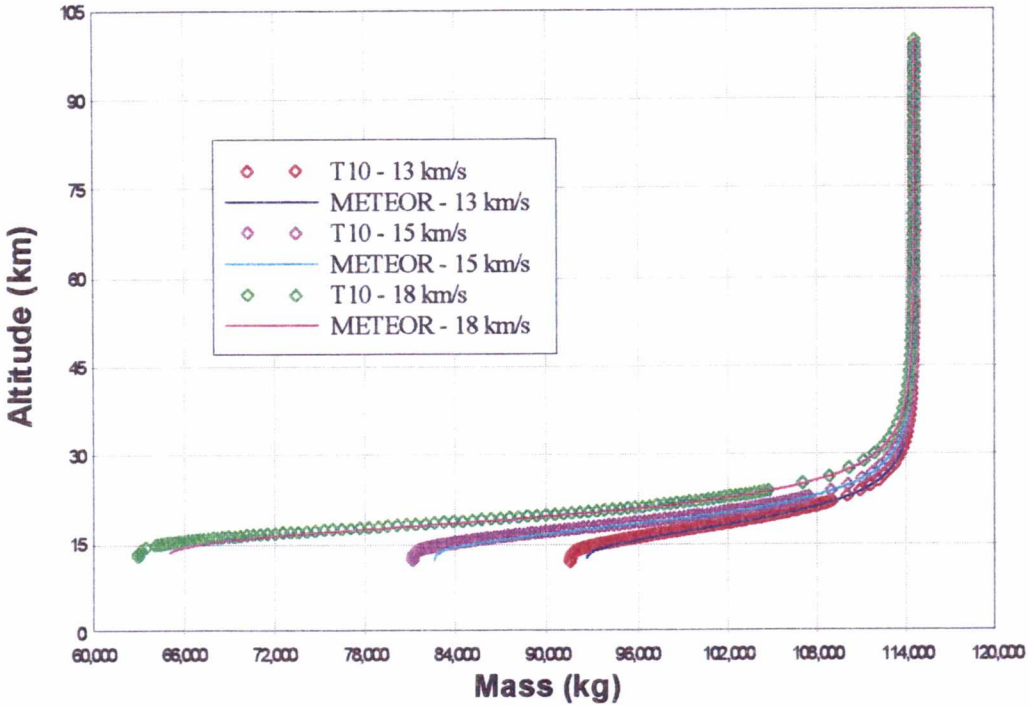


Figure 4.9 - METEOR vs. T10 body mass (Comet 50 m diam.)

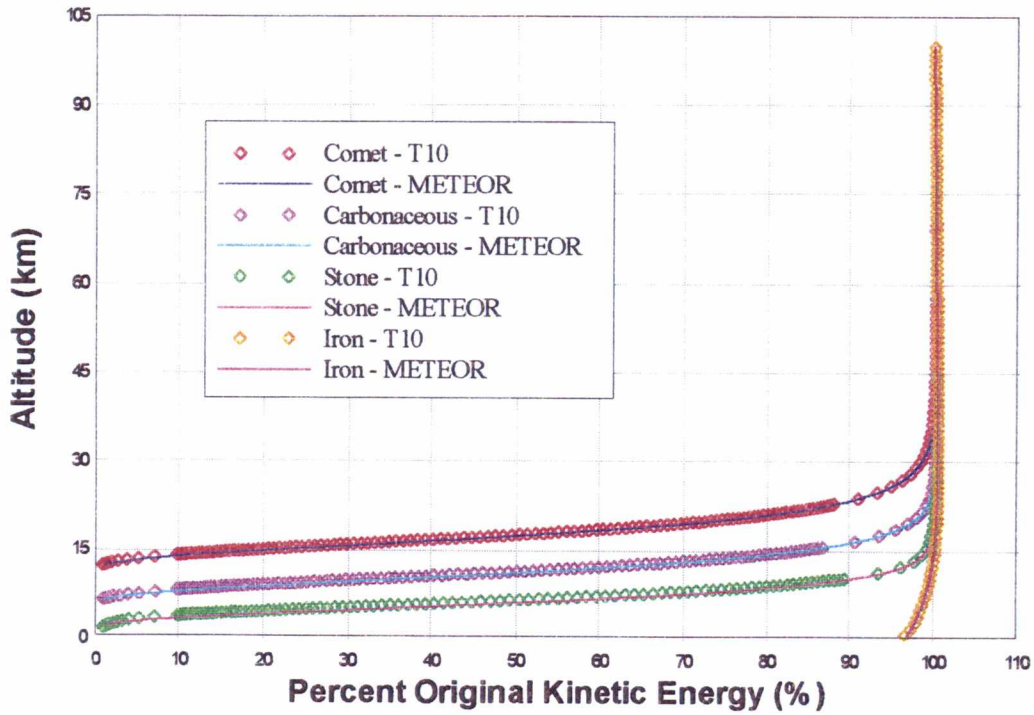


Figure 4.10 - METEOR vs. T10 kinetic energy (15 km/s, 50 m diam.)

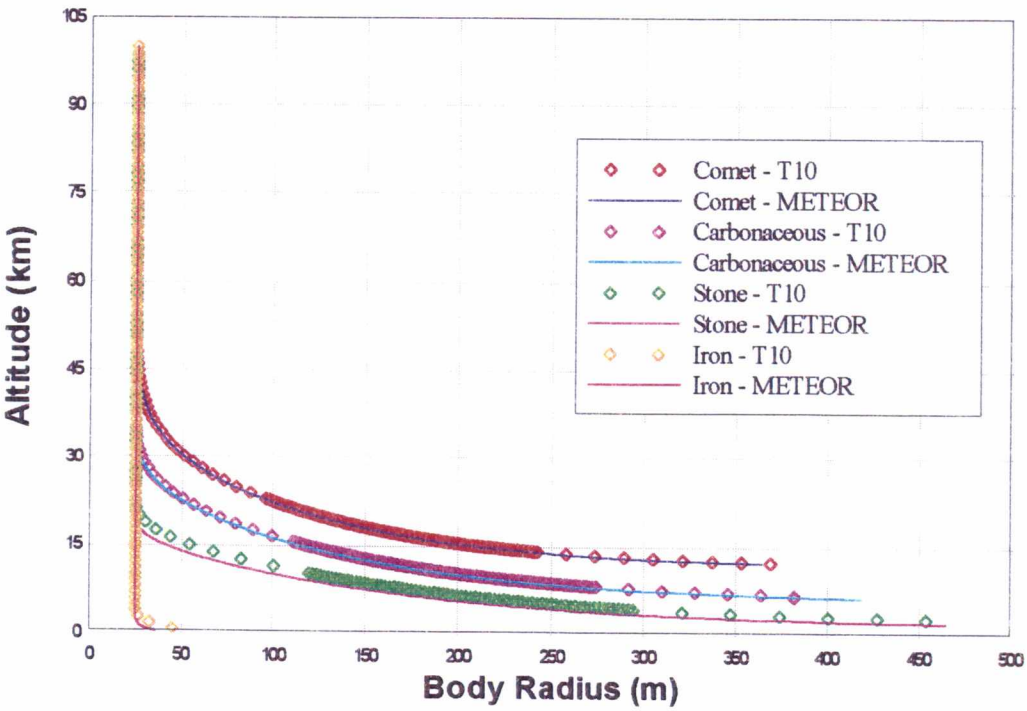


Figure 4.11 - METEOR vs. T10 body radius (15 km/s, 50 m diam.)

results. Calculations for other bodies also show very similar results for the two codes.

Table 4.4 - Fought's physical characteristics of entry bodies (Source: Fought, Richard M. *A Computer Model and Parametric Study of Earth-Atmospheric Entry of Meteors*. The University of Tennessee: Knoxville, TN, 1995.)

Material	Density (kg/m^3)	Strength (N/m^2)	Heat of Ablation (J/kg)
Iron	7900	10^8	8×10^6
Stone	3500	10^7	8×10^6
Carbonaceous	2200	10^6	5×10^6
Comet	1000	10^5	2.5×10^6

The only notable differences between the METEOR and T10 trajectories are found in the stagnation pressure, the body mass, and the stagnation point temperature. The stagnation pressure and mass differences appear to be a result of the additional density terms. The term slightly reduces the pressure, which results in a slightly lower maximum stagnation pressure as seen in **Figure 4.8**. This, combined with a much smaller change in the enthalpy, leads to a very small change in the temperature within the shock. This in turn leads to a reduced heating of the body and ultimately a slightly smaller mass loss as is seen in **Figure 4.9**. The only other discrepancy noticed is on the high velocity bodies in the stagnation point temperature graph, **Figure 4.7**. The initial separation of the curves appears to be due to the argon ionization, which extends into the second ionization at high altitudes. However, trajectories at these speeds are near the upper limits of the T10 code, and inconsistencies can already be seen in part of this curve.

4.4 Green-Nicolet Comparison

Lastly, analysis was done to confirm that the changes made to the Green-Nicolet blowing correction described in section 3.7 would improve the results. A previous trajectory that had shown a jump in the heat transfer curve as a result of the failure of the approximation to converge was rerun with the new approximation. These two heat flux curves are plotted in **Figure 4.12**. The new method yields a significantly smoother result. Plots of other characteristics of the trajectory showed no other significant effects, besides a less than 2% change in the final body mass.

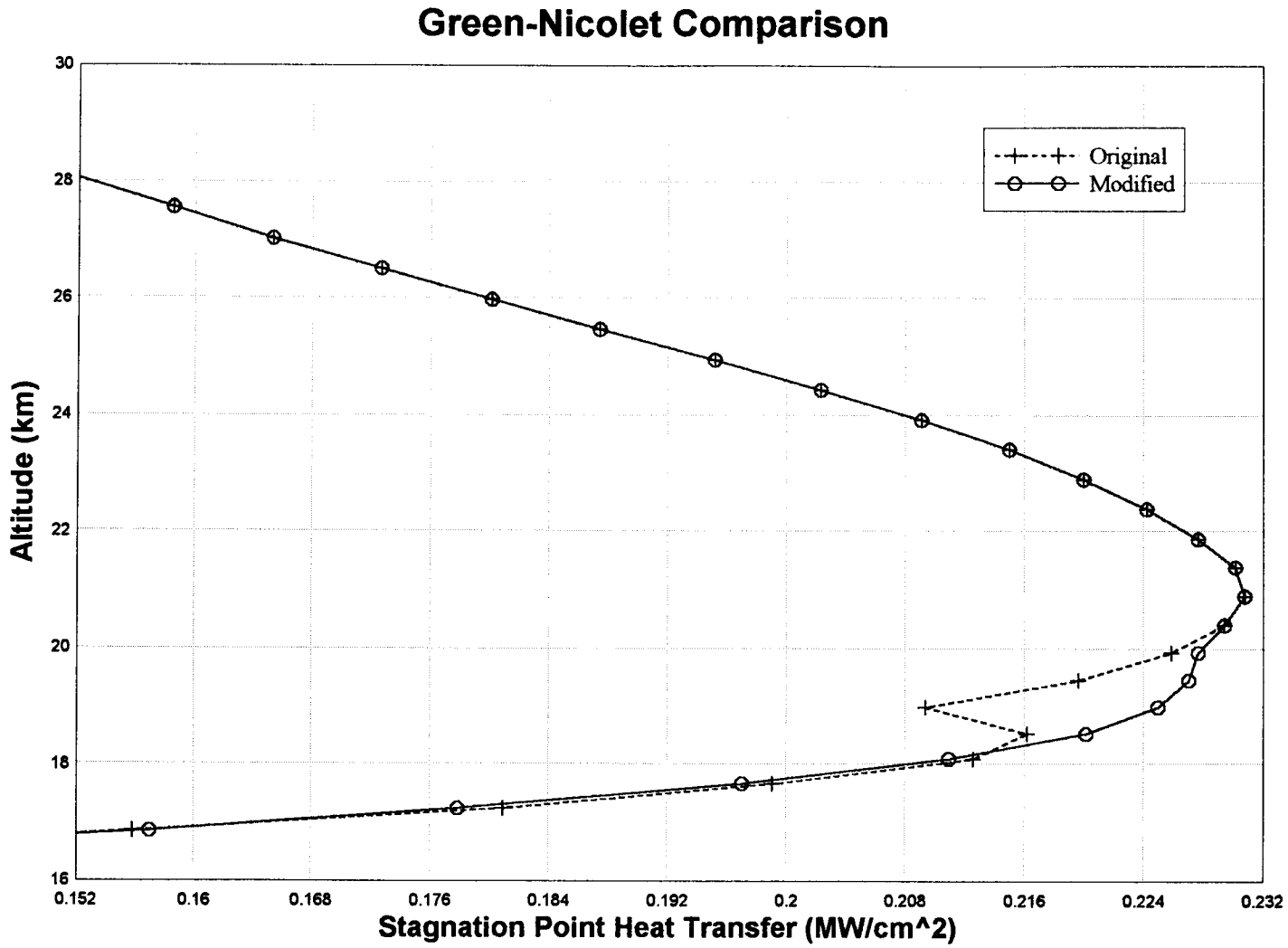


Figure 4.12 - Green-Nicolet Approximation (Comet 15 km/s, 50 m diam.)

CHAPTER 5

TRAJECTORY ANALYSIS

5.1 Analysis Approach

Previous research has commonly used specific types of bodies, i.e. comets and iron asteroids, with set densities, heat of ablations, and yield strengths, such as those found in **Table 4.4** and used in the comparisons in Chapter 4. Rather than focusing on such specific bodies, a much broader approach has been used to show the results of varying body parameters. However, it is not reasonable to simply let the density, heat of ablation, and yield strength vary from a minimum to a maximum value, because the three parameters are not entirely independent. While there is no steadfast formula correlating the three, it is not feasible to say that a body has the maximum possible yield strength and the minimum possible density.

To avoid unreasonable bodies, the range of the parameters has been broken into three groups of meteor types: weak, medium, and strong. Weak bodies would include objects such as comets and weak carbonaceous asteroids. Medium bodies would include most carbonaceous asteroids and weak stony asteroids. Strong bodies would include the stronger stony asteroids and iron asteroids. Within each group, the body parameters of density, heat of ablation, and yield strength, were given a minimum, median, and maximum value. The values used for each set of bodies are shown in **Table 5.1**.

Table 5.1 - Body Physical Parameters

Weak			
	Min	Median	Max
Body Density (kg/m^3)	1000	2250	3500
Heat of Ablation (MJ/kg)	2.5	3.5	4.5
Yield Strength (MPa)	0.1	1.0	10.0
Medium			
	Min	Median	Max
Body Density (kg/m^3)	3000	4500	6000
Heat of Ablation (MJ/kg)	4.0	5.25	6.5
Yield Strength (MPa)	5	25	50
Strong			
	Min	Median	Max
Body Density (kg/m^3)	5500	6750	8000
Heat of Ablation (MJ/kg)	6.0	7.0	8.0
Yield Strength (MPa)	10	50	100

The median body for each group was used for analysis. Additional bodies were evaluated by using a minimum or maximum value of one body parameter and the median values for the other two parameters. Therefore, seven bodies were analyzed in each group, for a total of twenty-one.

Each of these bodies was run through sixty-four different trajectories depending on entry velocity, entry angle, and initial body radius. Four different values were used for each entry parameter as shown in **Table 5.2**. All of these values were interchanged to achieve the sixty-four trajectories for each body. The entry angle is oriented with 90° perpendicular to the atmosphere and 0° parallel to the atmosphere.

Table 5.2 - Entry Parameters

Entry Velocity	25, 35, 40, 45 km/s
Entry Angle	15°, 30°, 45°, 90°
Body Radius	10, 50, 100, 250 m

Trajectories were performed by setting initial values for these entry parameters and calculating the trajectory from 100 km above the surface of the Earth until the kinetic energy dropped below 1% of the initial kinetic energy or when the body was below 1 km altitude.

The following sections describe some of the results from these trajectories. This chapter is not meant to be an exhaustive study of all possible meteor trajectories. It is simply a sampling of the trajectories possible with the METEOR program. Therefore, the various bodies used in the graphs throughout this chapter are not necessarily the same, but have been chosen because they provide a good representation of the trends.

5.2 Trajectory Analysis: Entry Parameters

This section gives a comparison of how the entry parameters (velocity, angle, and body radius) affect the trajectory of the meteor. Most of the data in this section comes from trajectories of the median weak body. However, a few of the graphs have been taken from other bodies, because they give a better representation of the trends found.

Figures 5.1 and **5.2** show the rate at which the body falls toward Earth. The first figure shows how the variation of the entry velocity affects this. As expected, the

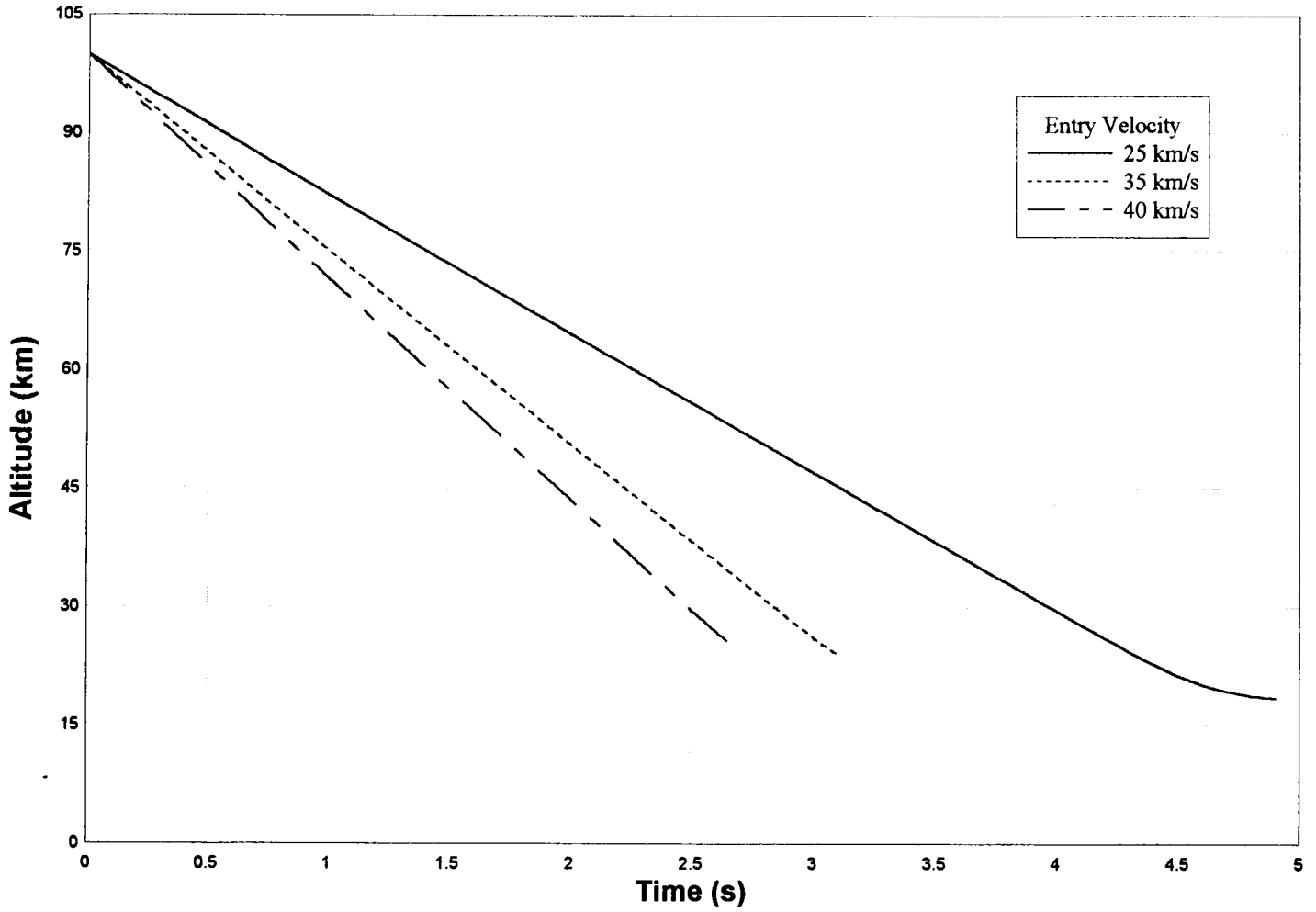


Figure 5.1 - Altitude vs. Time (Weak median body, 10 m radius, 45°)

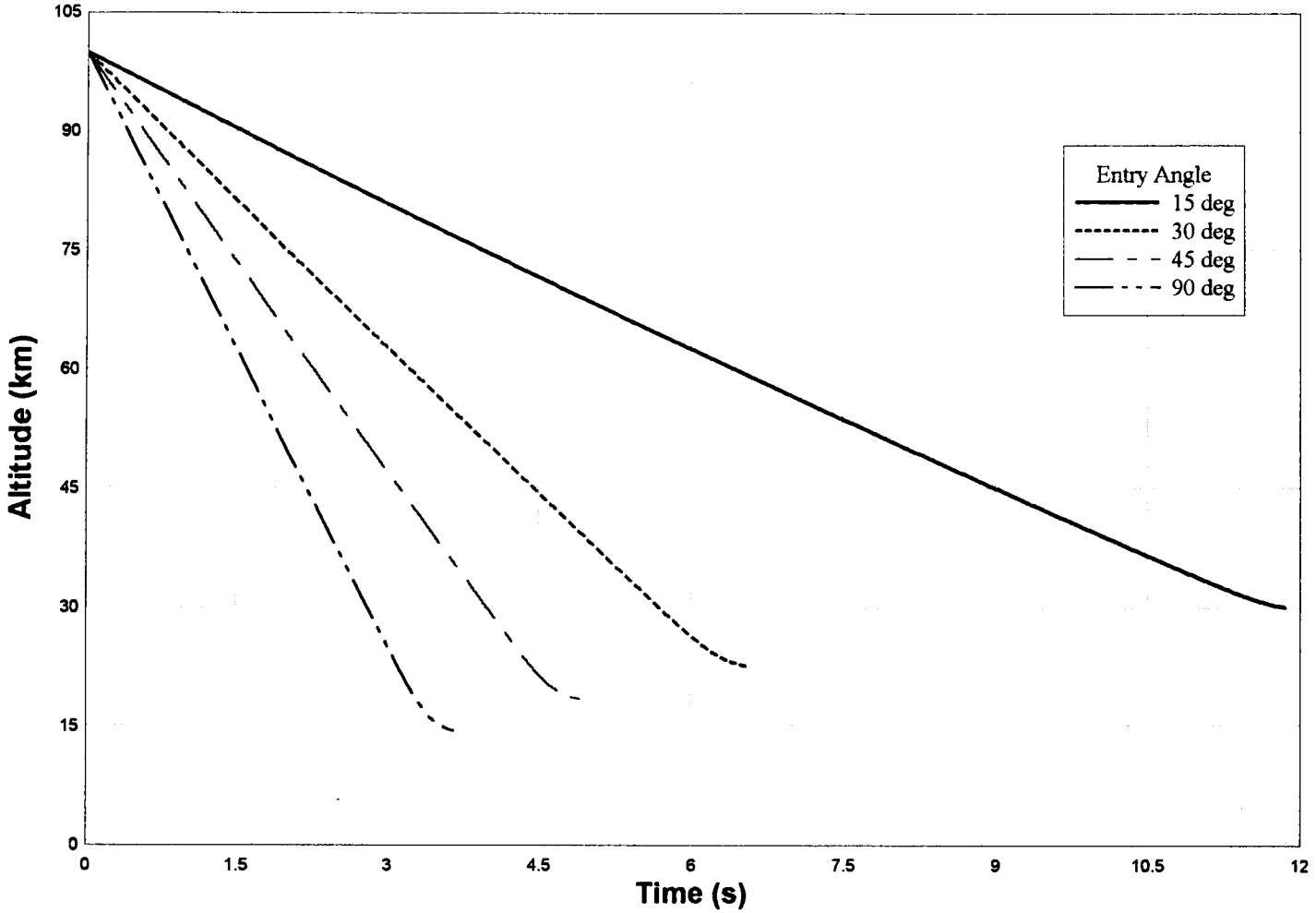


Figure 5.2 - Altitude vs. Time (Weak median body, 10 m radius, 25 km/s)

faster the entry velocity of the body, the faster it approaches the surface. The second figure shows how the rate of fall is affected by the entry angle. The more shallow the entry, the longer it takes the body to reach the surface. The body radius is not shown here because it only affects the fall rate by extending it longer for a larger body.

The velocities of the trajectories are shown in **Figures 5.3** and **5.4. Figure 5.3** shows the variation of velocity with altitude for various body radii. The velocity decreases earlier and at a faster rate for the smaller body. This is expected because the larger body will have more momentum. The various velocities show generally the same trends. However, it is notable that the faster bodies slow at a quicker rate than the slower bodies. This appears to be the result of ablation and the radius change, which will be discussed later in this section.

Figure 5.4 shows the effect that changing the entry angle has on the velocity. As expected, the steeper the angle, the farther the body gets before slowing. Additionally, bodies entering more steeply slow to a lower minimum velocity before their energy is dissipated. This is apparently due to the fact that these bodies travel into the lower, denser atmosphere.

Figures 5.5 and **5.6** show the kinetic energy of the meteor. The smaller bodies lose their kinetic energy earliest because they have less momentum. However, the fastest bodies lose more energy at higher altitudes. This is apparently the result of the faster bodies spreading radially more rapidly, causing an increase in the drag. The

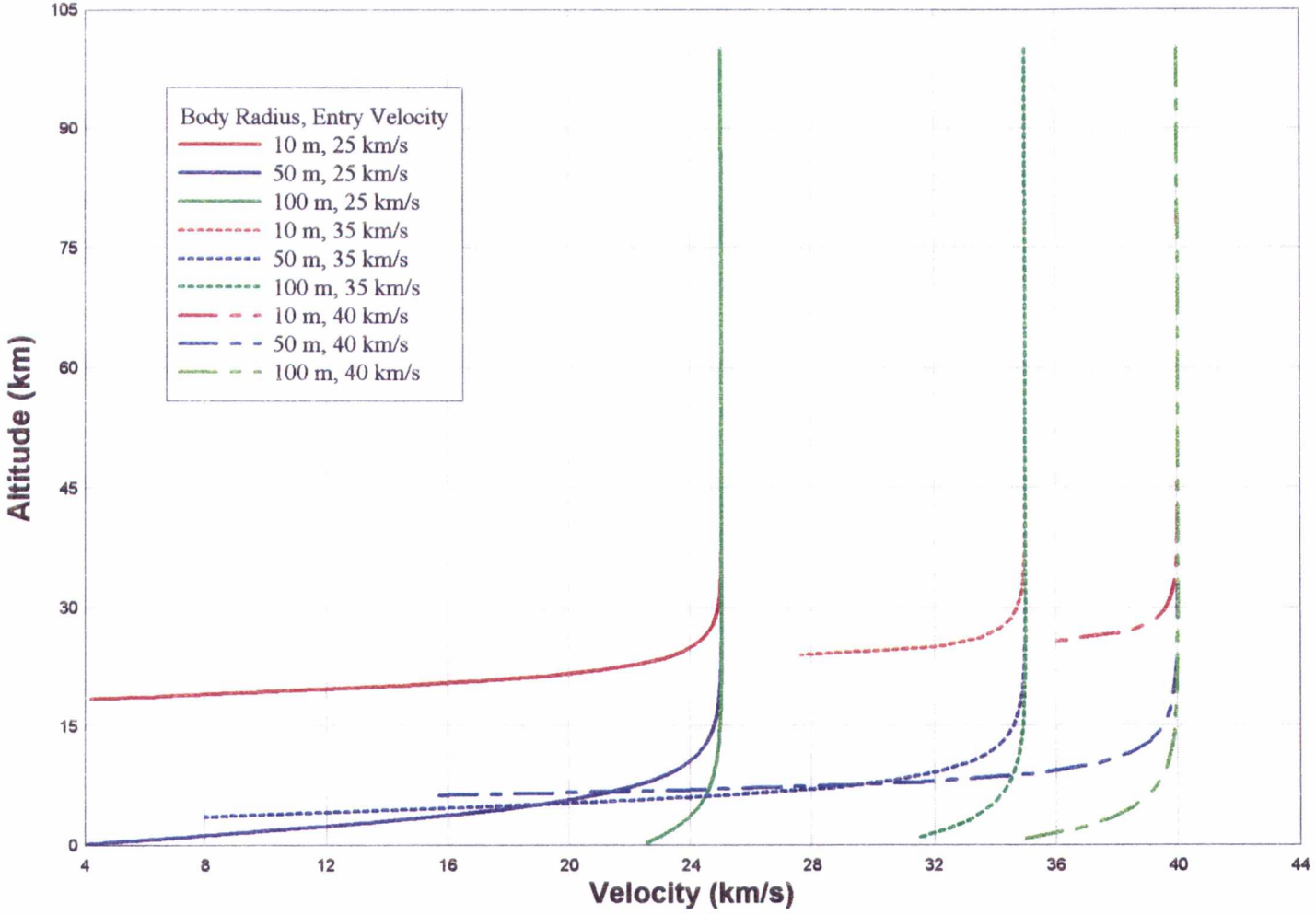


Figure 5.3 - Trajectory Velocity (Weak median body, 45°)

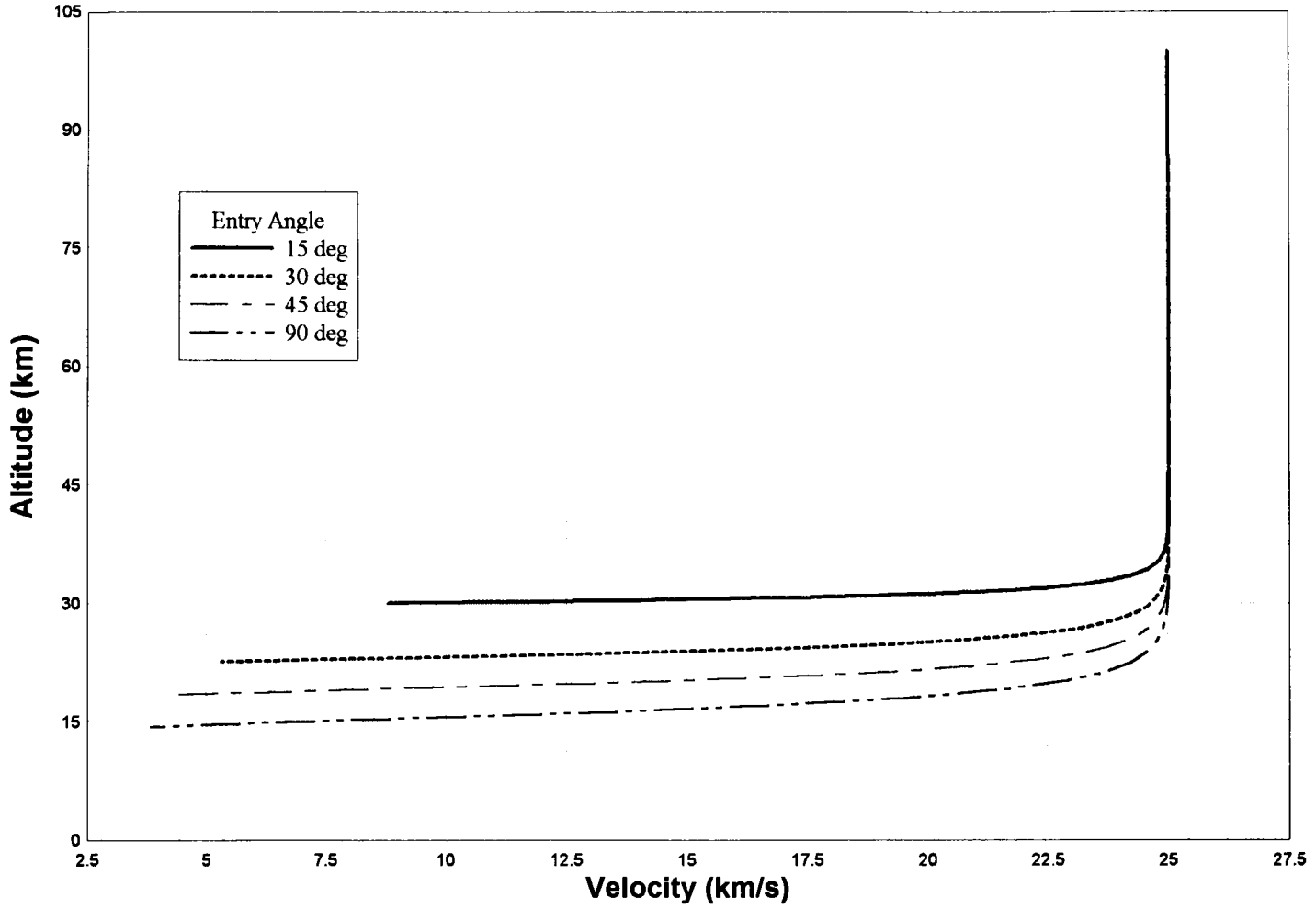


Figure 5.4 - Trajectory Velocity (Weak median body, 10 m radius, 25 km/s)

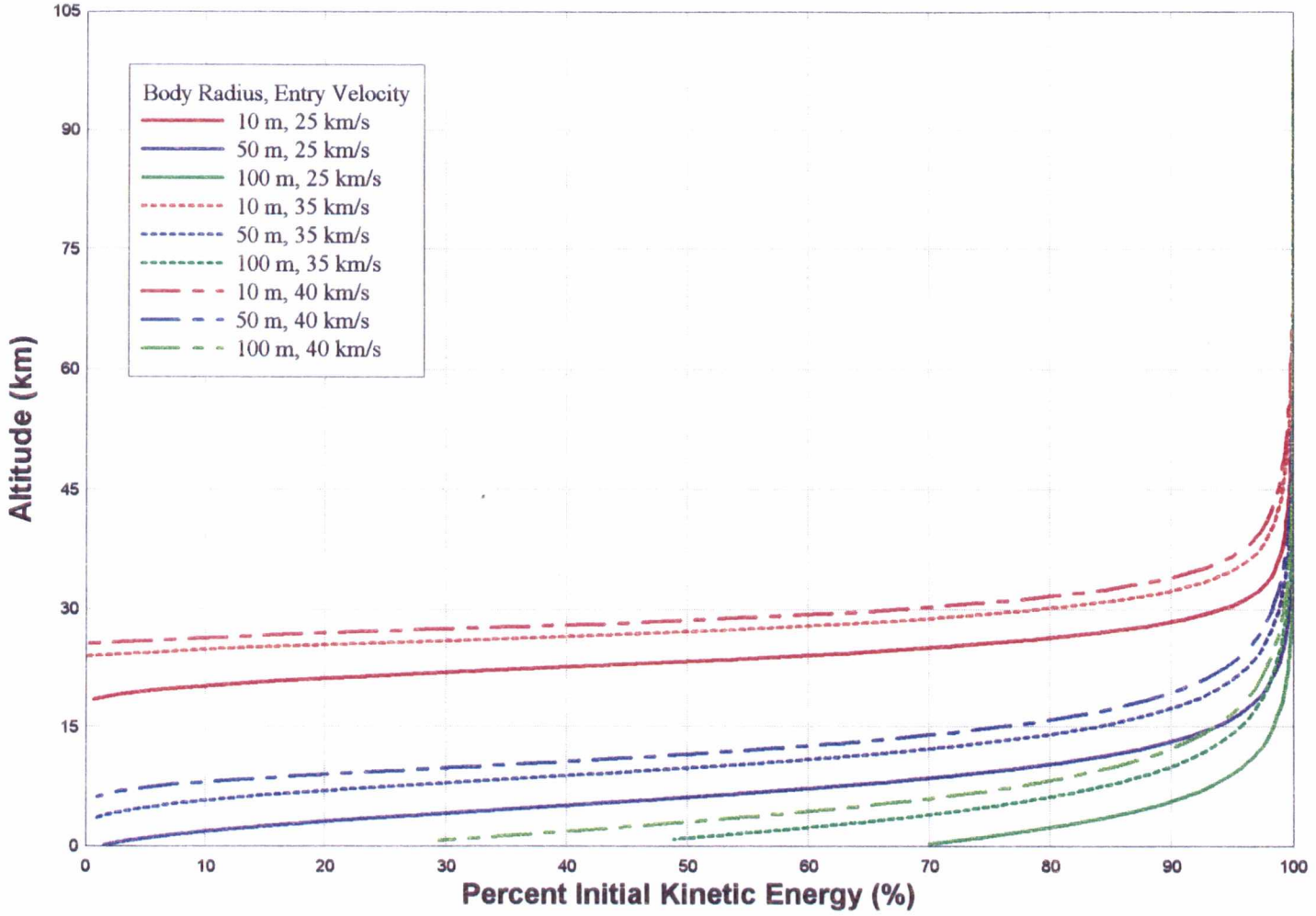


Figure 5.5 - Trajectory Kinetic Energy (Weak median body, 45°)

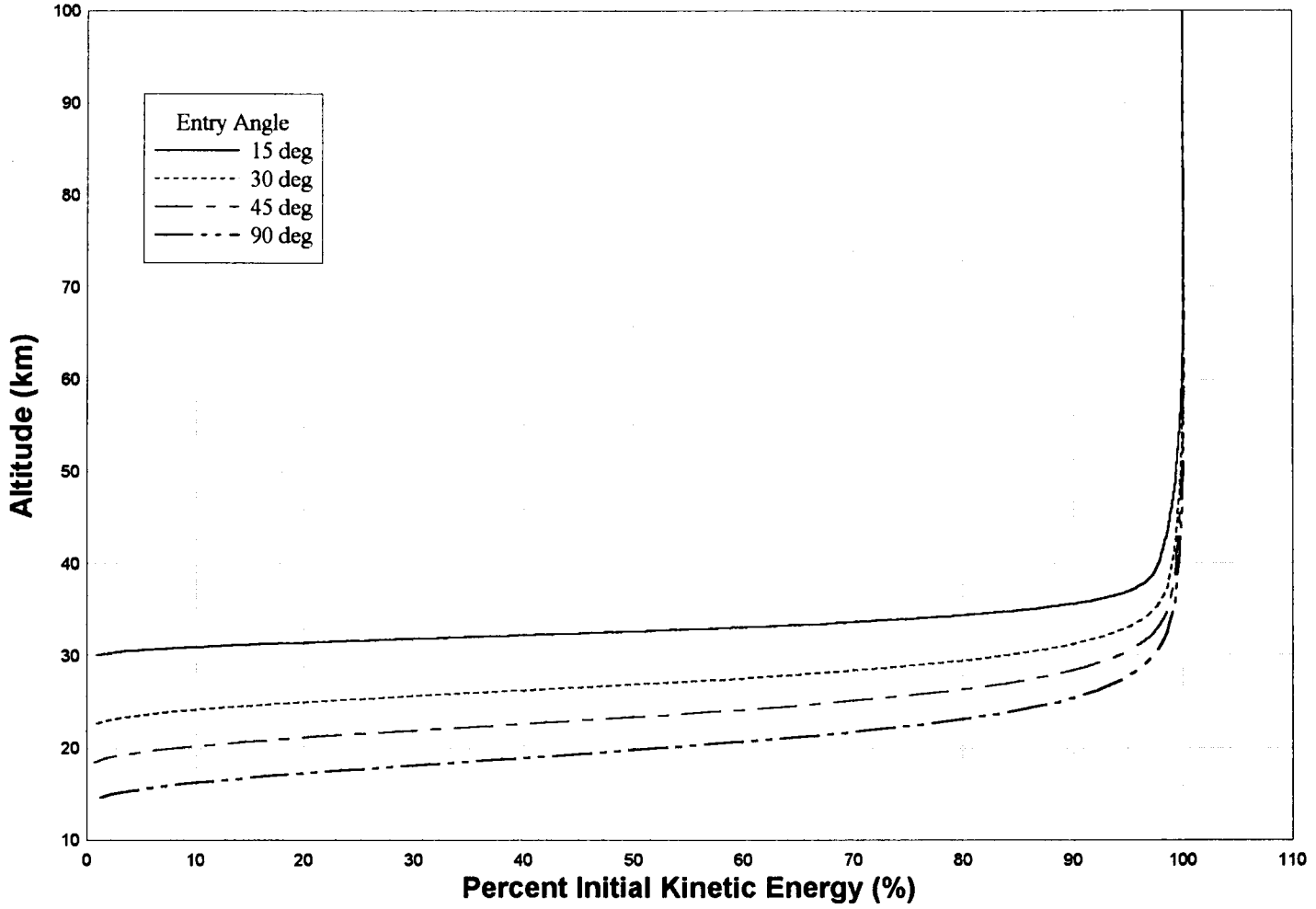


Figure 5.6 - Trajectory Kinetic Energy (Weak median body, 10 m radius, 25 km/s)

variable entry angle graph shows that a body with a shallow entry angle loses kinetic energy at a higher altitude than one with a steep entry angle. This occurs mainly because the shallow entry body is falling towards Earth at a slower rate allowing for more radial spread that results in more drag.

Figures 5.7 and **5.8** show some of the most significant graphs of this work. The graphs show the stagnation temperature throughout the trajectory for several bodies. The lowest peak temperature for this graph is with a 10 meter body entering the atmosphere at 25 km/s and 15°. For this body, the peak temperature is over 36,000 K. Before this work, trajectories with temperatures in excess of 30,000 K could not be analyzed.

As expected, the larger and faster meteors in **Figure 5.7** have the higher peak stagnation temperatures. Also, **Figure 5.8** shows that the steeper the entry angle, the higher the peak stagnation temperature. Larger bodies, faster bodies, and bodies entering at steeper angles all travel further into the dense atmosphere before slowing and this results in an increased heating of the body. It is also notable that the graph of the temperatures in **Figure 5.7** correlates with the graph of the velocities in **Figure 5.3**.

Another velocity dependent property is the stagnation pressure. **Figures 5.9** and **5.10** show the entry parameter effects on the stagnation pressure throughout the trajectory. With trends very similar to the temperature graphs, the maximum stagnation pressures occur on fast, large bodies entering at steep angles.

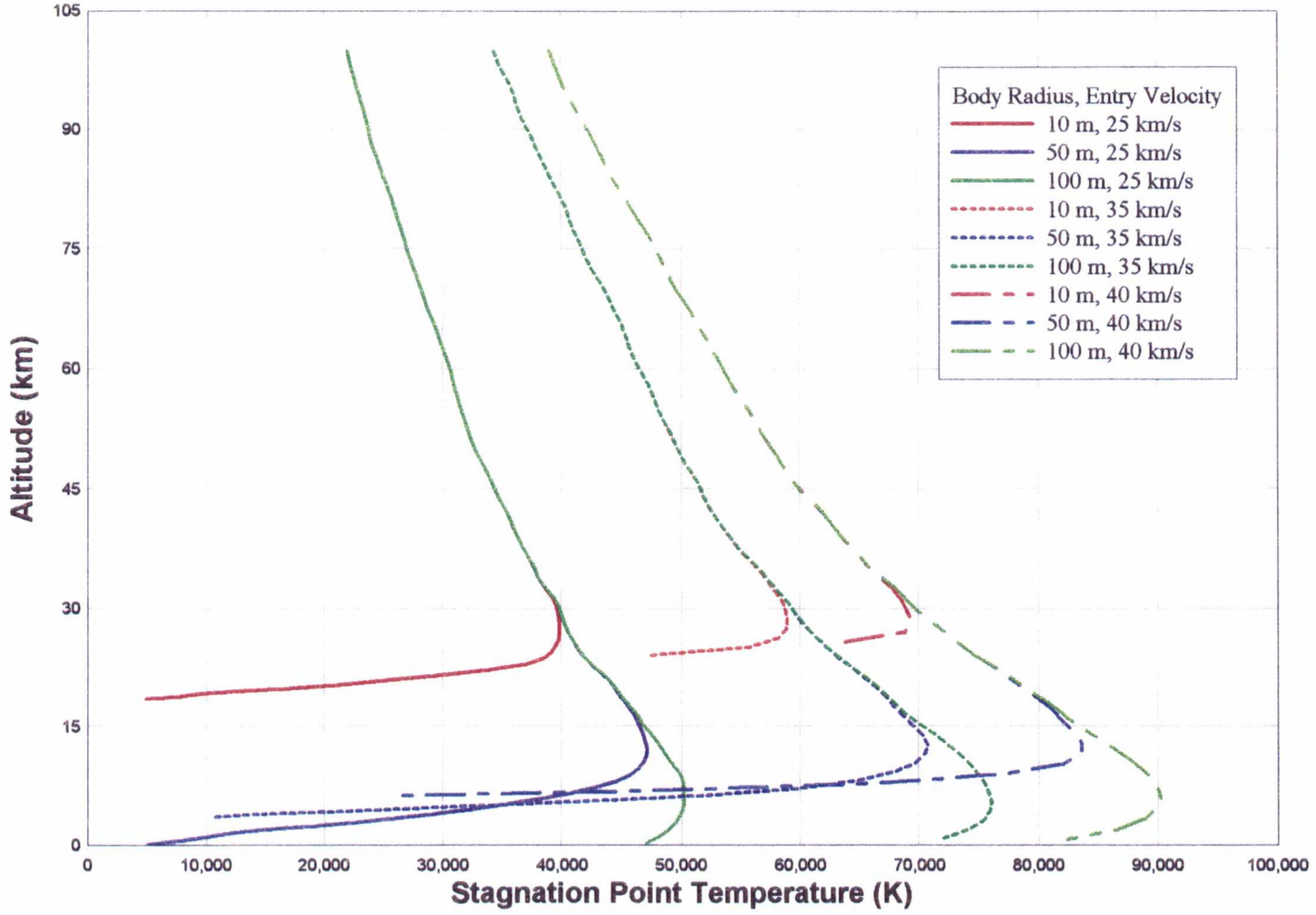


Figure 5.7 - Trajectory Temperature (Weak median body, 45°)

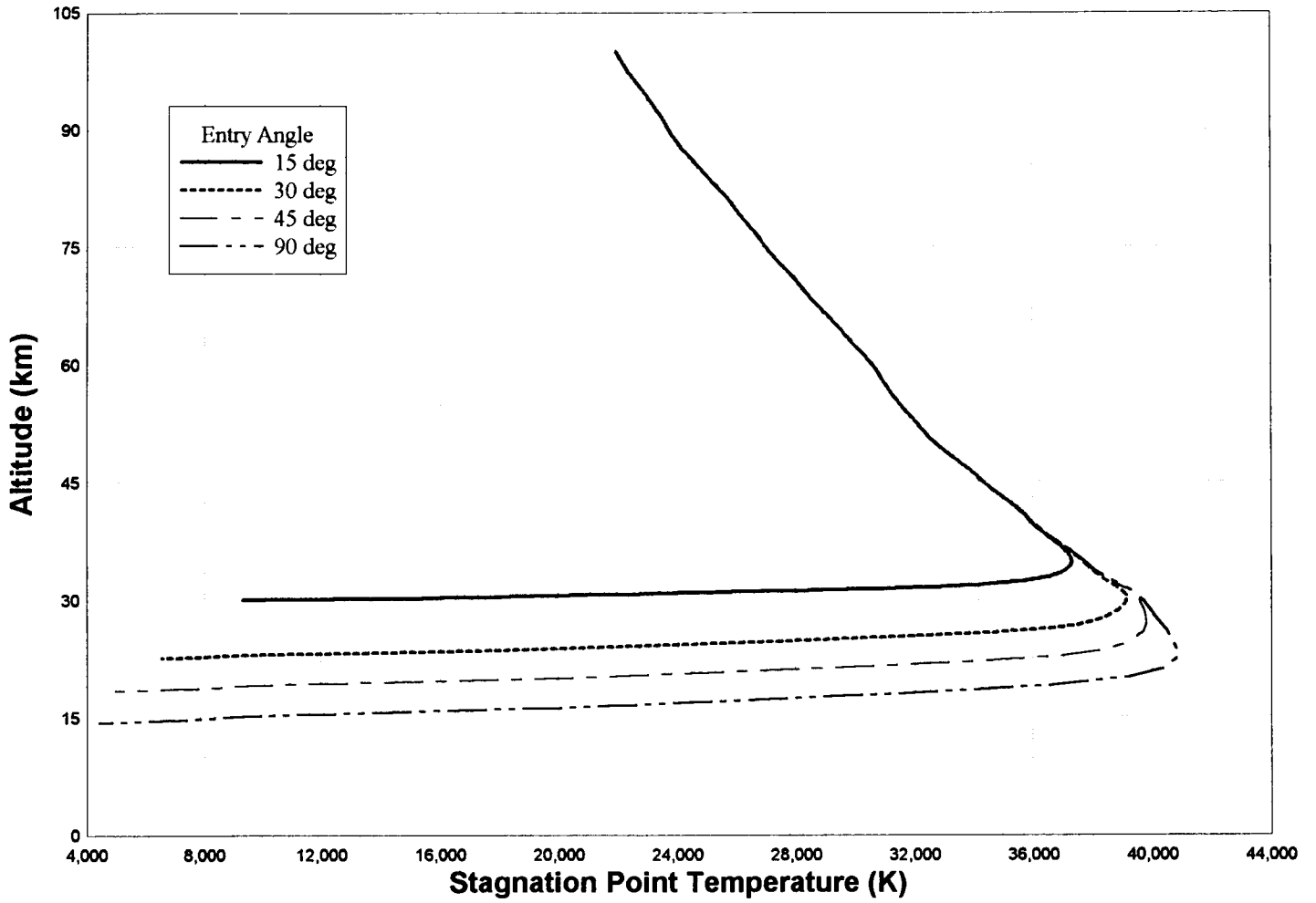


Figure 5.8 - Trajectory Temperature (Weak median body, 10 m radius, 25 km/s)

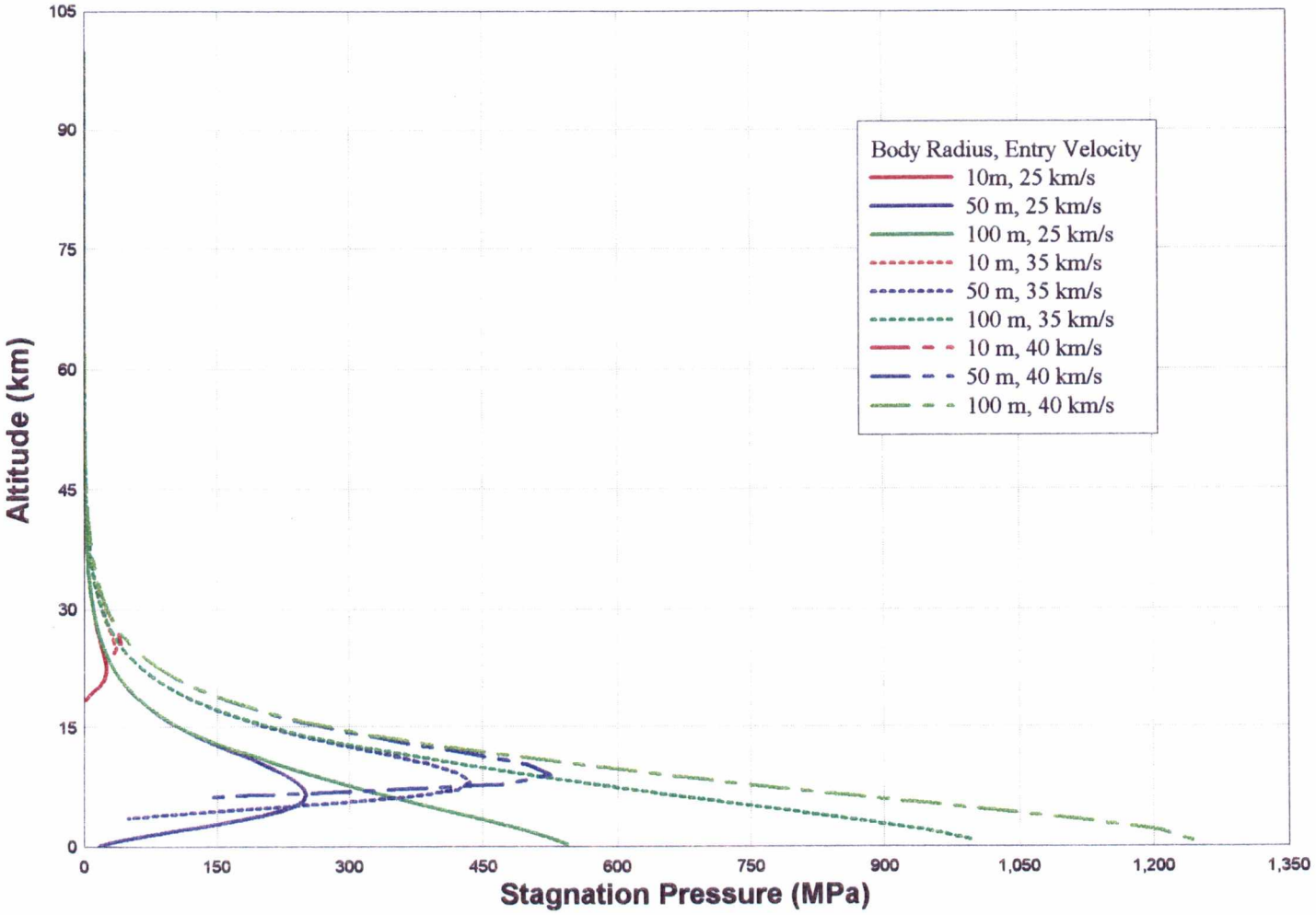


Figure 5.9 - Trajectory Pressure (Weak median body, 45°)

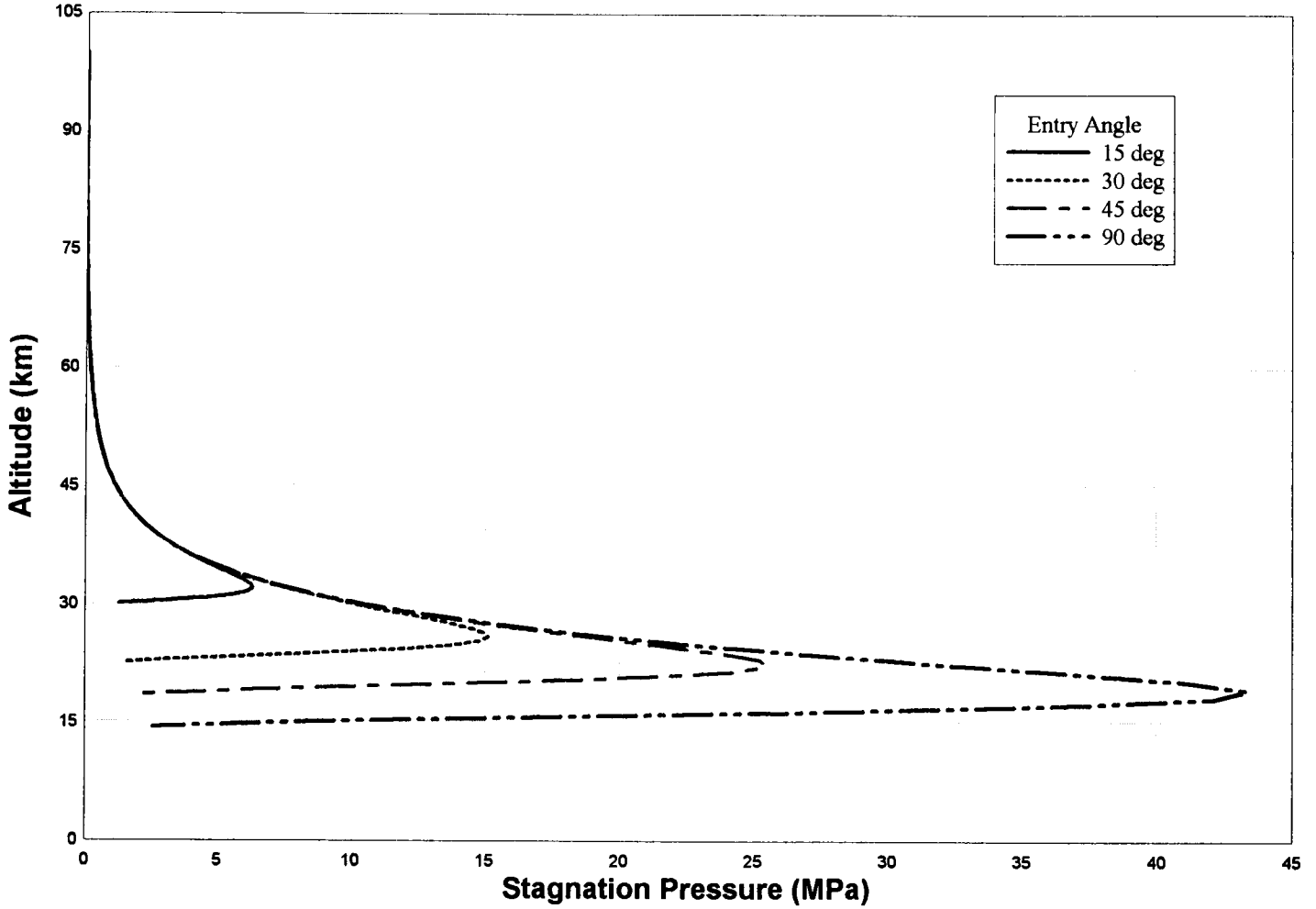


Figure 5.10 - Trajectory Pressure (Weak median body, 10 m radius, 25 km/s)

Figures 5.11 and **5.12** show how the mass of the body is affected by the entry parameters. Mass loss occurs as a result of the body heating to a temperature high enough to begin vaporizing and ablating away material. Therefore, the mass is very dependent on temperature. The 50 meter body in **Figure 5.11** is a good example of how temperature affects mass. The mass stays rather constant until the body reaches a high temperature. As the temperature increases, the amount of mass lost to ablation steadily increases. However, when the body begins to slow and the temperatures decrease, the amount of mass ablated away quickly declines. These trends are very clear for the 50 m body at 25 km/s and 35 km/s. The other bodies either run out of kinetic energy or collide with the Earth before any significant temperature drop occurs. This can be seen in **Figure 5.7**.

The last two graphs, **Figures 5.13** and **5.14**, show the radius of the body throughout the trajectory. A medium body was used for **Figure 5.13**, because the effects were more prominent than in the weak body. However, the trends are very similar.

While ablation does reduce the size of the body and is included in these graphs, it is negligible compared to the change in radius due to fracturing. In **Figure 5.13**, the radius change due to ablation is not visible because of the dominance of the radius growth. Because of the lower momentum of the smaller bodies, they will tend to crush at a higher altitude. Therefore, in **Figure 5.13** the radius of the smaller bodies can be seen to surpass that of the larger bodies. The breakup that causes the radius

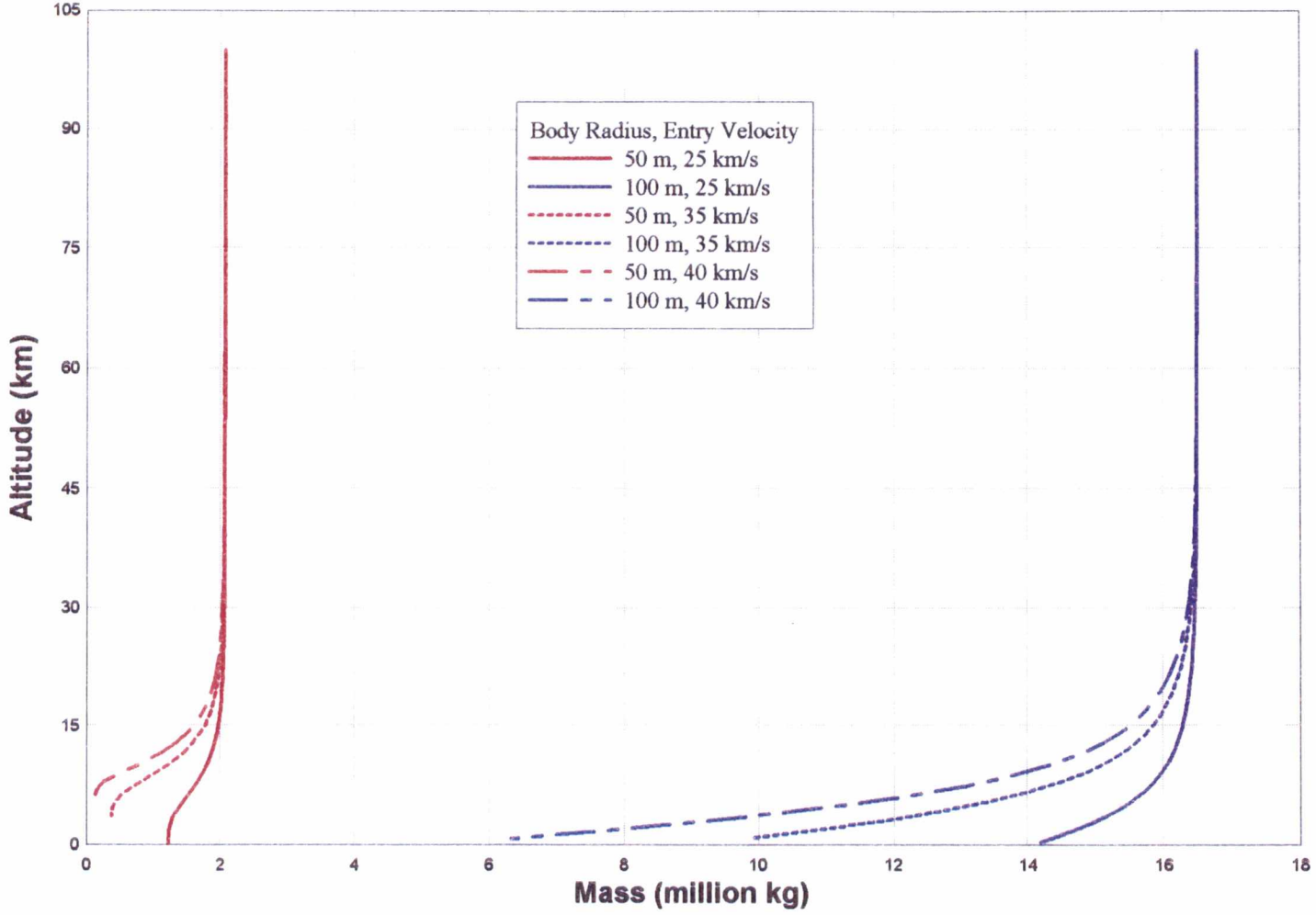


Figure 5.11 - Trajectory Mass (Weak median body, 45°)

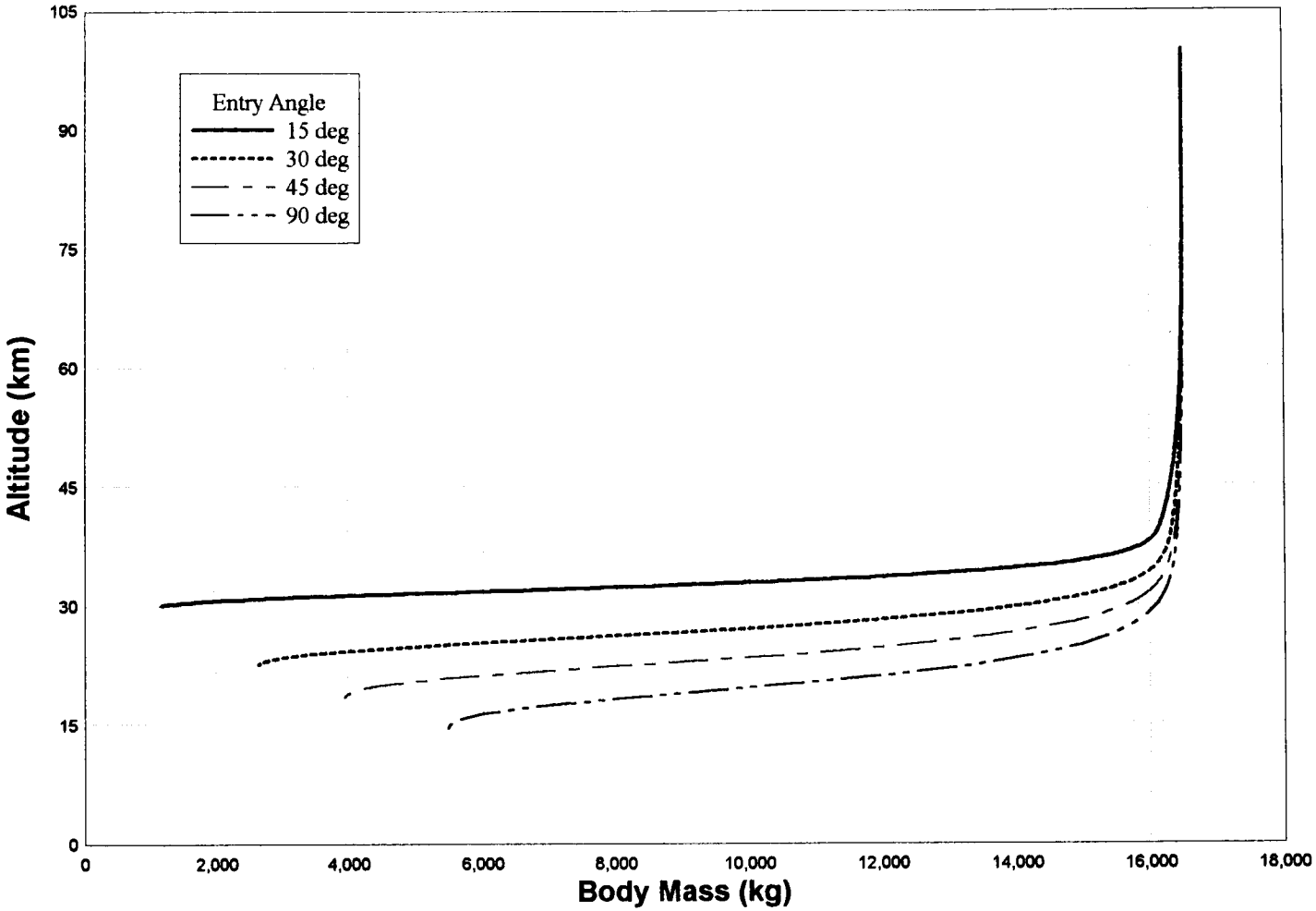


Figure 5.12 - Trajectory Mass (Weak median body, 10 m radius, 25 km/s)

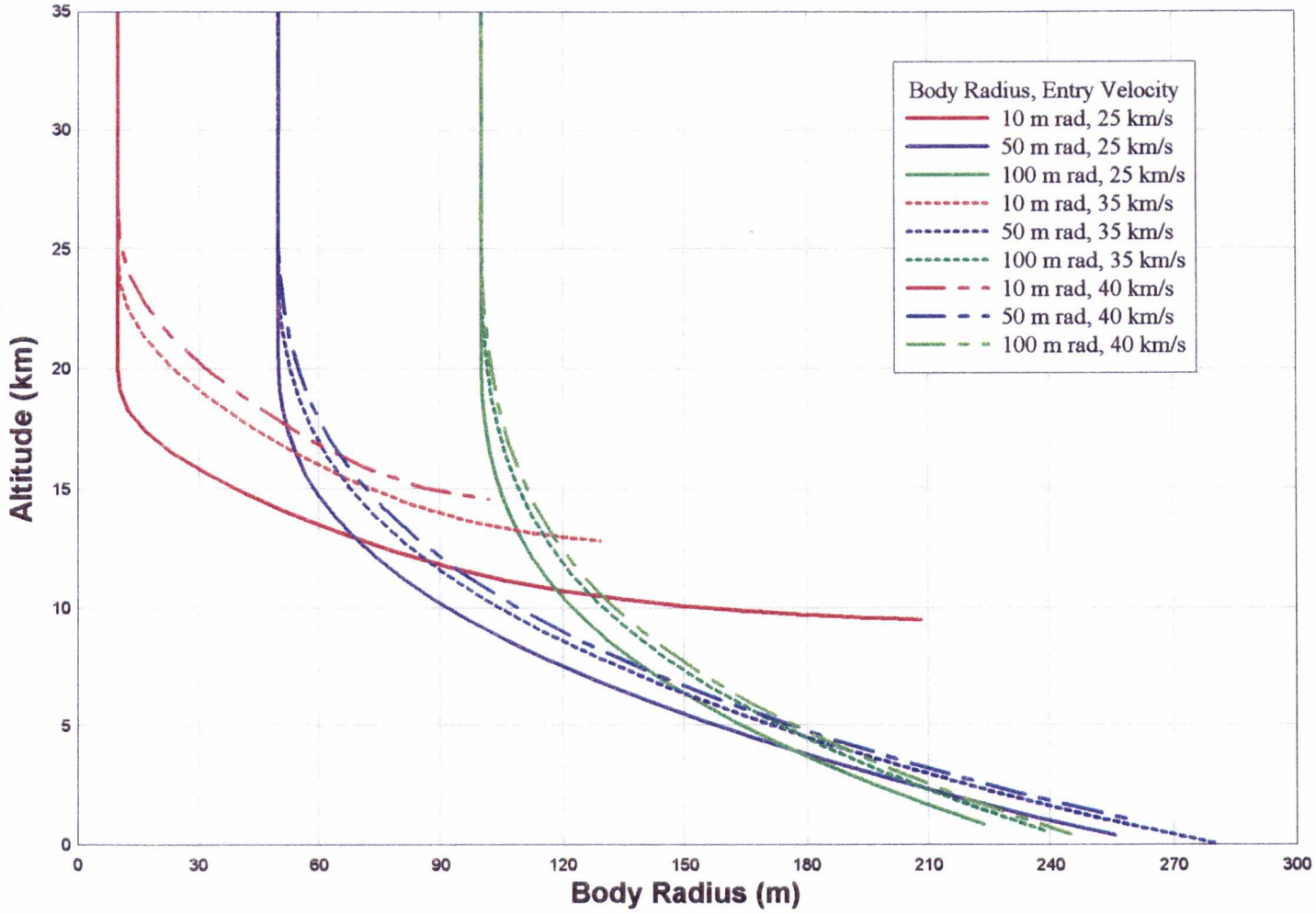


Figure 5.13 - Trajectory Radius (Medium median body, 45°)

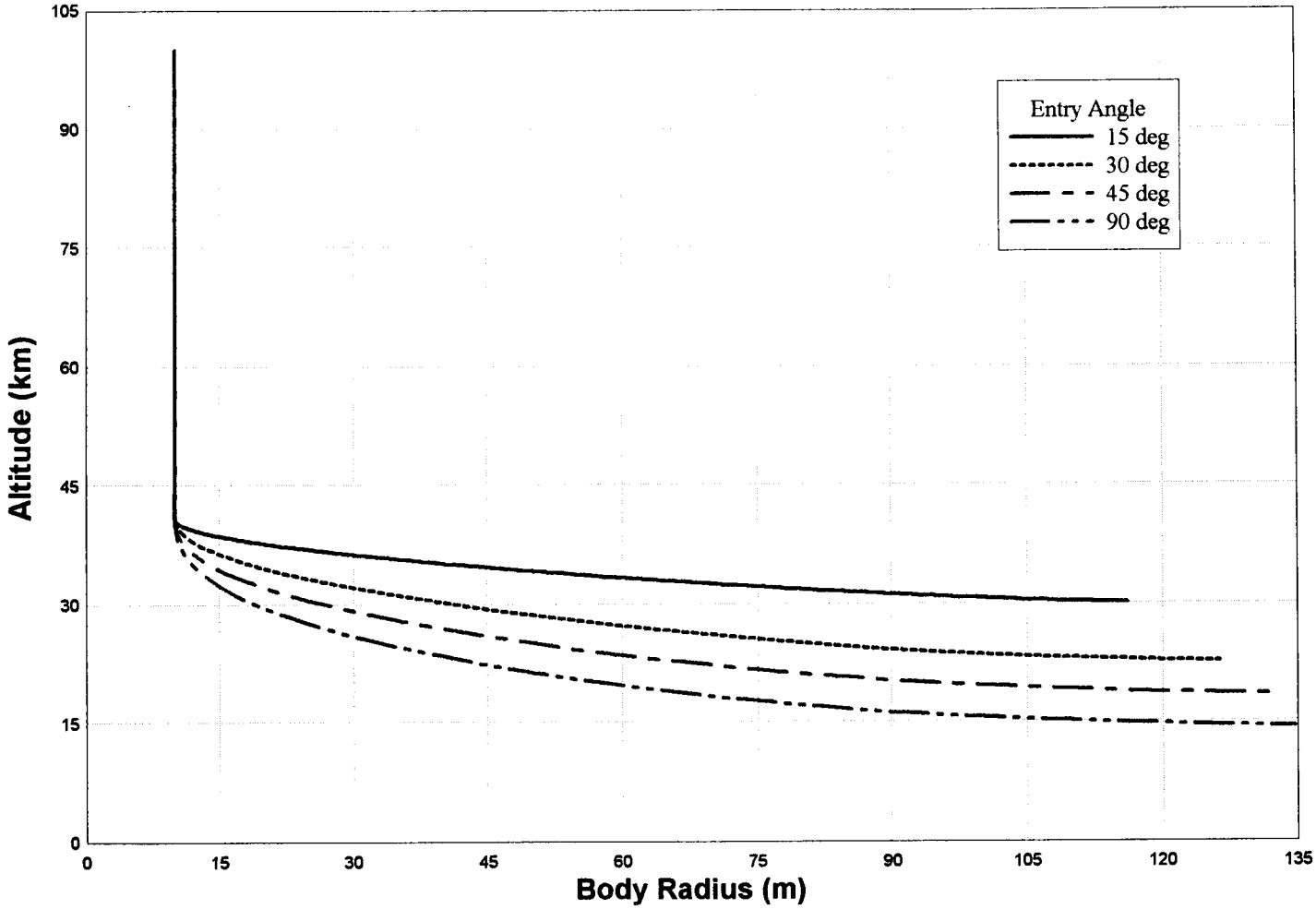


Figure 5.14 - Trajectory Radius (Weak median body, 10 m radius, 25 km/s)

increase of the smaller body does not necessarily occur earlier, just at a higher altitude.

Figure 5.14 shows that a body traveling at a steeper angle will produce a debris cloud with a smaller radius at a given altitude. However, this increase occurs at a lower altitude because the steeper body is falling towards Earth at a faster rate.

5.3 Trajectory Analysis: Body Parameters

In addition to the entry parameters considered in Section 5.2, three body parameters must be considered. These are body density, heat of ablation, and yield strength. These three parameters generally define the properties of a meteor necessary for trajectory analysis.

The most common body used in Section 5.2 was a weak median body. This same body will be used in this section with variations on the three body parameters. The entry parameters used were a body radius of 50 meters, an entry velocity of 35 km/s, and an entry angle of 45°. The values used for the weak body parameters are given in **Table 5.1**.

Figures 5.15 through **5.21** show how changes in the density affect the meteor trajectory. **Figure 5.15** shows that a denser body, which would also be a heavier body, maintains a higher velocity.

Likewise, **Figure 5.16** shows that the denser body maintains more kinetic energy. As noted in section 5.2, the stagnation point temperature and stagnation pressure are

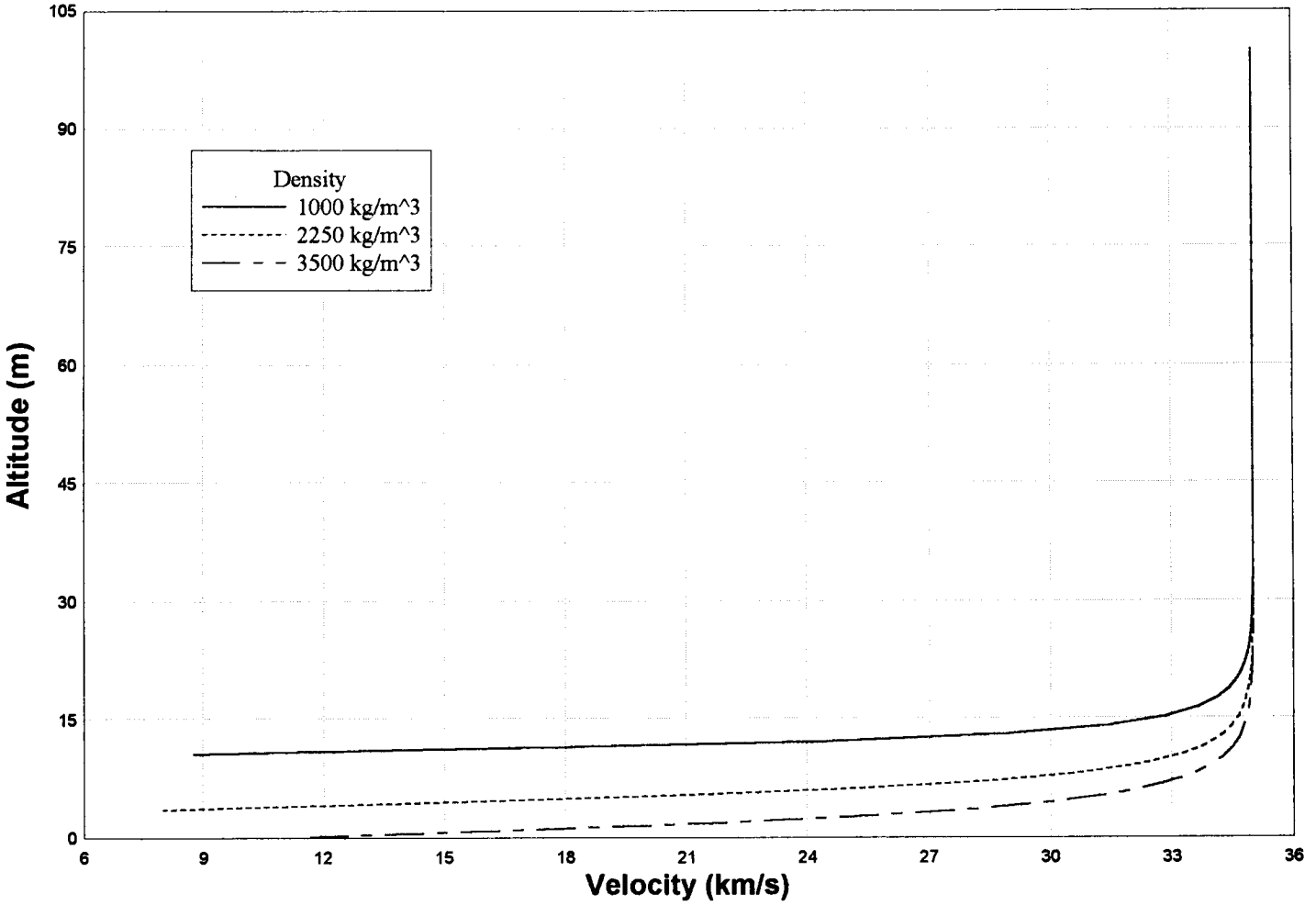


Figure 5.15 - Trajectory Velocity vs. Density (Weak median body, 50 m radius, 35 km/s, 45°)

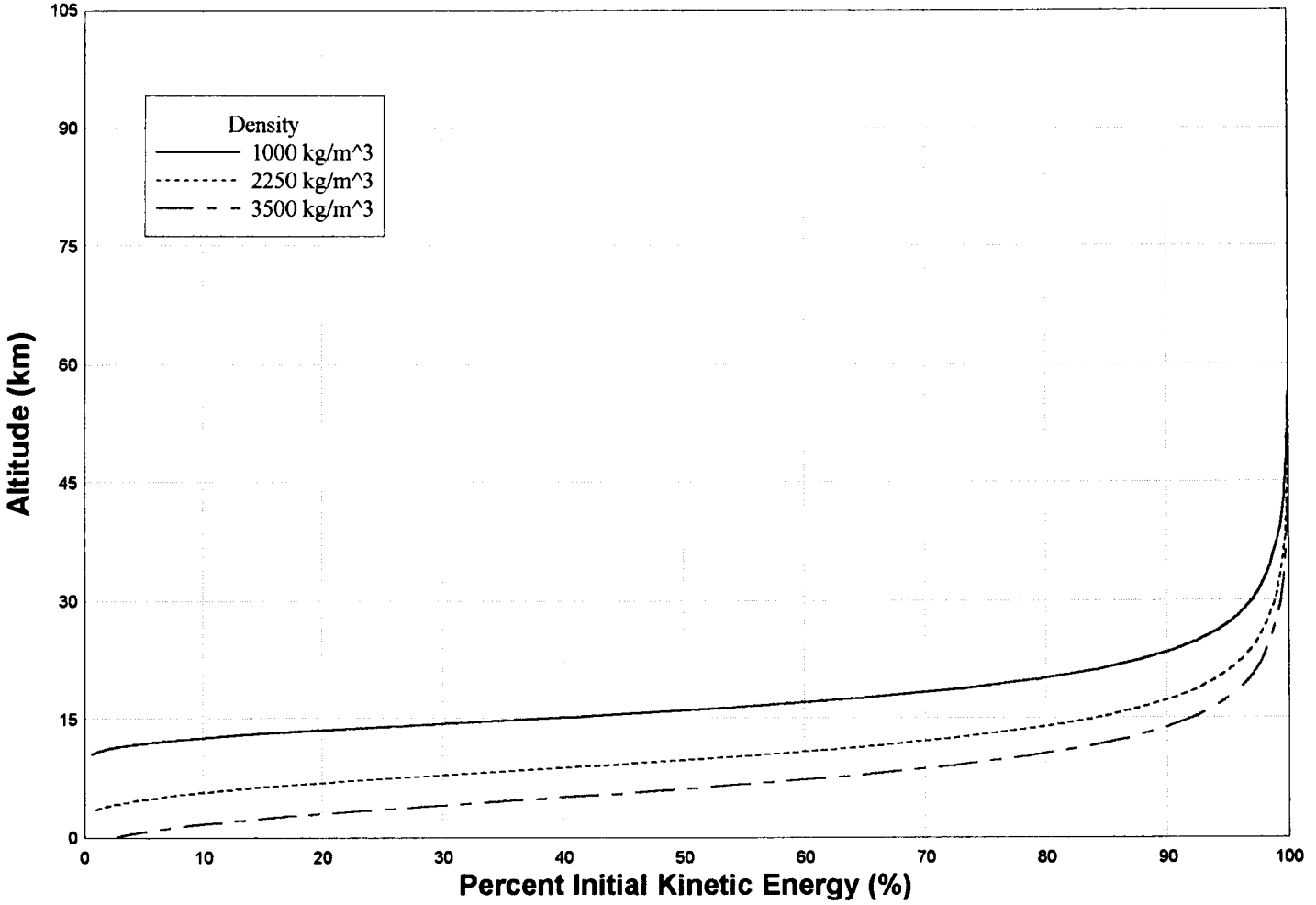


Figure 5.16 - Trajectory Kinetic Energy vs. Density (Weak median body, 50 m radius, 35 km/s, 45°)

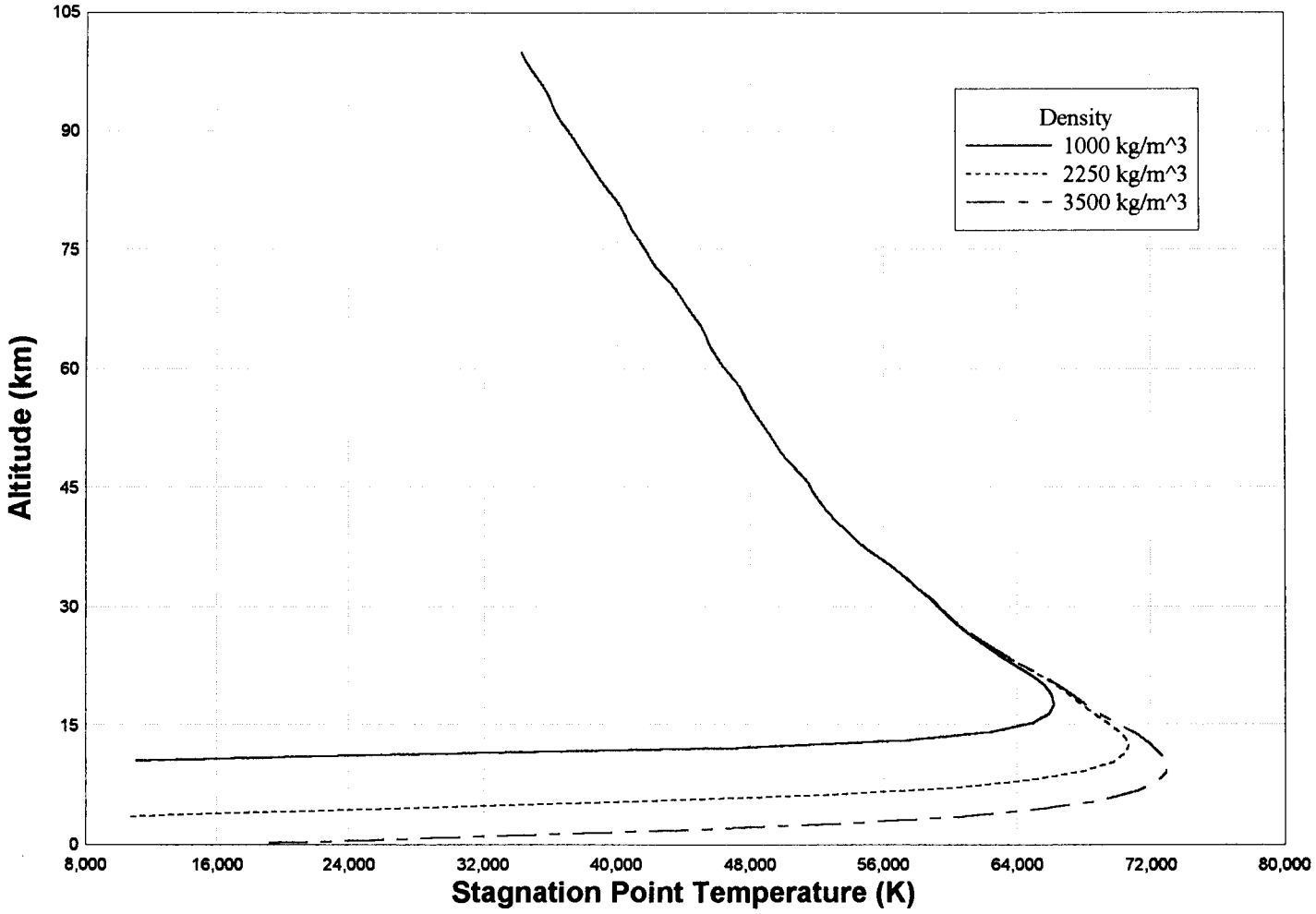


Figure 5.17 - Trajectory Temperature vs. Density (Weak median body, 50 m radius, 35 km/s, 45°)

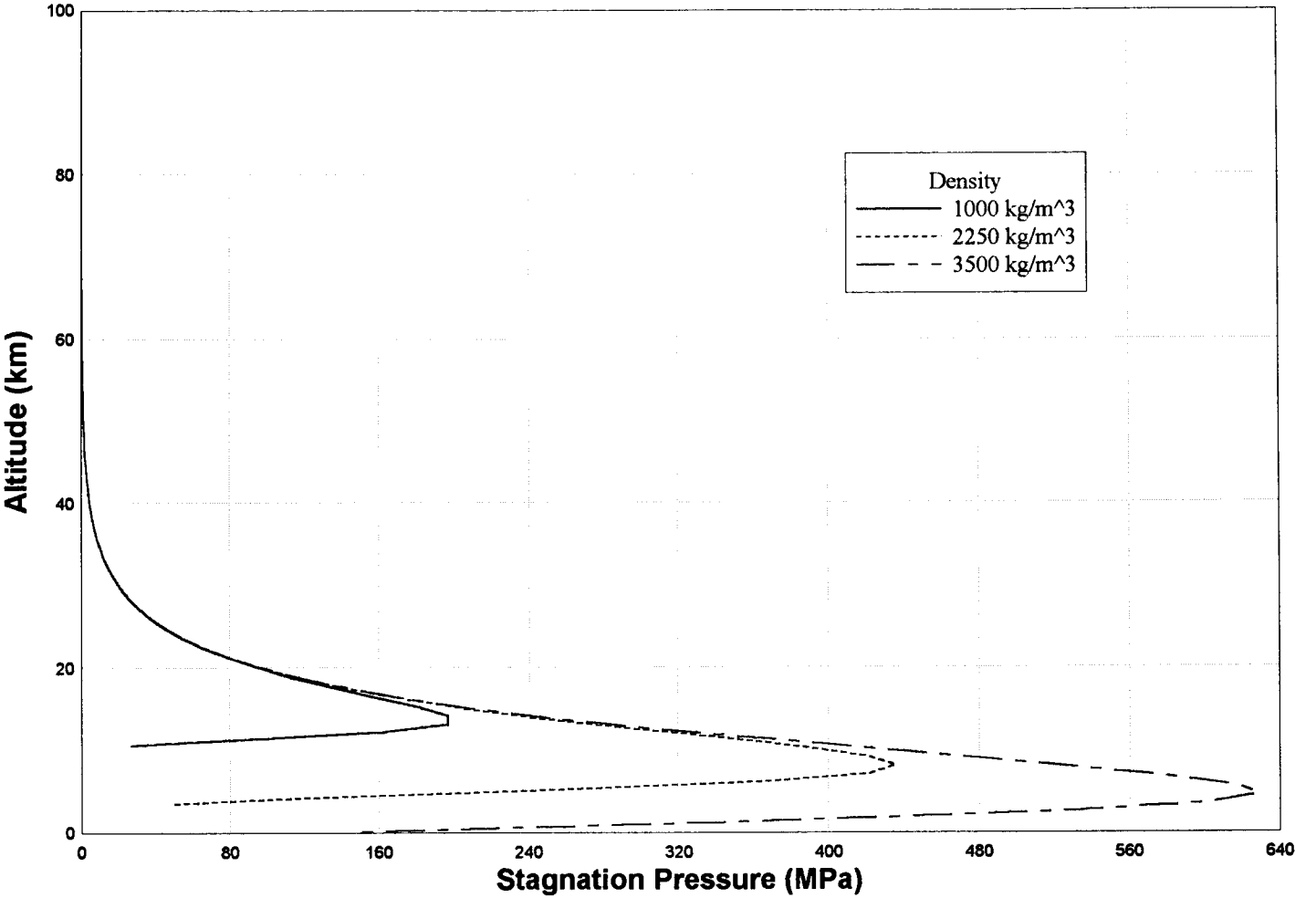


Figure 5.18 - Trajectory Pressure vs. Density (Weak median body, 50 m radius, 35 km/s, 45°)

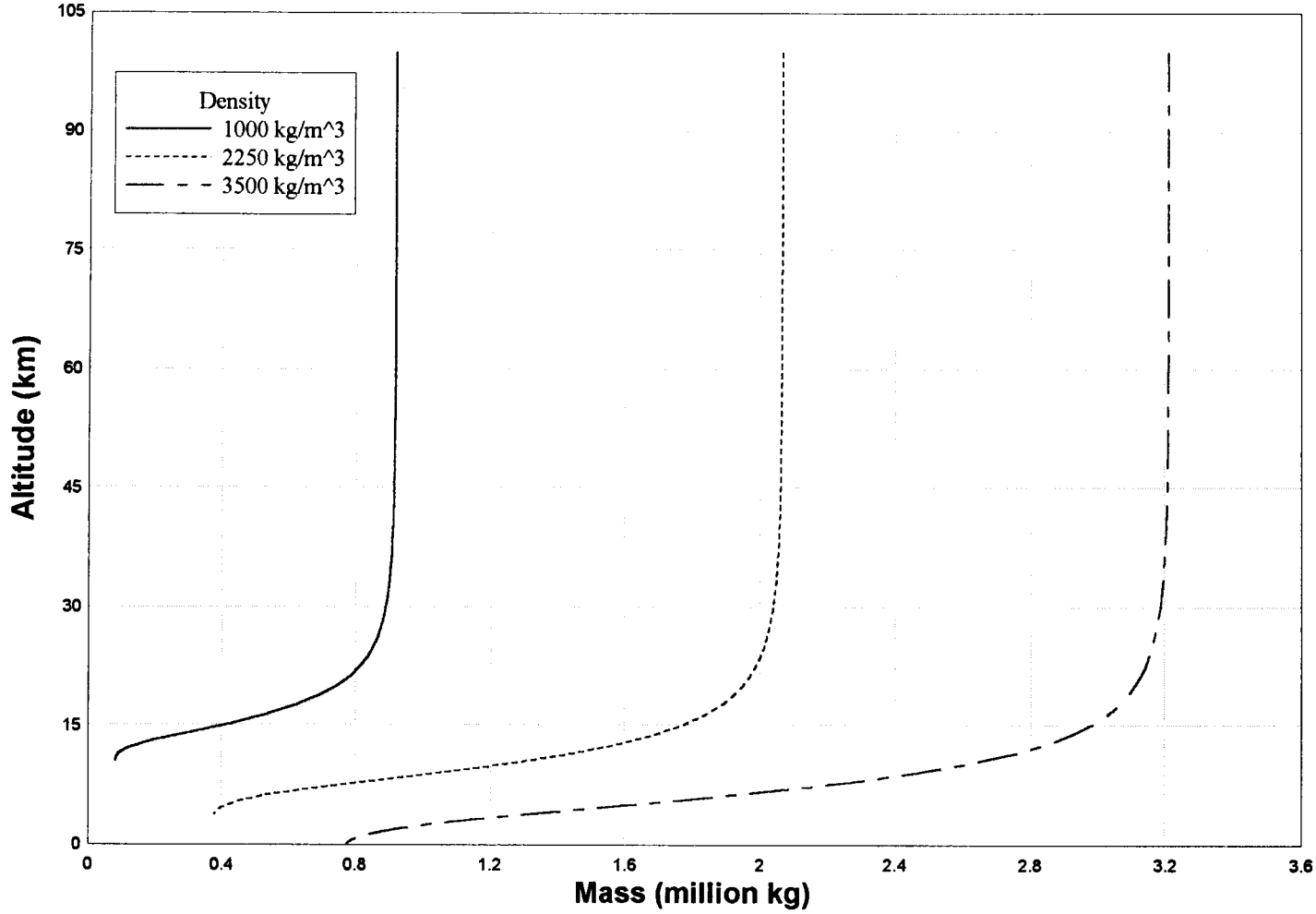


Figure 5.19 - Trajectory Mass vs. Density (Weak median body, 50 m radius, 35 km/s, 45°)

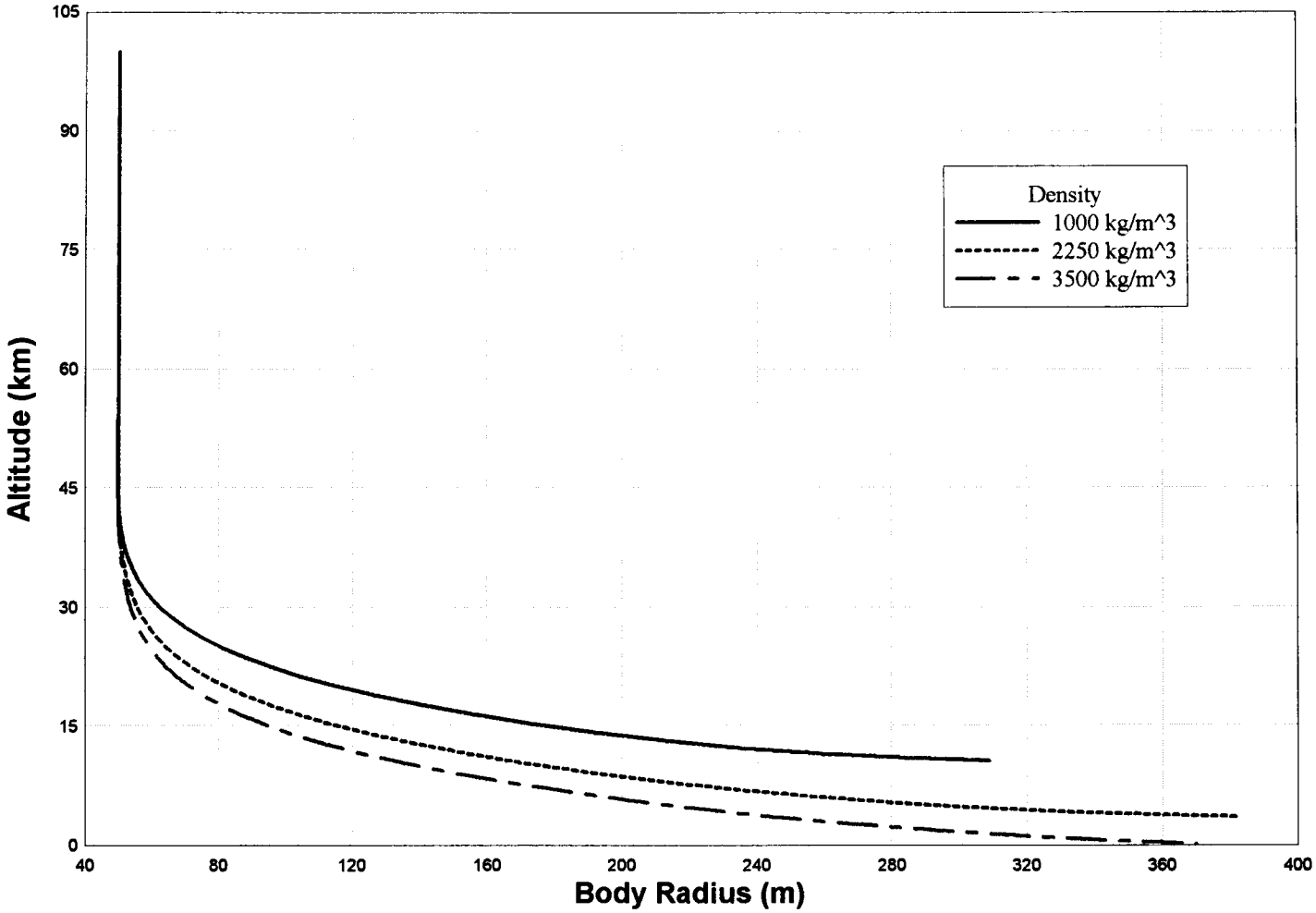


Figure 5.20 - Trajectory Radius vs. Density (Weak median body, 50 m radius, 35 km/s, 45°)

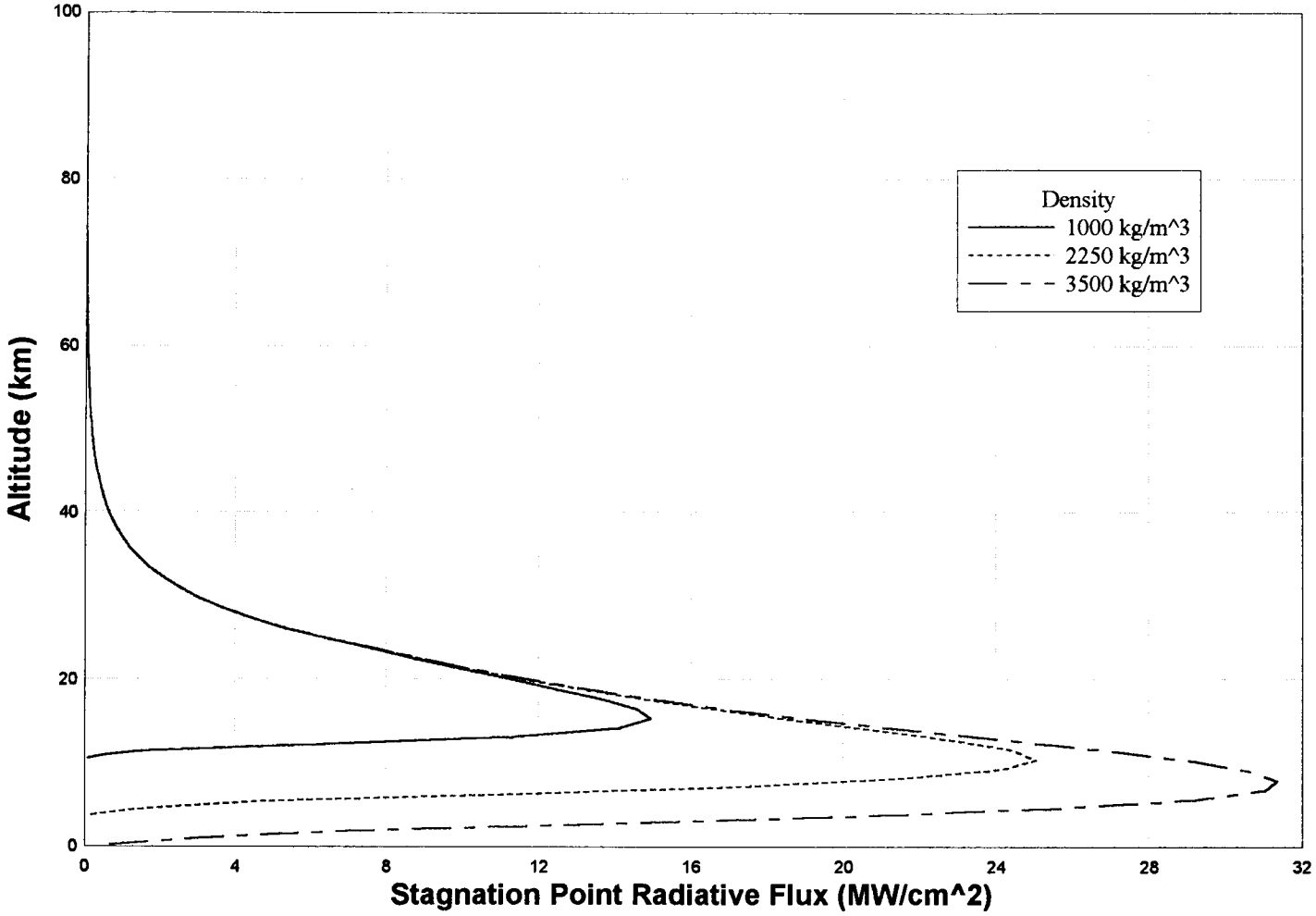


Figure 5.21 - Trajectory Heat Flux vs. Density (Weak median body, 50 m radius, 35 km/s, 45°)

very dependent on the velocity. Therefore, as seen in **Figures 5.17** and **5.18**, the more dense bodies result in a higher peak stagnation point temperature and peak stagnation pressure. **Figure 5.19** shows the trajectory mass for the various body densities. Also, **Figure 5.20** shows that the denser body expands at a lower altitude. Finally, **Figure 5.21** shows the radiative heat flux into the body at the stagnation point for the various densities. Since the denser body has a higher peak temperature, the heat flux is naturally greater.

As is shown in **Figures 5.22** through **5.27**, the effects of varying the heat of ablation are not as pronounced. The heat of ablation defines the amount of energy required to ablate a certain amount of material. Therefore, a higher heat of ablation would result in a reduced mass loss. This is exactly the result seen in **Figure 5.22**. This higher mass also affects other parameters of the trajectory. In **Figure 5.23**, the kinetic energy loss is reduced. **Figure 5.24** shows that the velocity is slightly higher at equal altitudes. **Figure 5.25** shows a slightly higher maintained stagnation point temperature, and **Figure 5.26** shows an increased peak stagnation pressure. The last graph of this series shows a slightly higher radiative heat flux to the stagnation point. All of these are a result of the increased mass due to the larger heat of ablation.

The last body parameter considered is the yield strength. The yield strength is used to determine when the body will begin to breakup. Varying this parameter ultimately had little effect on the trajectory. **Figures 5.28** through **5.31** show some of

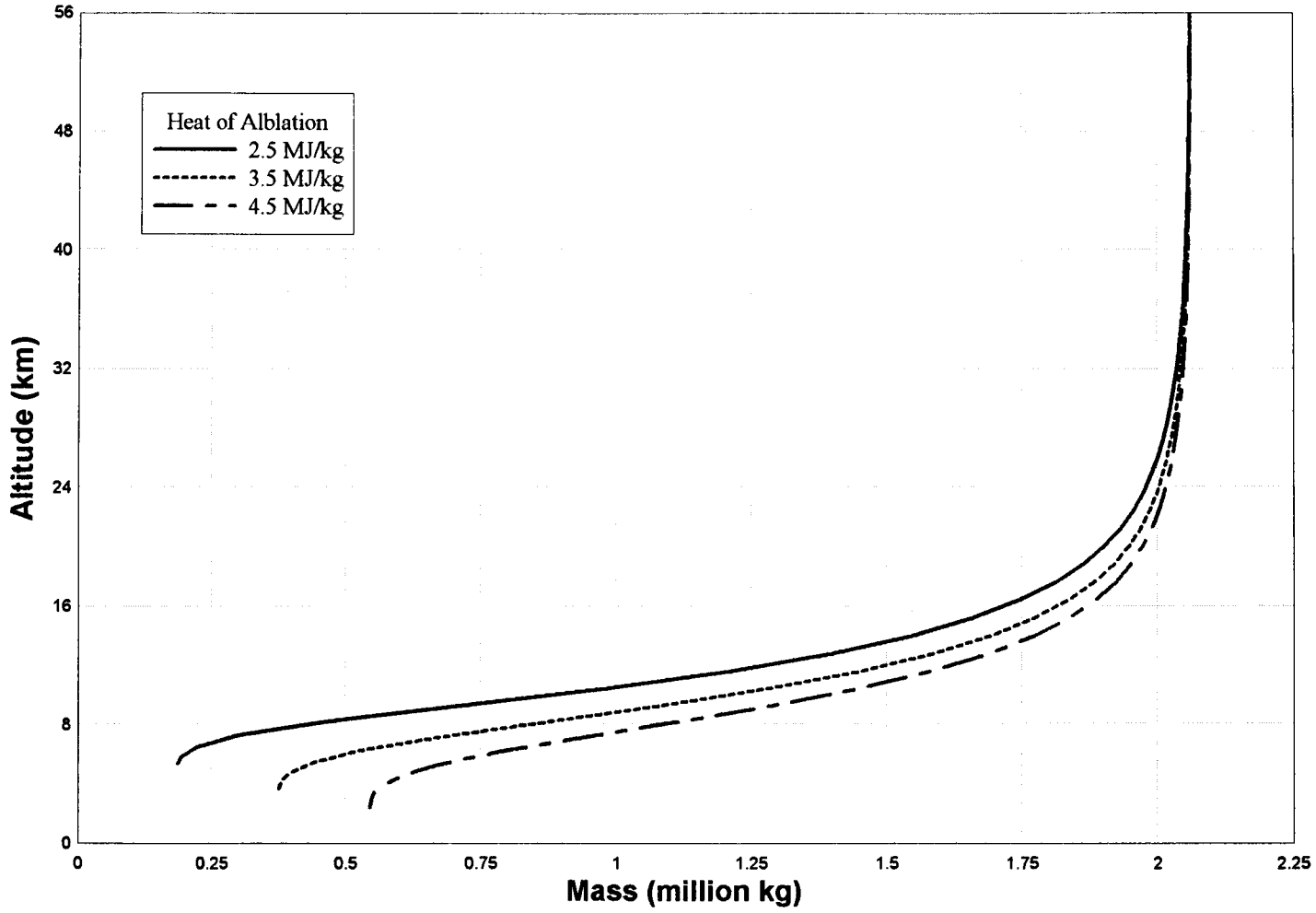


Figure 5.22 - Trajectory Mass vs. Heat of Ablation (Weak median body, 50 m radius, 35 km/s, 45°)

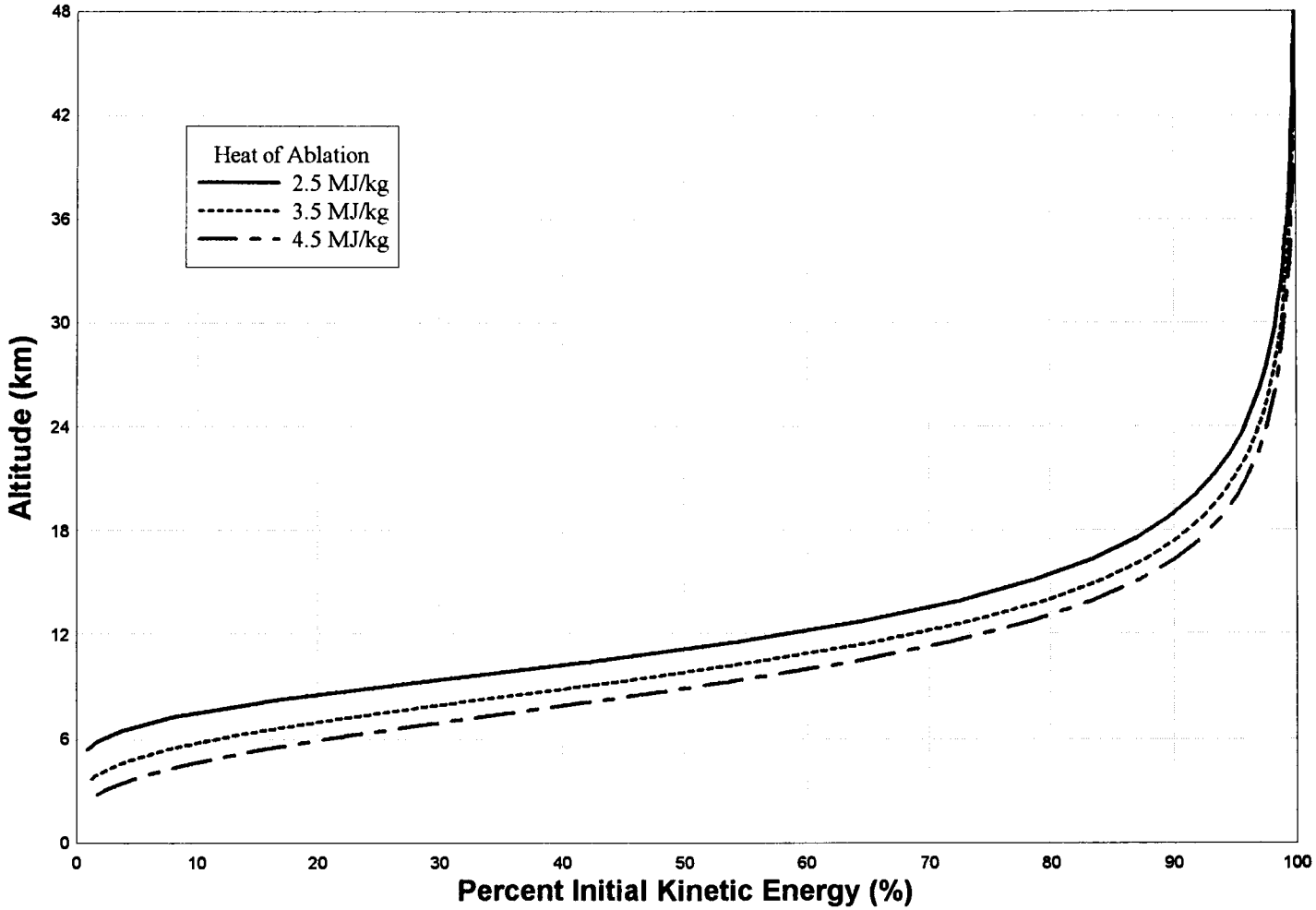


Figure 5.23 - Trajectory Kinetic Energy vs. Heat of Ablation (Weak median body, 50 m radius, 35 km/s, 45°)

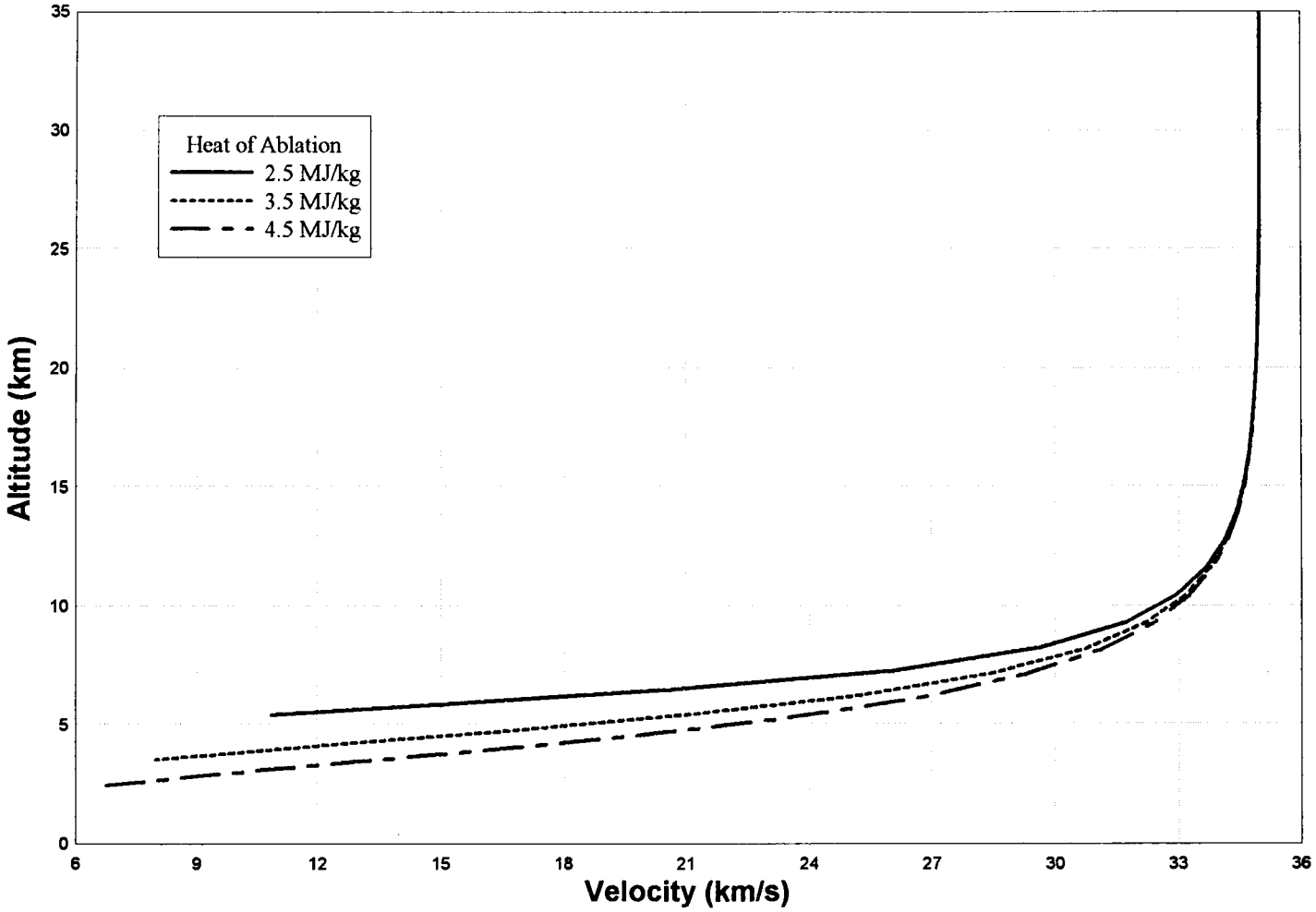


Figure 5.24 - Trajectory Velocity vs. Heat of Ablation (Weak median body, 50 m radius, 35 km/s, 45°)

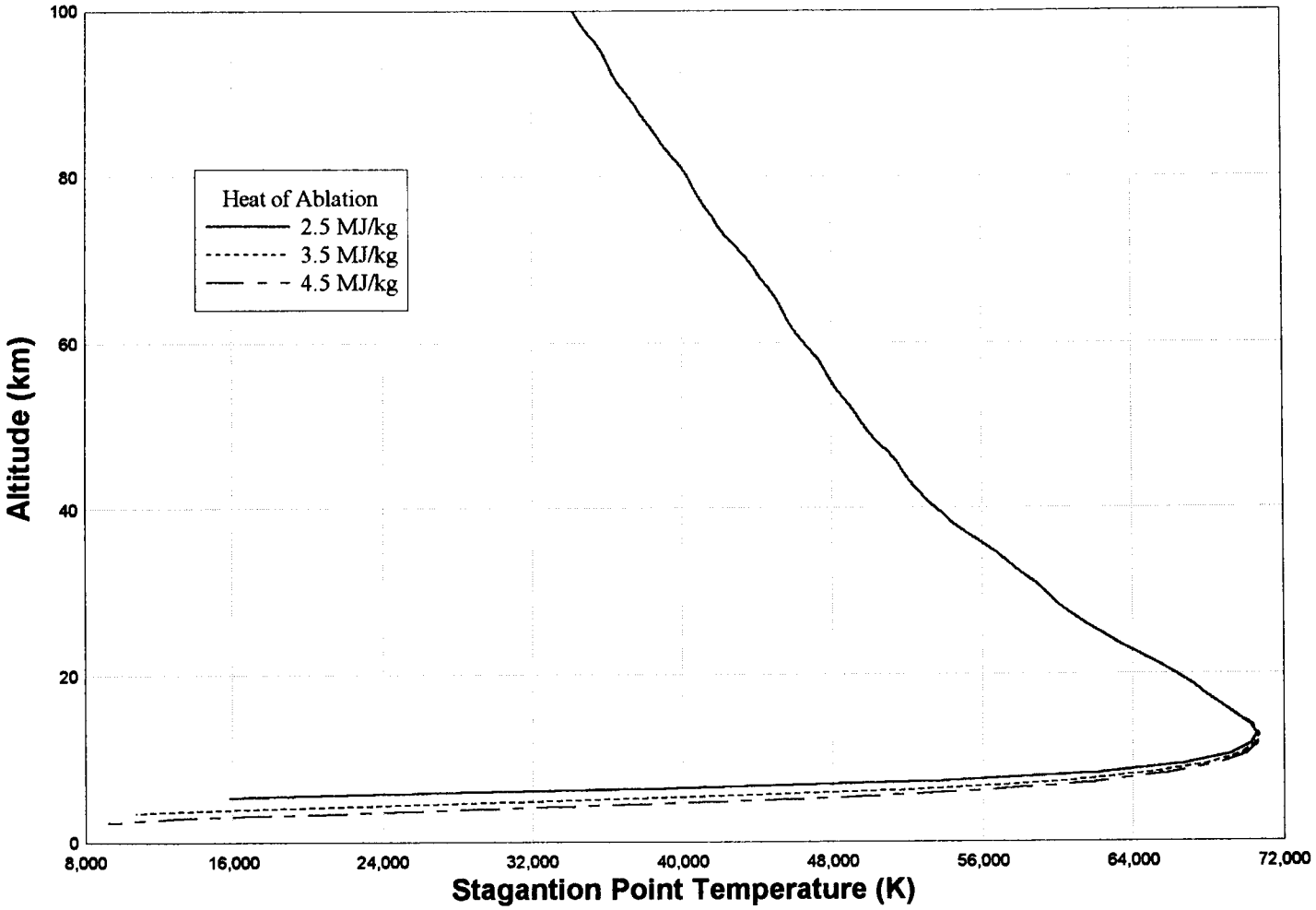


Figure 5.25 - Trajectory Temperature vs. Heat of Ablation (Weak median body, 50 m radius, 35 km/s, 45°)

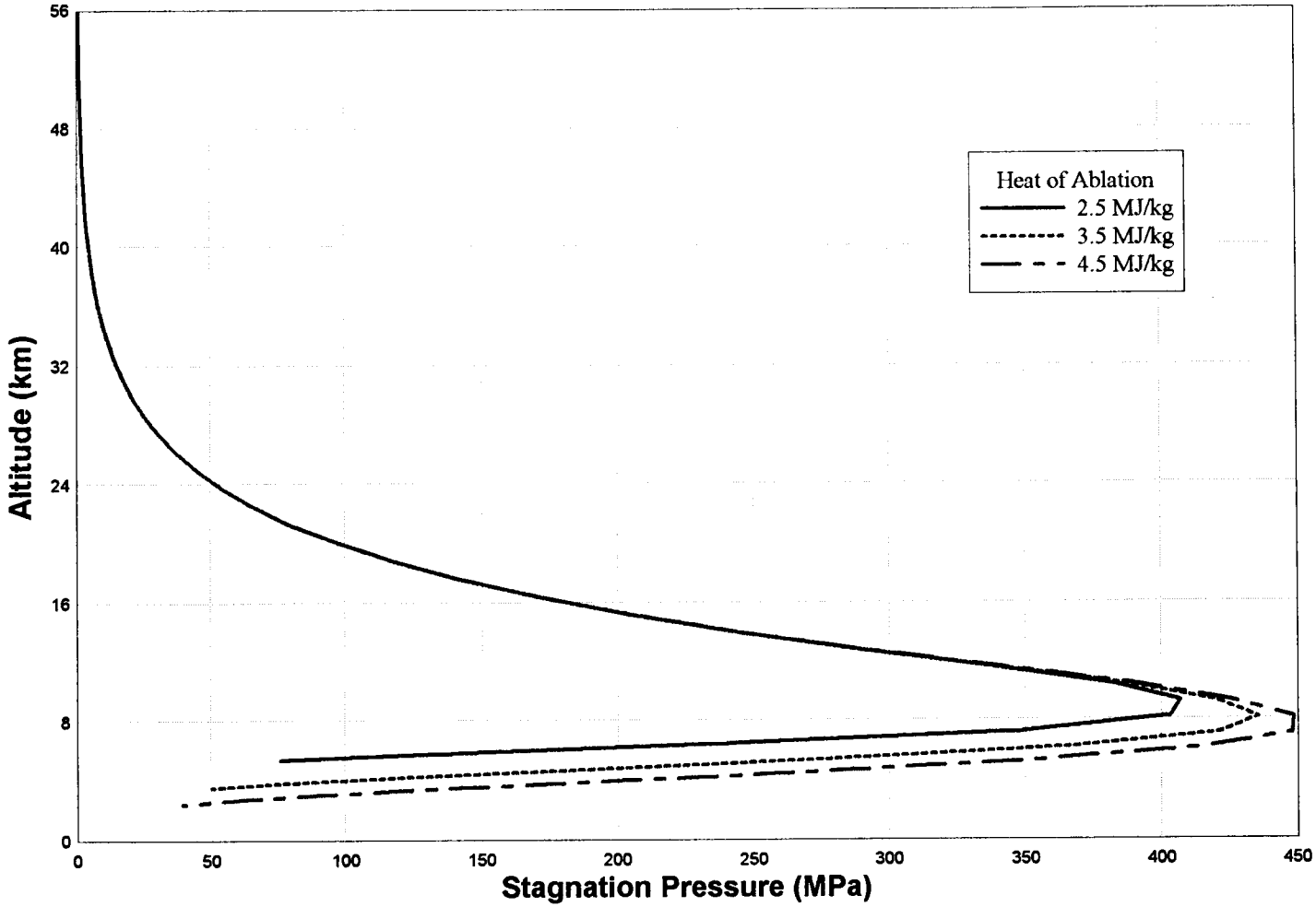


Figure 5.26 - Trajectory Pressure vs. Heat of Ablation (Weak median body, 50 m radius, 35 km/s, 45°)

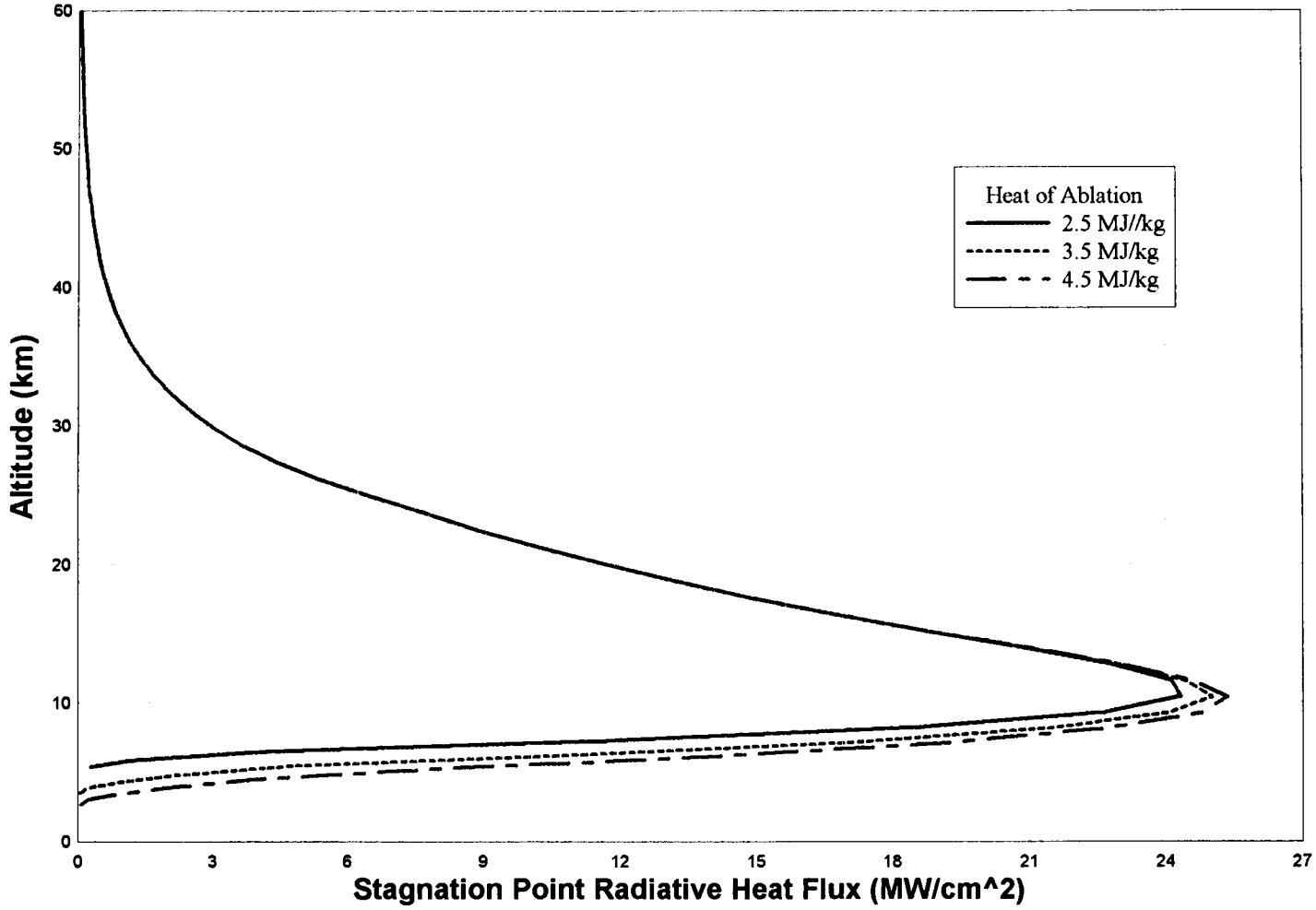


Figure 5.27 - Trajectory Heat Flux vs. Heat of Ablation (Weak median body, 50 m radius, 35 km/s, 45°)

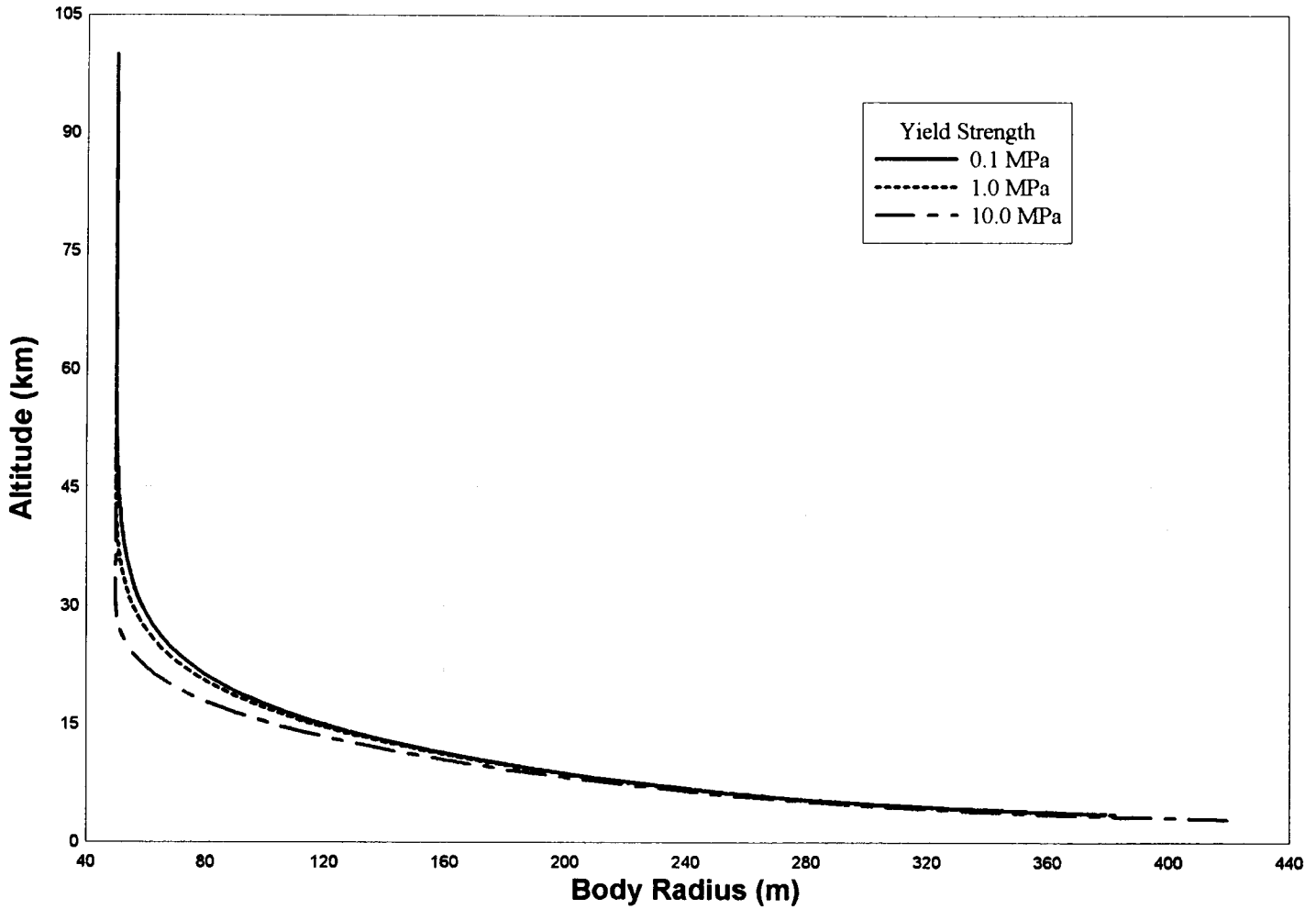


Figure 5.28 - Trajectory Radius vs. Yield Strength (Weak median body, 50 m radius, 35 km/s, 45°)

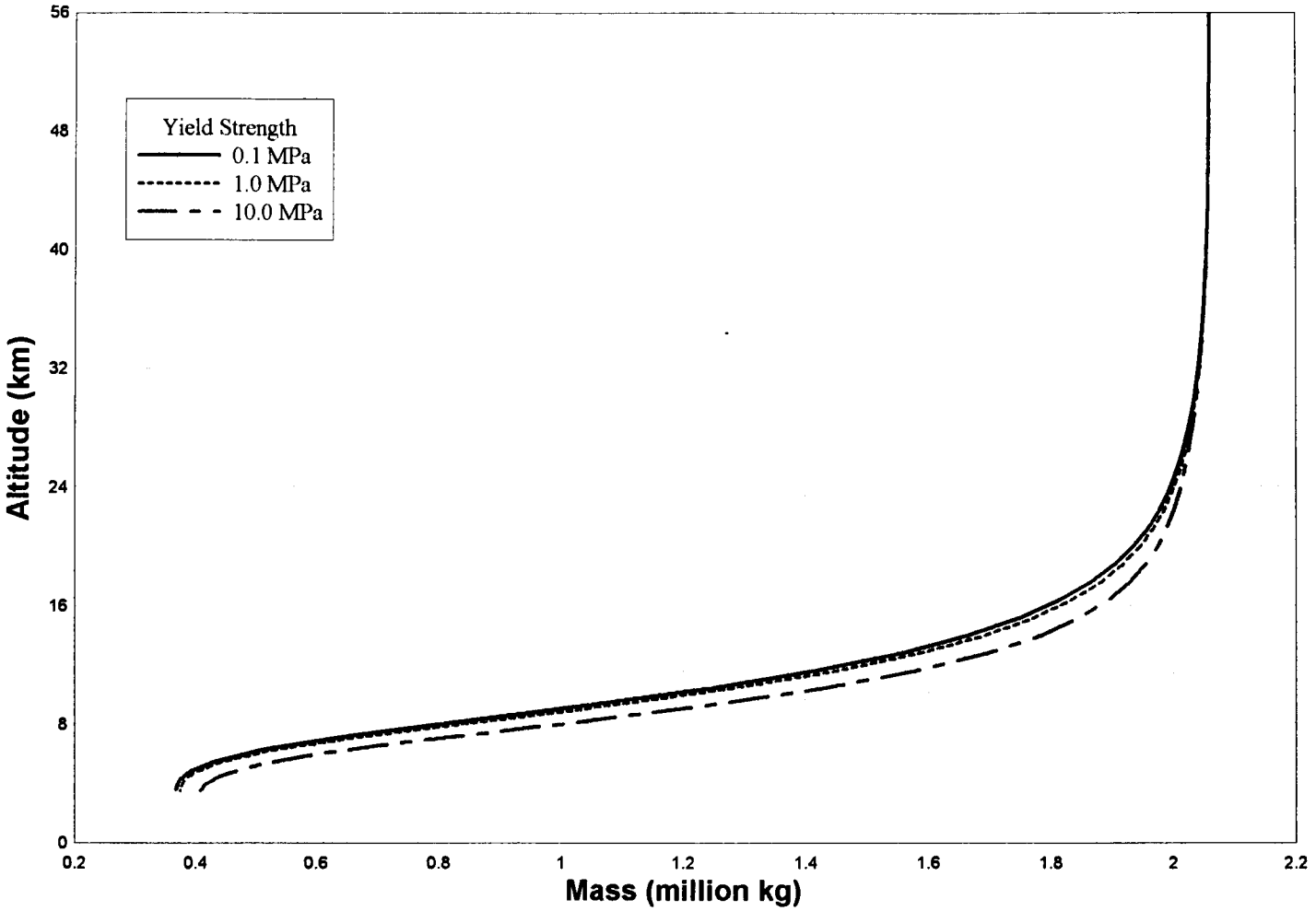


Figure 5.29 - Trajectory Mass vs. Yield Strength (Weak median body, 50 m radius, 35 km/s, 45°)

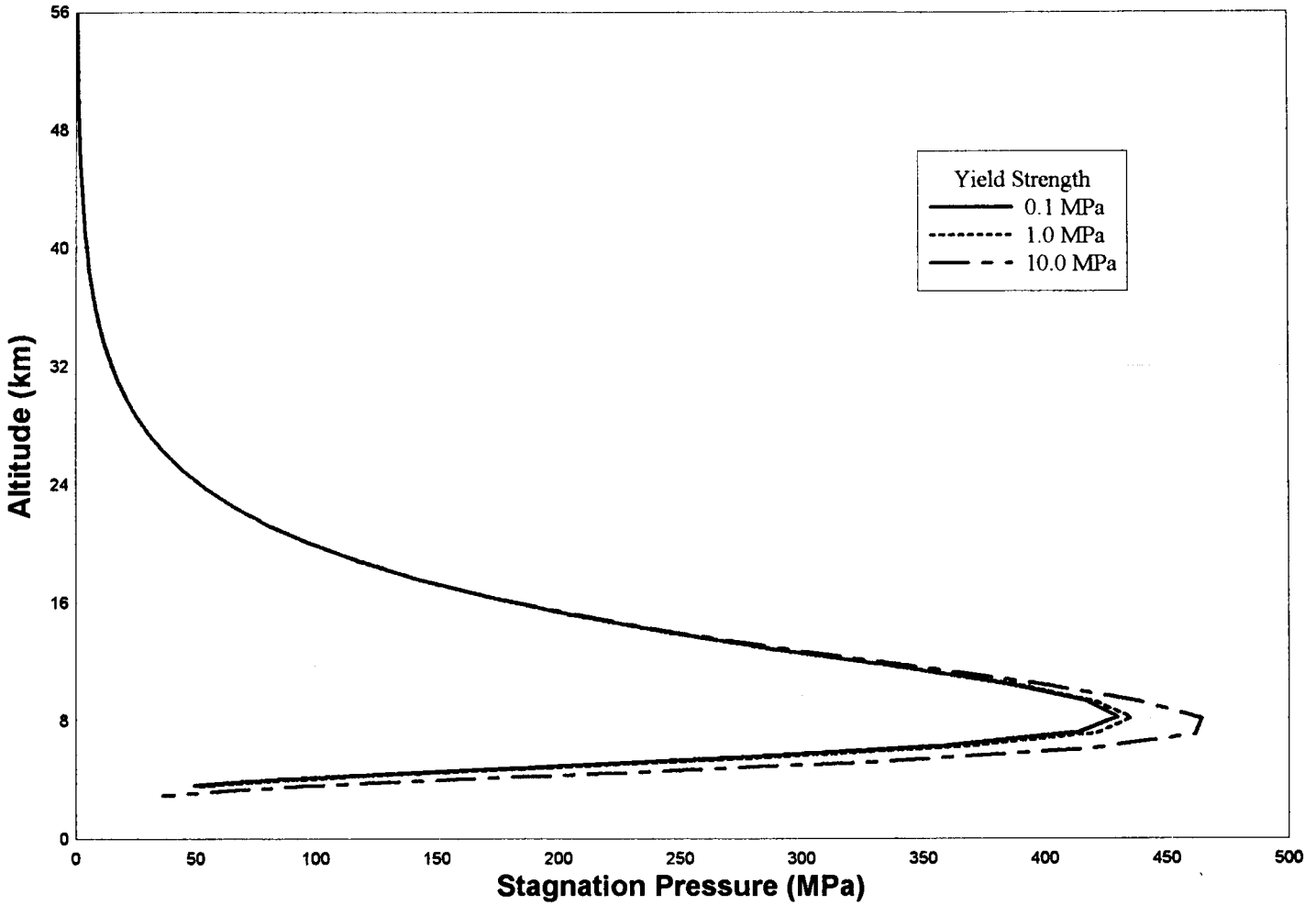


Figure 5.30 - Trajectory Pressure vs. Yield Strength (Weak median body, 50 m radius, 35 km/s, 45°)

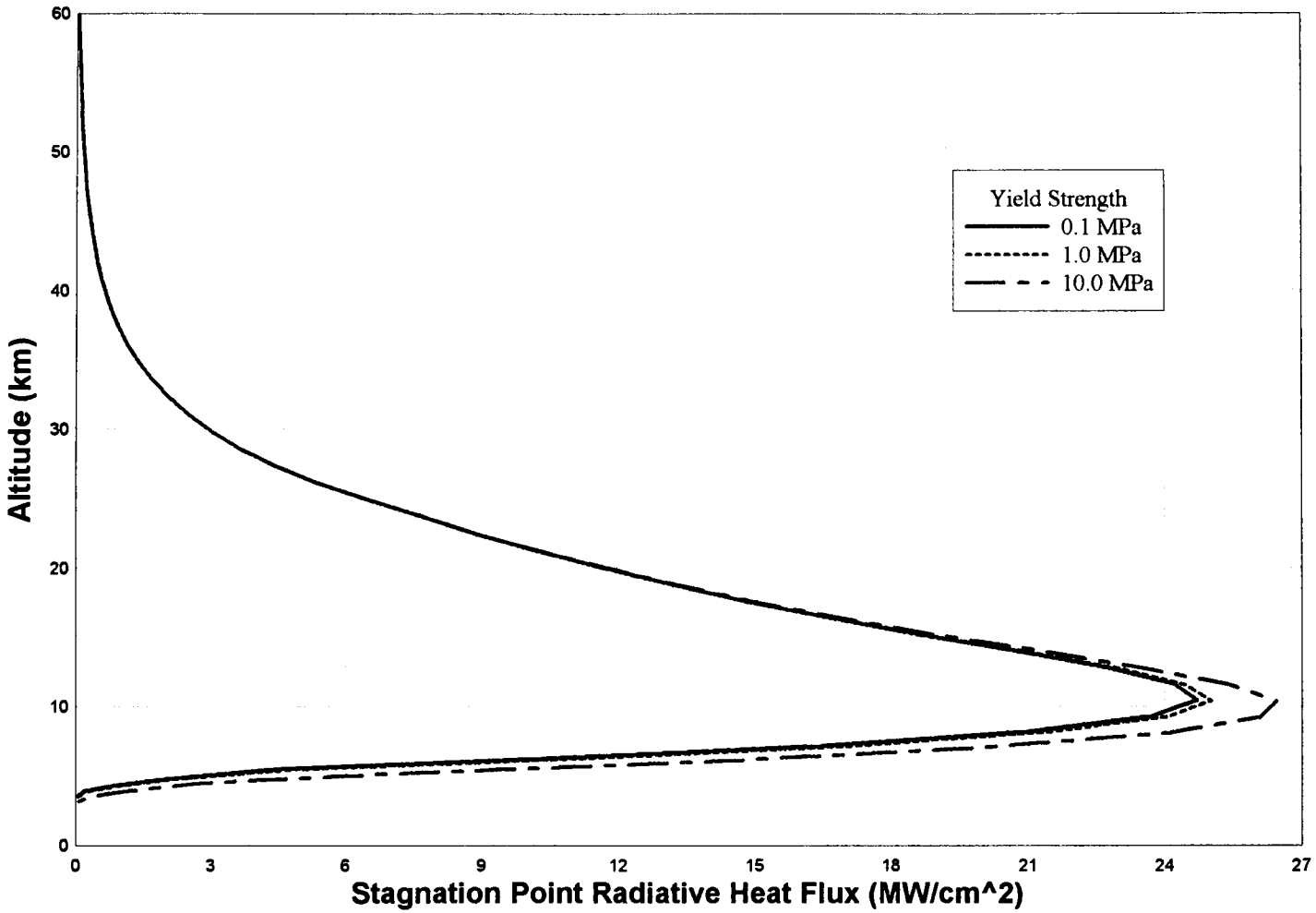


Figure 5.31 - Trajectory Heat Flux vs. Yield Strength (Weak median body, 50 m radius, 35 km/s, 45°)

the results. The trajectory graphs that have been excluded showed no notable differences.

Figure 5.28 is the most significant of these graphs. It shows that a meteor with a high yield strength, breaks up at a lower velocity. However, it is interesting to note that the curves converge back into almost the exact same curve. The difference does result in a larger final body radius at a lower altitude for the higher yield strength body. Other trajectory parameters effected by this variation include the body mass, stagnation pressure, and radiative heat flux. **Figure 5.29** shows that the higher yield strength results in a slightly lower mass loss. **Figures 5.30** and **5.31** also show that the higher yield strength results in a higher peak stagnation pressure and radiative heat flux at the stagnation point.

5.4 Trajectory Analysis: Body Types

As mentioned in section 5.1 and shown in **Table 5.1**, the different meteor body types have been divided into weak, medium, and strong bodies. The previous two sections concentrated mainly on the weak bodies. This section gives a comparison of the trajectories of the three body types. A body with a 50 m radius and entering the atmosphere at 45° is used for the trajectories in this section. The weak, medium, and strong bodies in this section refer to the median values in **Table 5.1**.

Figure 5.32 shows how the trajectory velocity varies for the different body types at different entry velocities. As expected, the strong bodies maintain a higher velocity

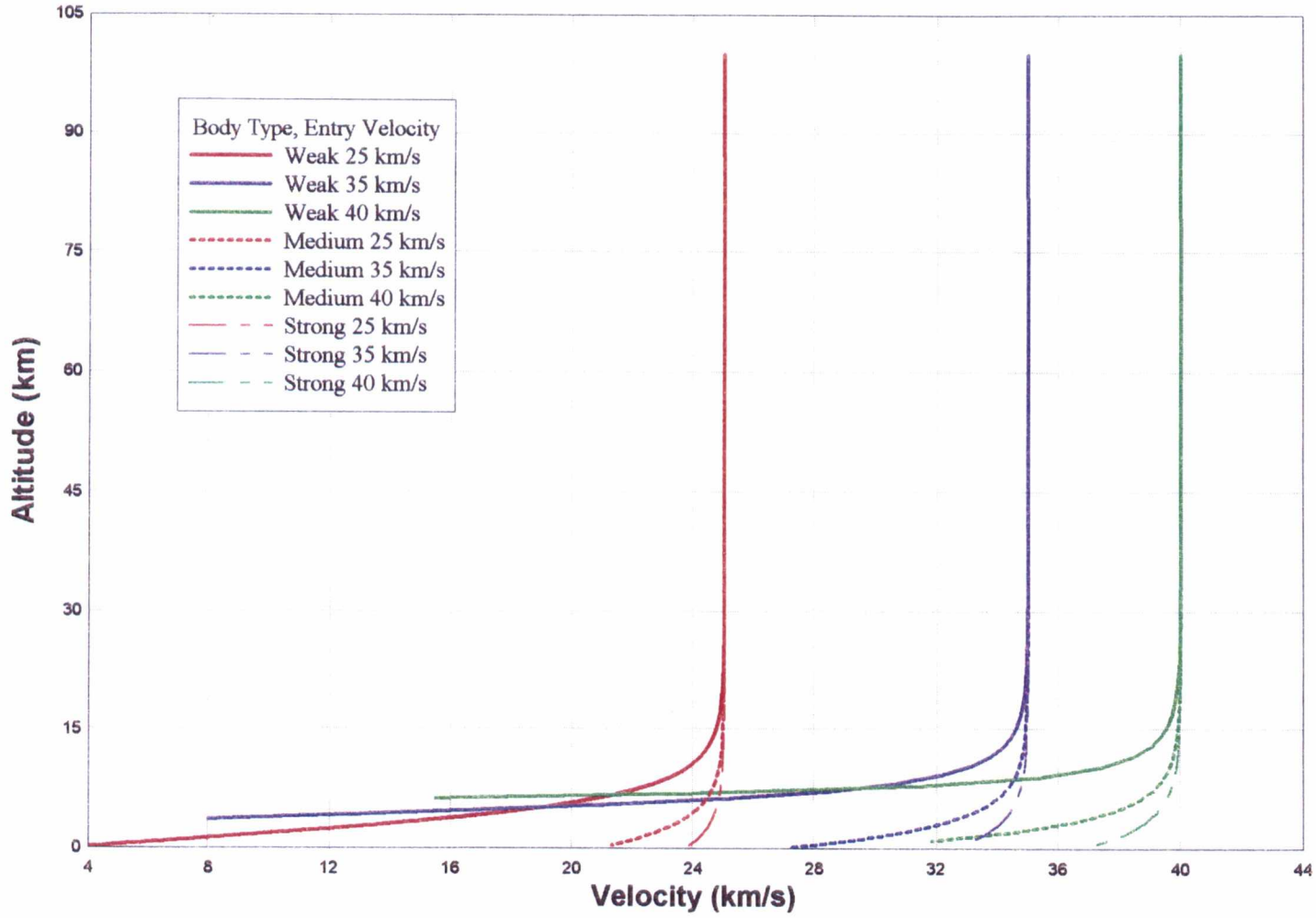


Figure 5.32 - Trajectory Velocity (Median body, 50 m radius, 45°)

because of their higher momentum, the higher yield strength keeping the body intact longer, and the higher heat of ablation reducing the total mass loss. It is notable that the medium and strong bodies show the same trends as are seen in the weak body.

Figure 5.33 gives the percent of initial kinetic energy remaining through the trajectory for the various body types at different entry velocities. Again, the medium and strong bodies show trends similar to the weak body. The weak body loses kinetic energy at a higher altitude than either the medium or strong bodies for several reasons, including a smaller momentum, a lower yield strength resulting in an earlier breakup, and a lower heat of ablation allowing a larger mass loss due to ablation. This graph is very significant, in that the airburst altitude, or the height of the maximum rate of deposition of energy in the atmosphere, is approximated as occurring when 50% of the initial kinetic energy remains. The airburst altitude is discussed further in section 5.5.

Figures 5.34 and **5.35** show that the stronger bodies have a higher peak stagnation point temperature and peak stagnation pressure. This is an expected result since the stronger bodies maintain a higher velocity, and the stagnation properties are strongly dependent on the velocity of the body.

Figure 5.36 shows the similarities in the mass loss for the different body types. The most significant trend here is that the stronger bodies have a smaller mass loss. This is because the bodies resist breakup for a longer period of time. After breakup

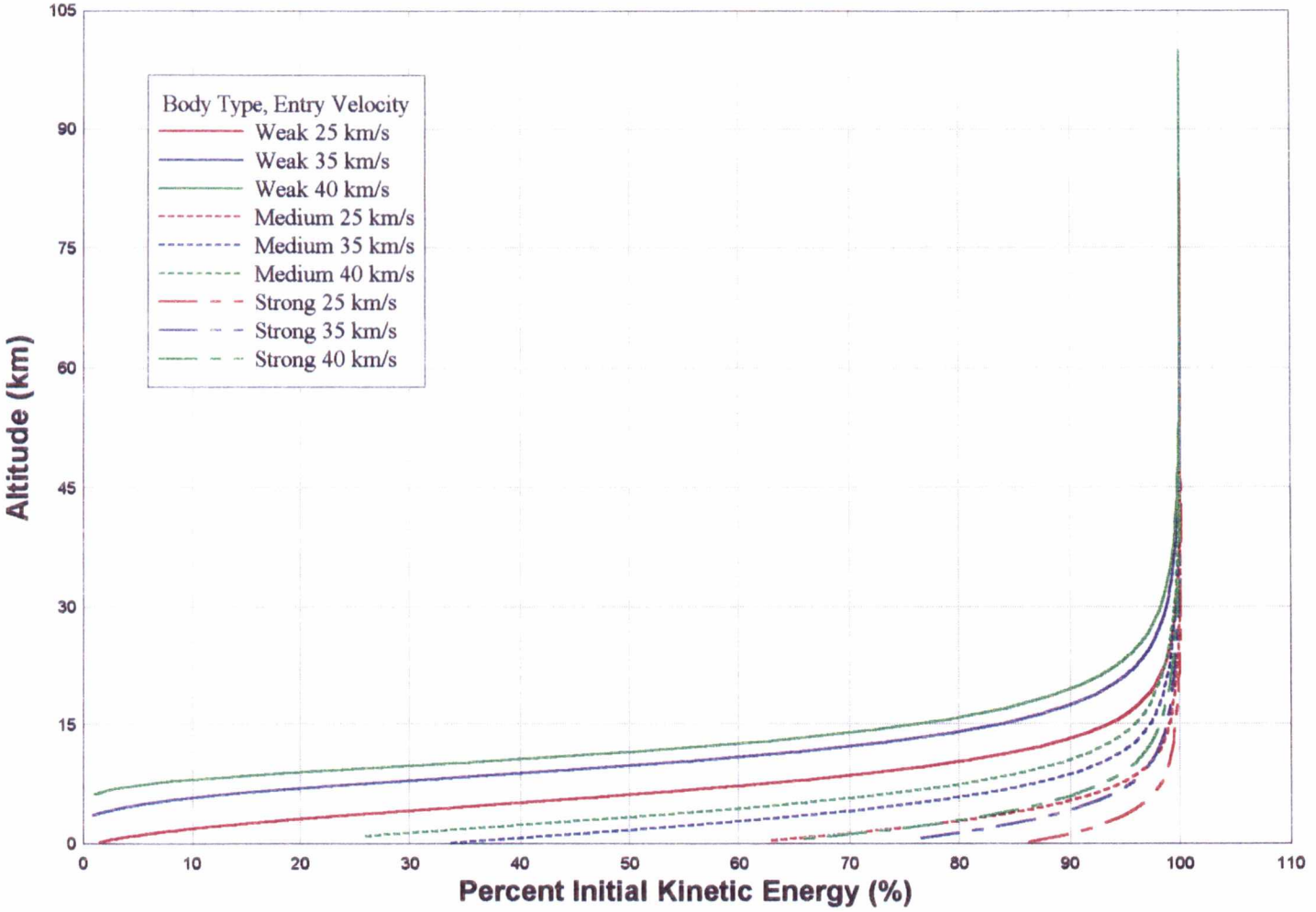


Figure 5.33 - Trajectory Kinetic Energy (Median body, 50 m radius, 45°)

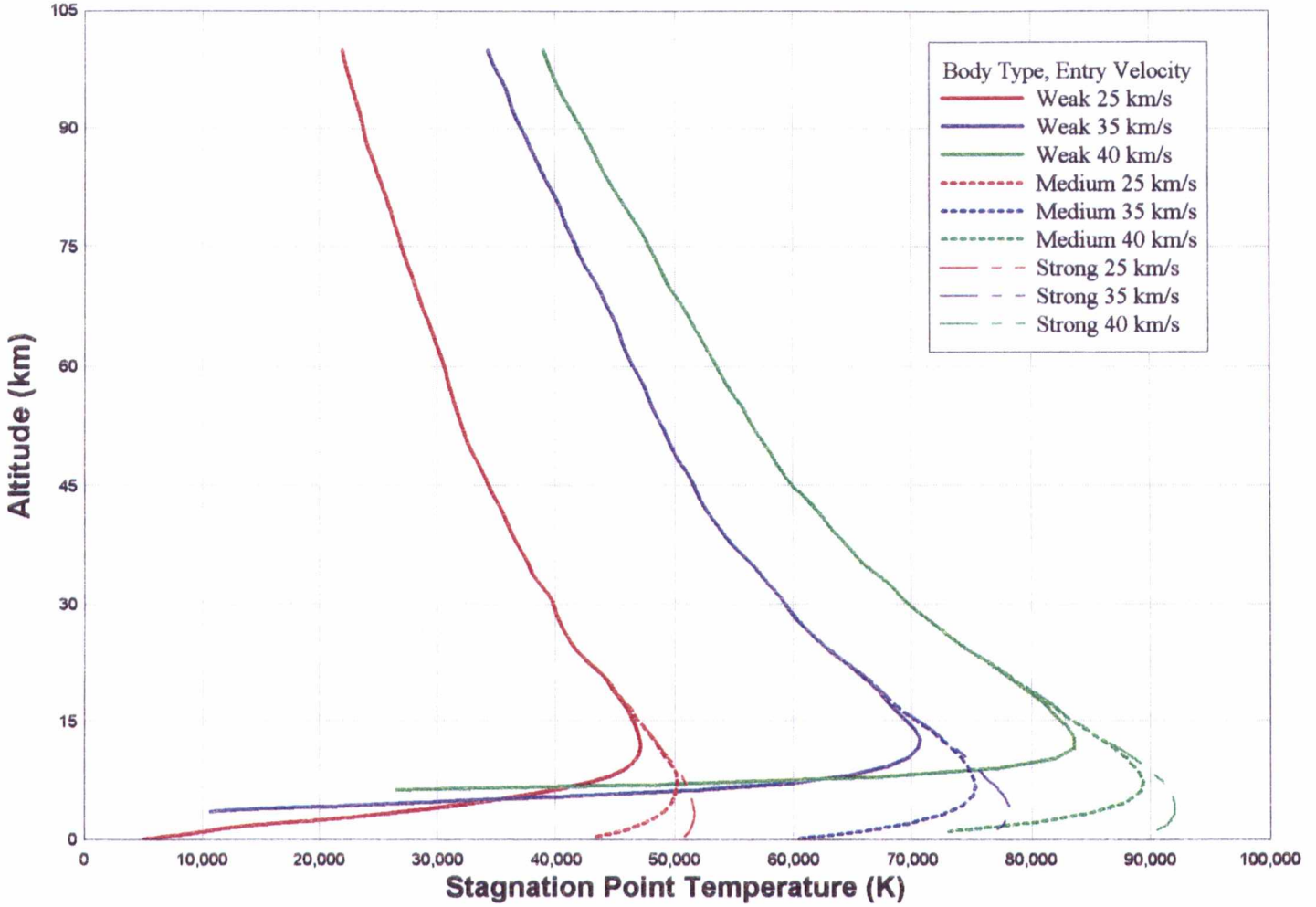


Figure 5.34 - Trajectory Temperature (Median body, 50 m radius, 45°)

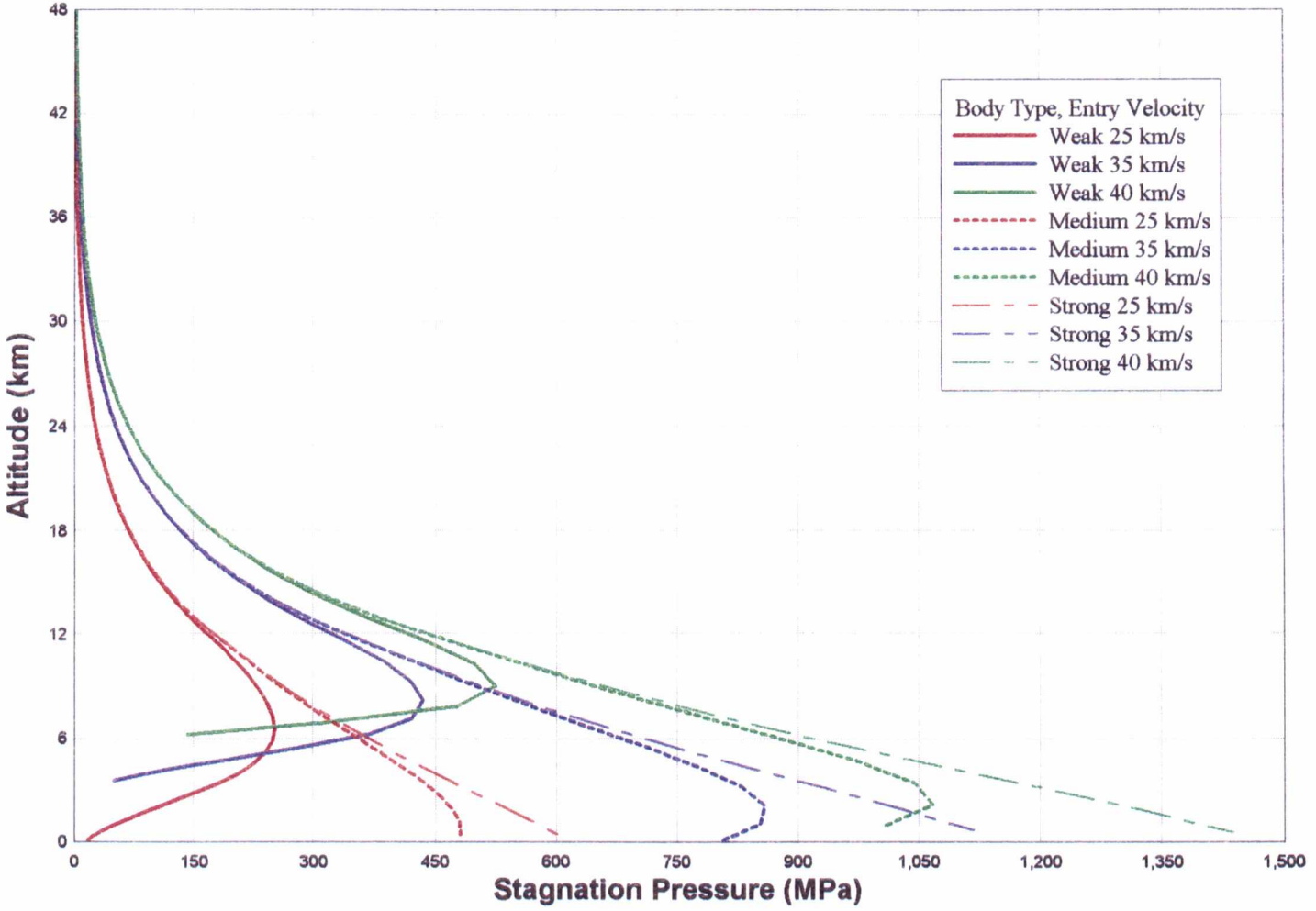


Figure 5.35 - Trajectory Pressure (Median body, 50 m radius, 45°)

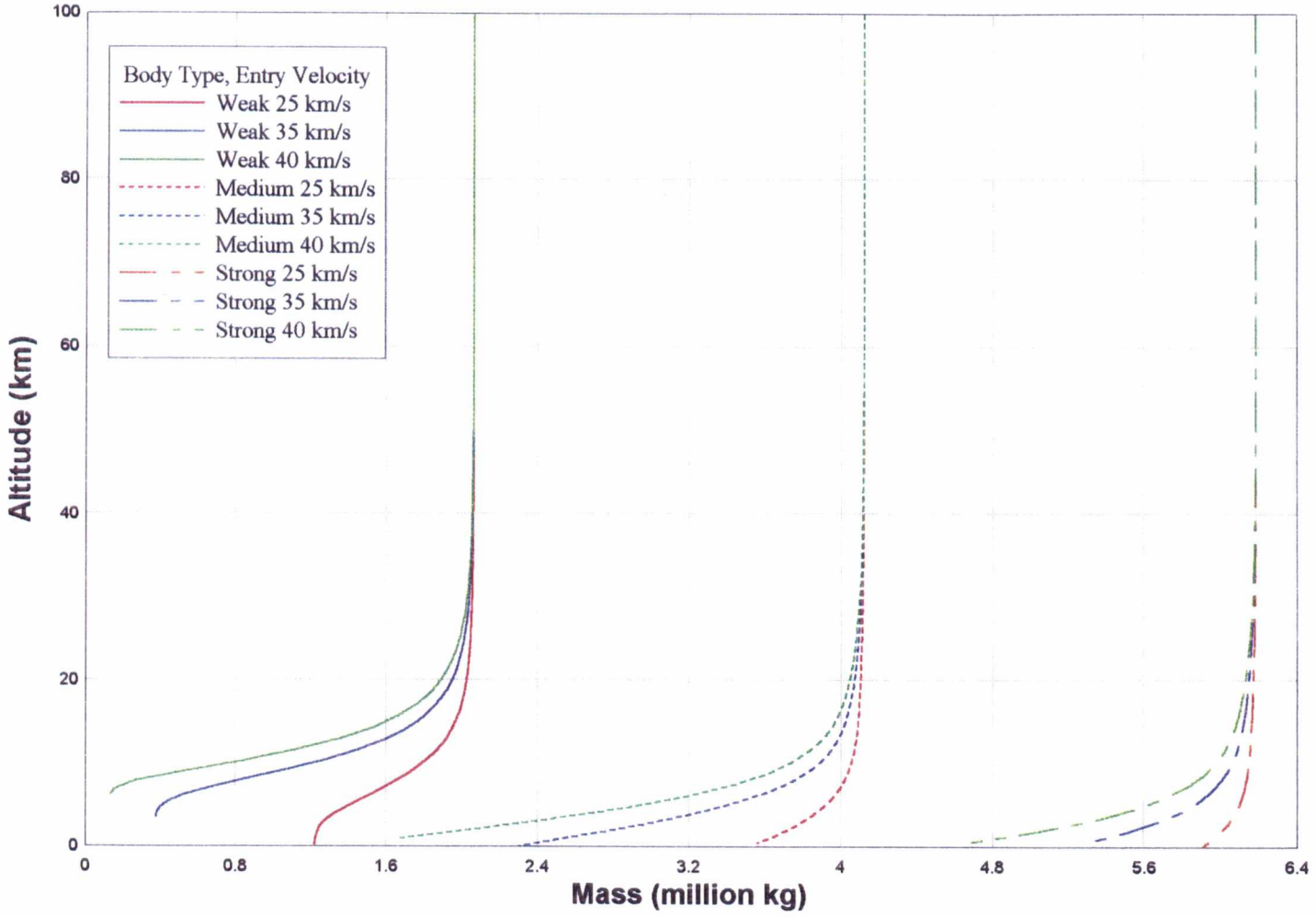


Figure 5.36 - Trajectory Mass (Median body, 50 m radius, 45°)

occurs, the body radius begins to increase, giving the radiative heat flux a larger surface. The larger surface absorbs more heat and results in higher ablation. Only the weak body slows enough to reduce the amount of ablation before striking the ground or losing all of the body's kinetic energy.

The body radius during the trajectory is given in **Figure 5.37**. As mentioned above and graphically shown in **Figure 5.37**, the stronger bodies begin to breakup at a lower altitude than either of the two other body types. It is also notable that the slowest velocity results in the largest radius increase, if the body does not strike the ground first.

The final graph of this section, **Figure 5.38**, shows the amount of radiative heat transfer at the stagnation point for various body types. The radiative heat flux is very temperature dependent. Comparison with **Figure 5.34** shows the same general trends. The stronger bodies and the faster bodies achieve a larger radiative heat flux at the stagnation point.

5.5 Airburst Altitude

As a meteor is traveling through the atmosphere, it gradually loses energy. However, if the body does not impact the surface first, there is a small altitude range over which the body loses most of its energy. Within this range exists the point of peak energy dispersion. This occurs when approximately 50% of the body's initial kinetic energy has been lost.

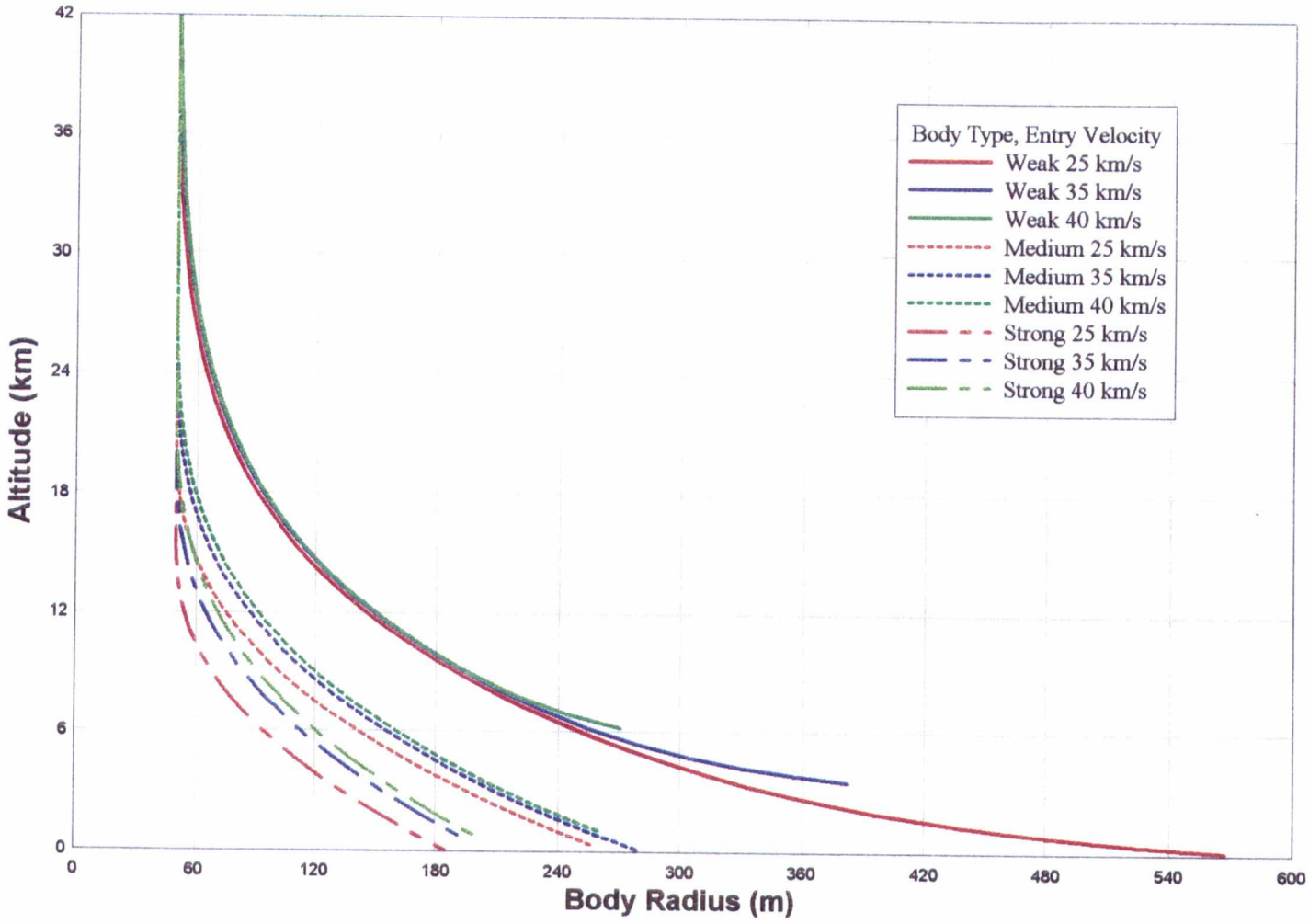


Figure 5.37 - Trajectory Radius (Median body, 50 m radius, 45°)

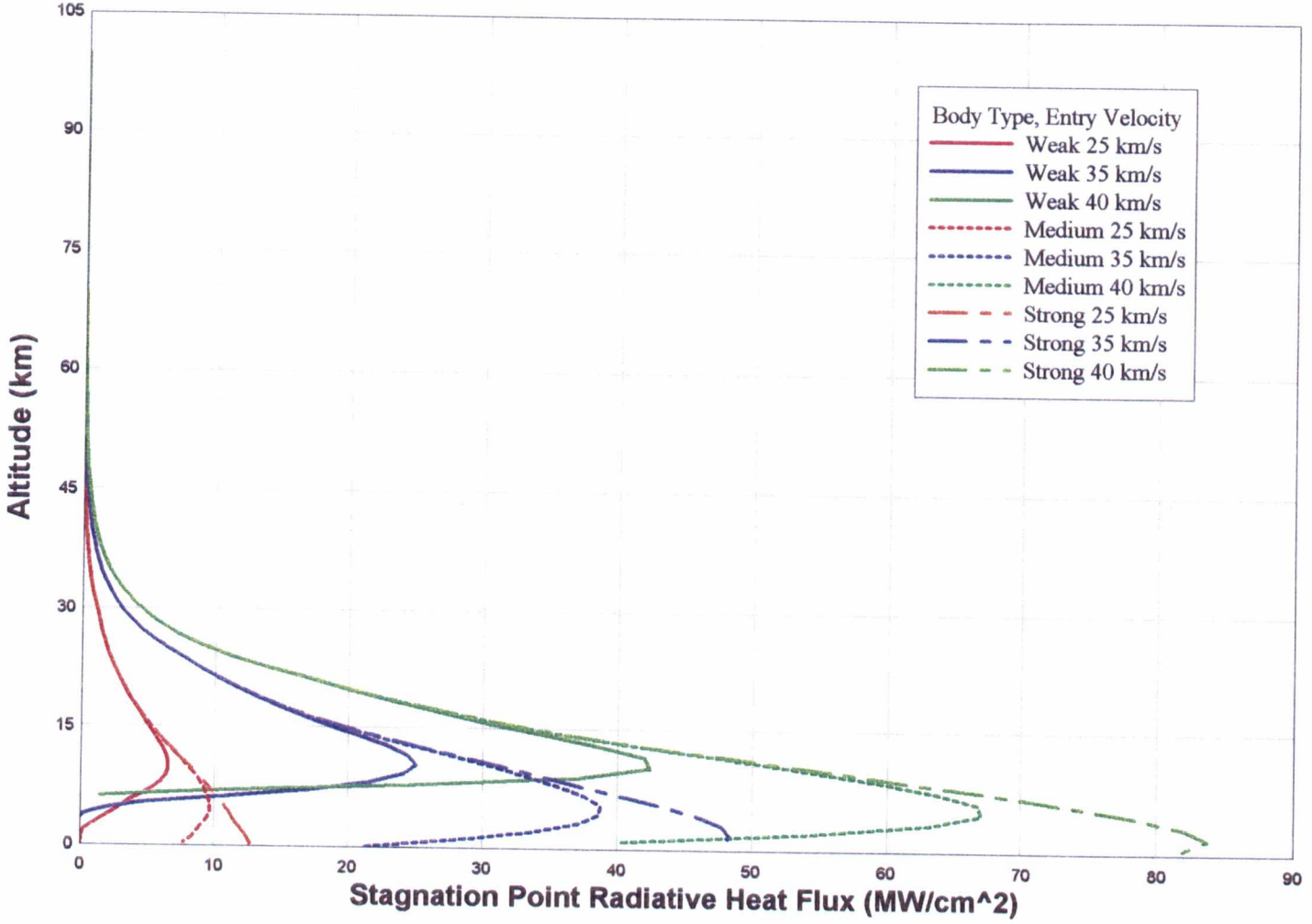


Figure 5.38 - Trajectory Heat Flux (Median body, 50 m radius, 45°)

Much of the current work on meteor entry was driven by a debate over a meteor that apparently airburst over a remote area of Siberia called Tunguska. In fact, a major driving force behind the original code used for this work was the Tunguska event. The general debate has focused around what type of body could have airburst within the range of acceptable altitudes. Some argue that it had to be an asteroid (Sekanina 1983), while others insisted that it had to be a comet (Turco 1982), and still others have argued that it could have been either (Lyne 1996). Therefore, this section has been dedicated to looking at the airburst altitude for the various body types.

The first graph in **Figure 5.39** shows the various airburst altitudes for a weak body with different entry parameters. It shows that higher velocity bodies will airburst at a higher altitude, a steeper entry angle results in a lower airburst altitude, and a larger initial body radius results in a lower airburst altitude.

Figure 5.40 shows a comparison of the different body types and when they airburst. Generally, the weaker bodies airburst at a higher altitude than the stronger bodies. Also, note that the trends for the weak body appear to be very similar to the medium and strong bodies.

Figures 5.41 through **5.43** show how the body parameters affect the airburst altitude. **Figure 5.41** gives the airburst altitude for various densities using each of the three body types. An increase in the density clearly lowers the airburst altitude. However, this effect is much more pronounced for the weaker body. The second

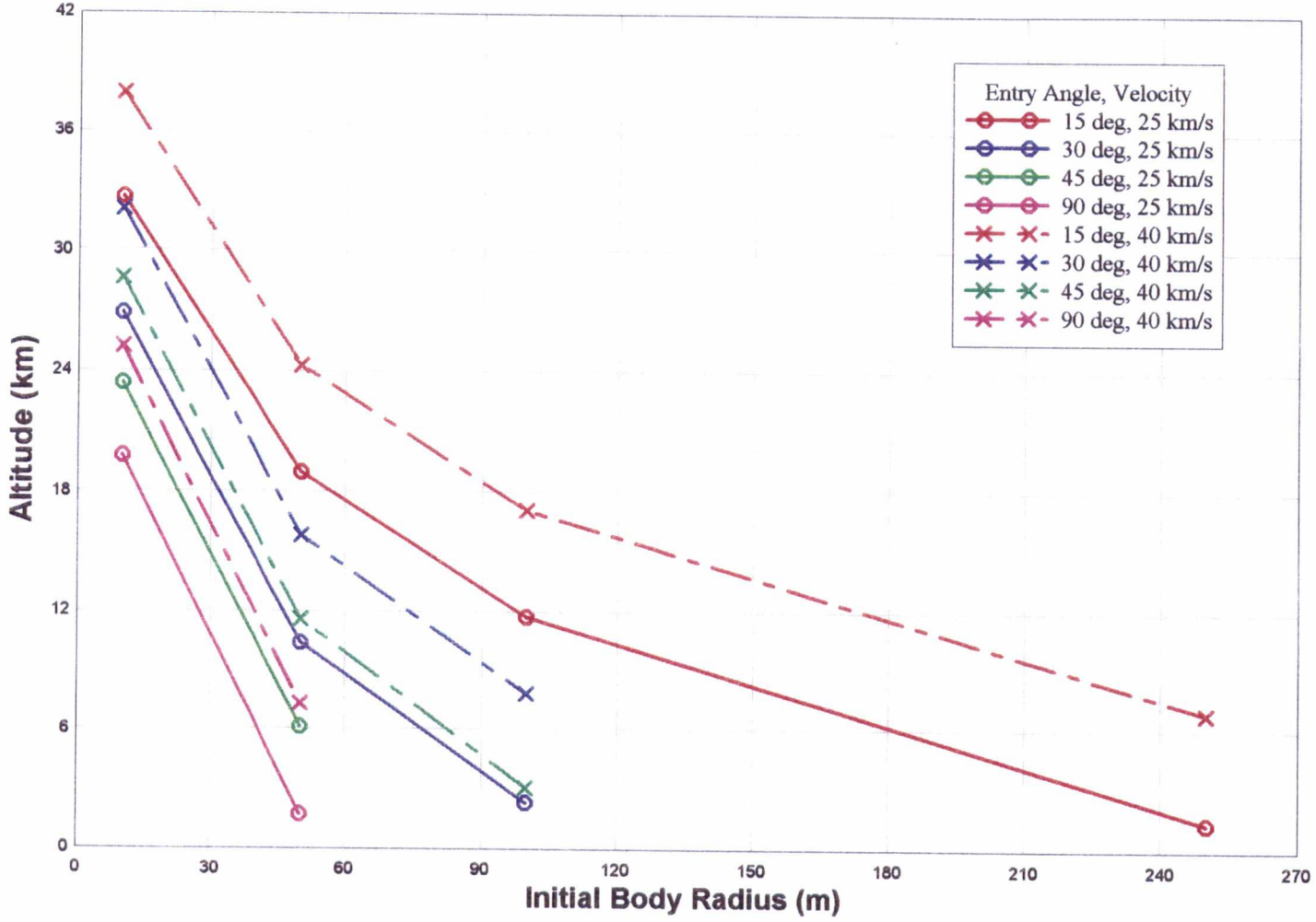


Figure 5.39 - Airburst altitude for various entry parameters (Weak median body)

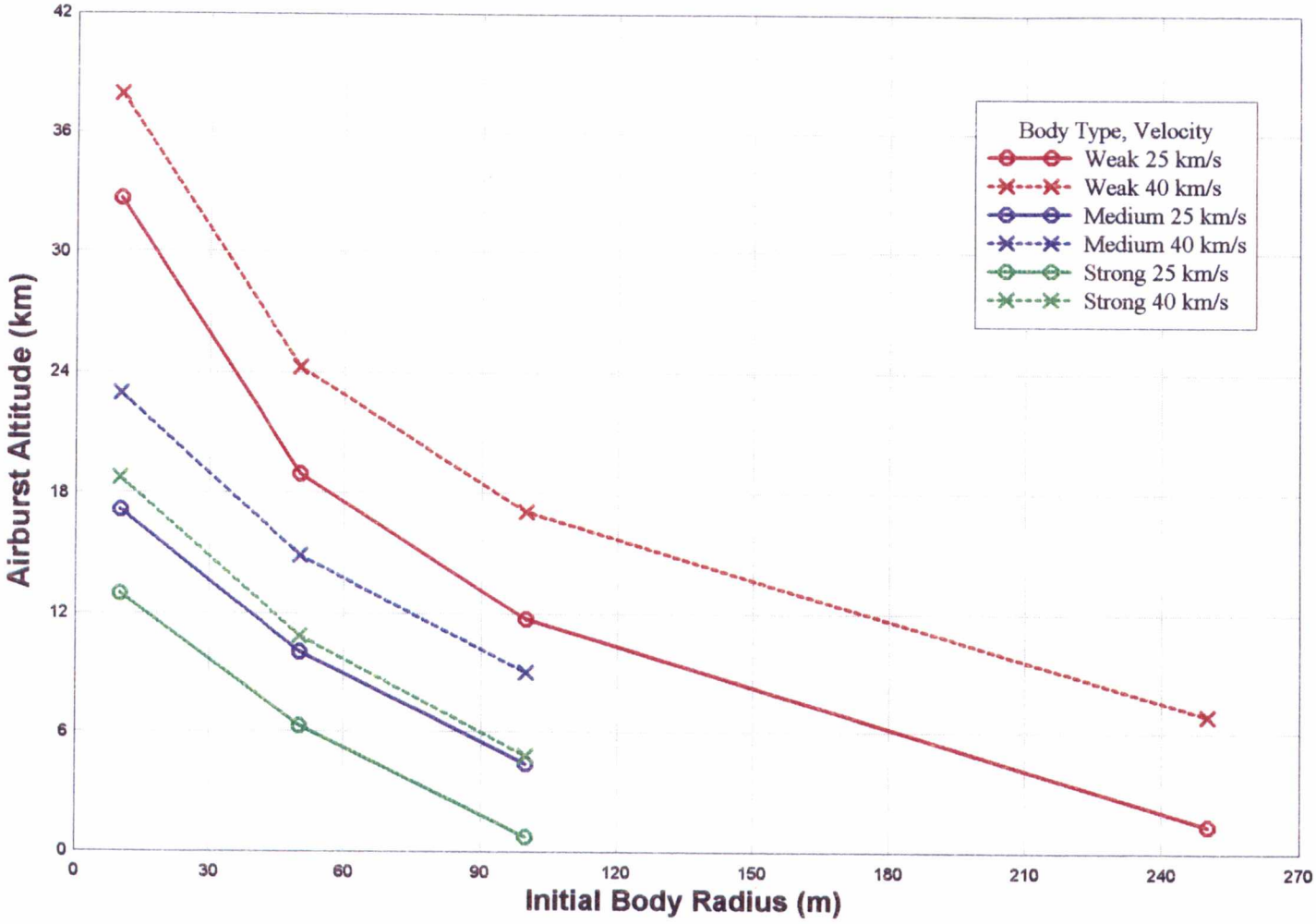


Figure 5.40 - Airburst altitude for various body types (Median body, 15°)

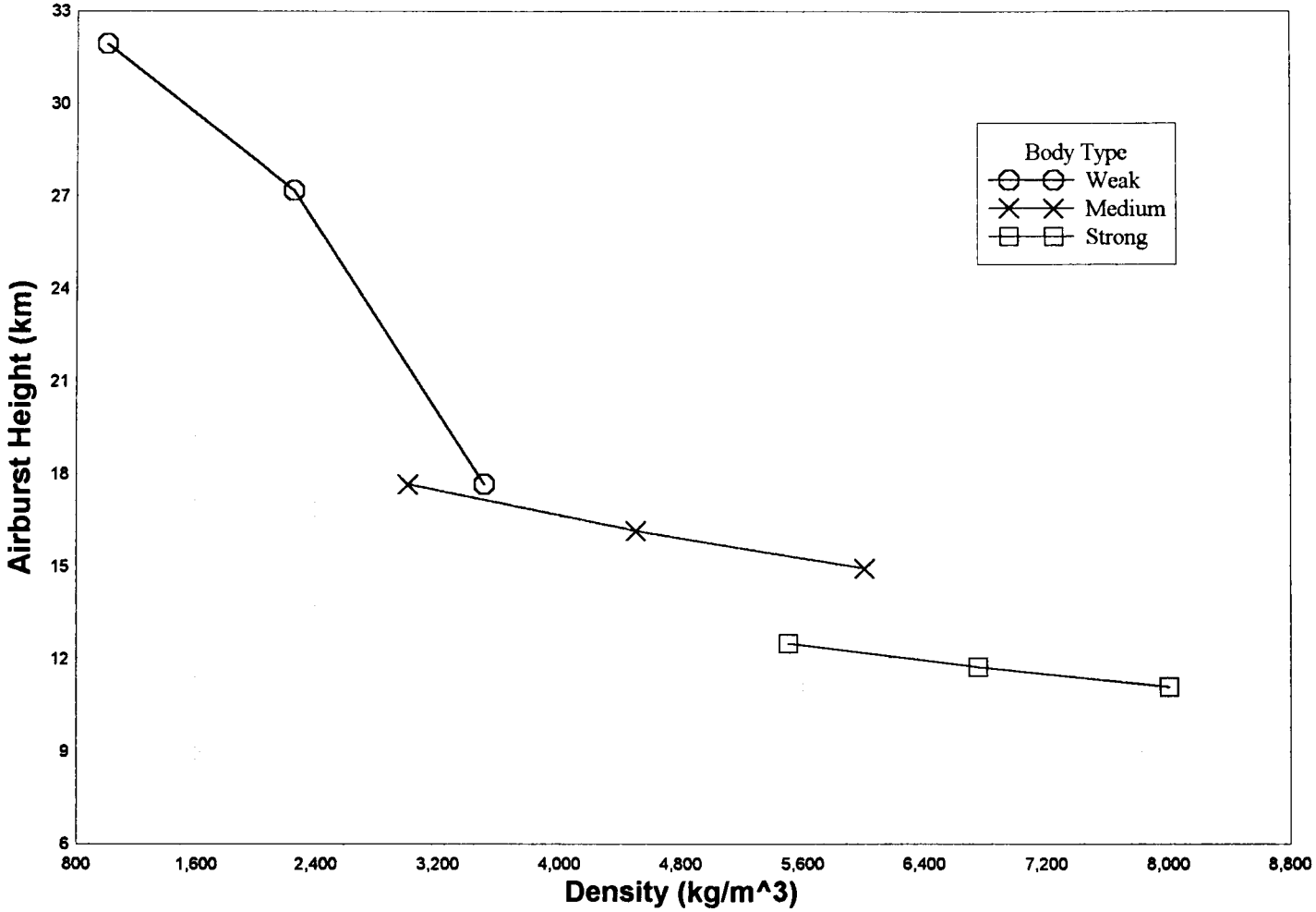


Figure 5.41 - Airburst Height vs. Density (10 m radius, 35 km/s, 15°)

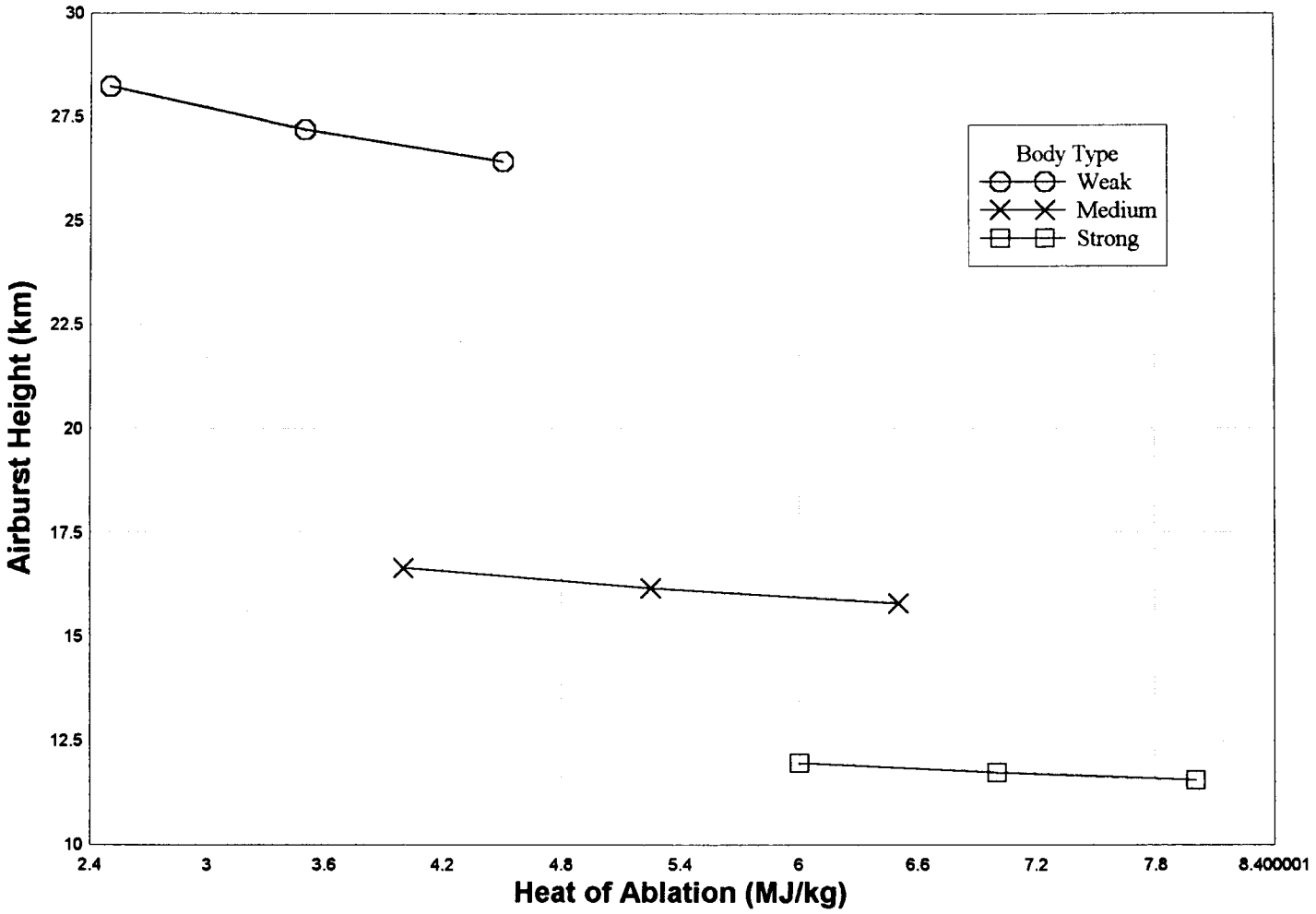


Figure 5.42 - Airburst Height vs. Heat of Ablation (10 m radius, 35 km/s, 15°)

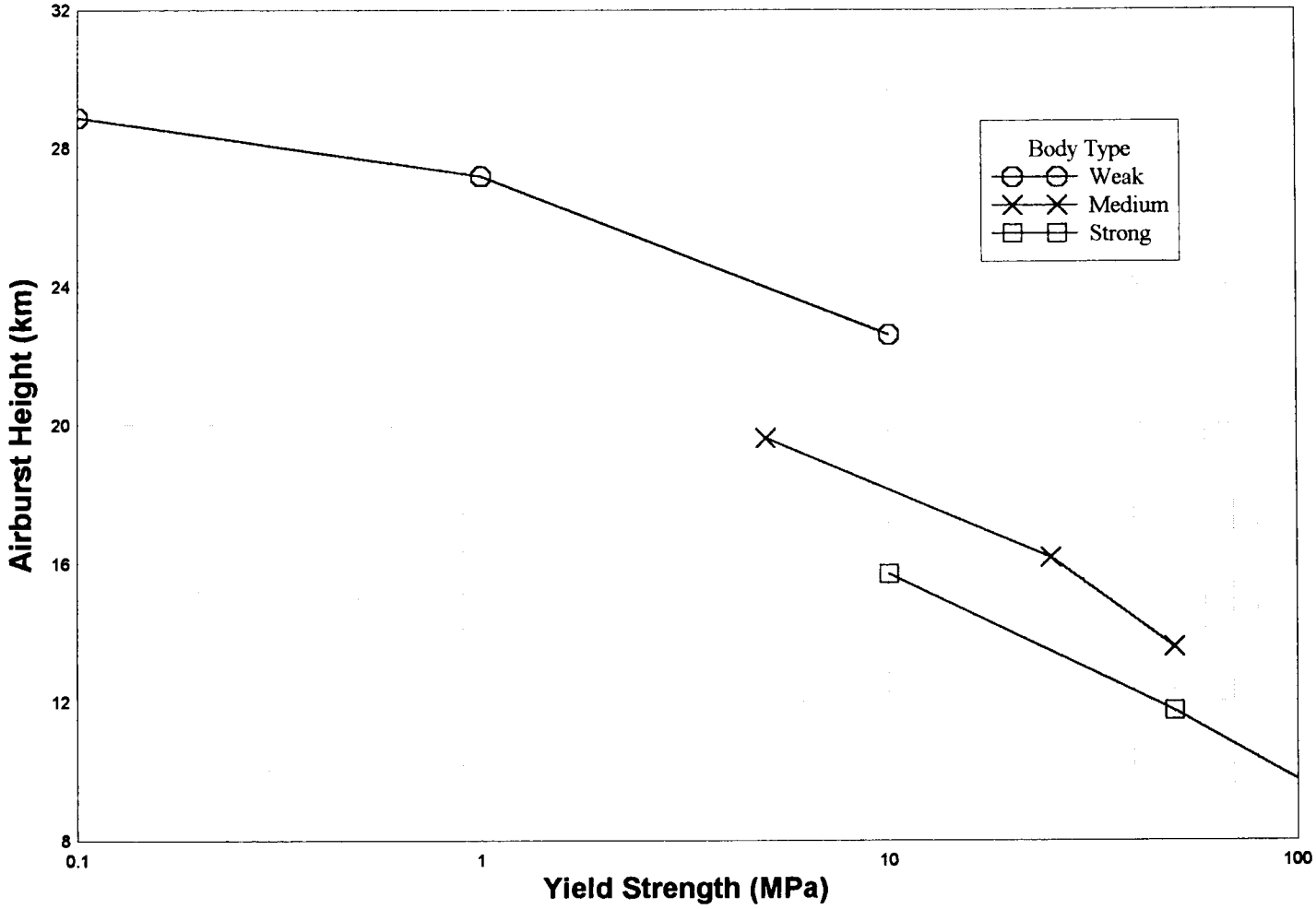


Figure 5.43 - Airburst Height vs. Yield Strength (10 m radius, 35 km/s, 15°)

figure shows how the heat of ablation affects the airburst altitude. The airburst height declines as the heat of ablation is increased. This effect is again most pronounced for the weak body. The effect of varying the heat of ablation of the body had a significantly smaller effect on the airburst altitude than the density variation. Finally, **Figure 5.43** shows the effect that various yield strengths have on the airburst altitude. For all three cases, a significant drop in the airburst altitude is found for an increase in the yield strength of the body.

5.6 Radiative Heat Flux Across the Face

In addition to the analysis presented thus far in this chapter, the METEOR program is capable of calculating the temperature and radiative heat flux across the windward face of the body during the trajectory. **Figures 5.44** and **5.45** show the temperature and stagnation point heat transfer along the windward face at time intervals through the trajectory. The reference point of zero seconds represents when the body is 100 km from the surface. These graphs show temperature and heat flux across half of the nose with 0° corresponding to the center and rotating across to an edge at 90°.

Figure 5.44 shows the temperature across half of the body face at half-second intervals through part of the trajectory. The most notable effect here is the gradual smoothing of the curve. This shows that a smooth temperature transition ultimately exists across the windward face. The second graph, **Figure 5.45**, shows the radiative heat flux at each point along half of the body face.

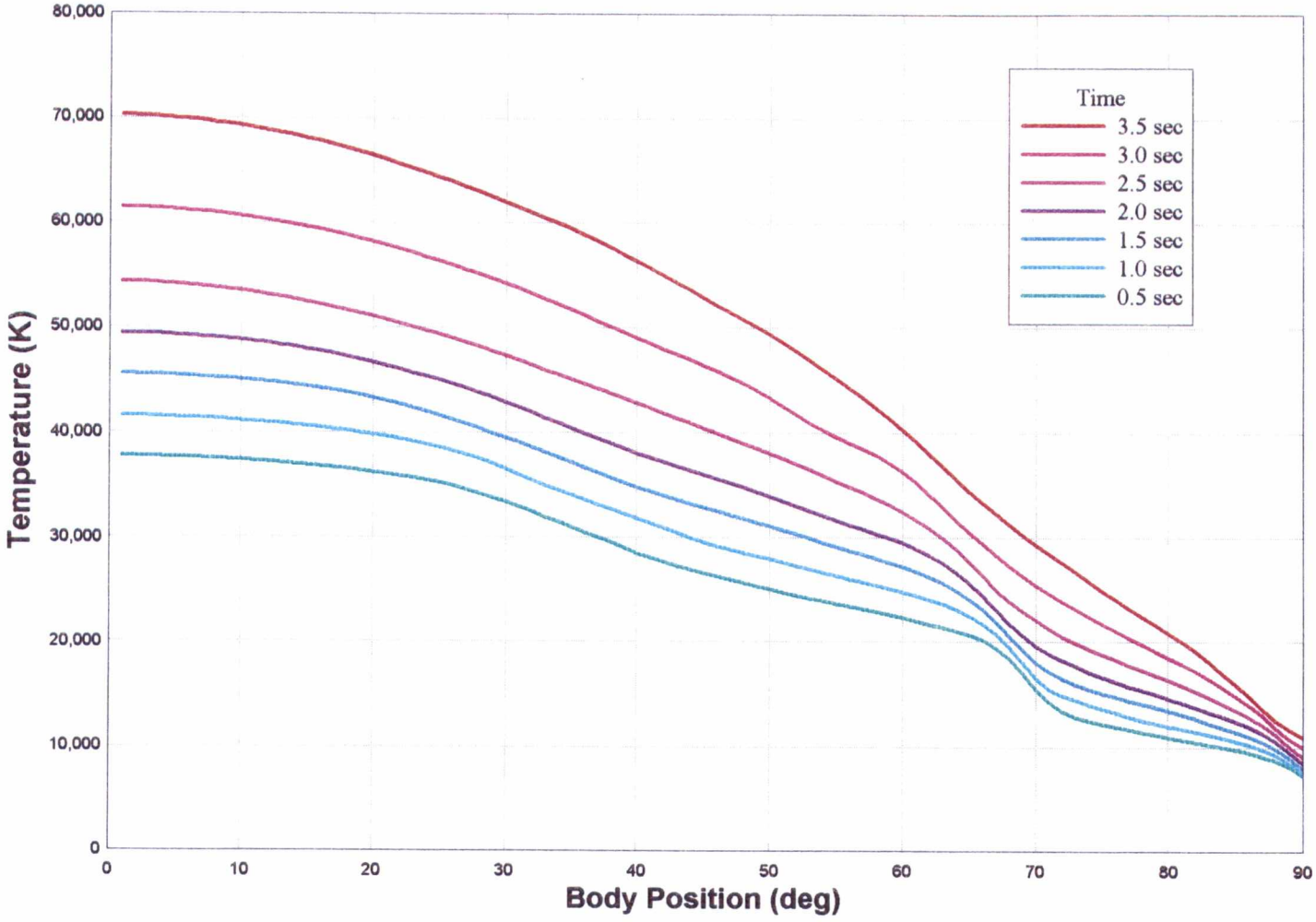


Figure 5.44 - Temperature vs. Body Position (Weak median body, 50 m radius, 35 km/s, 45°)

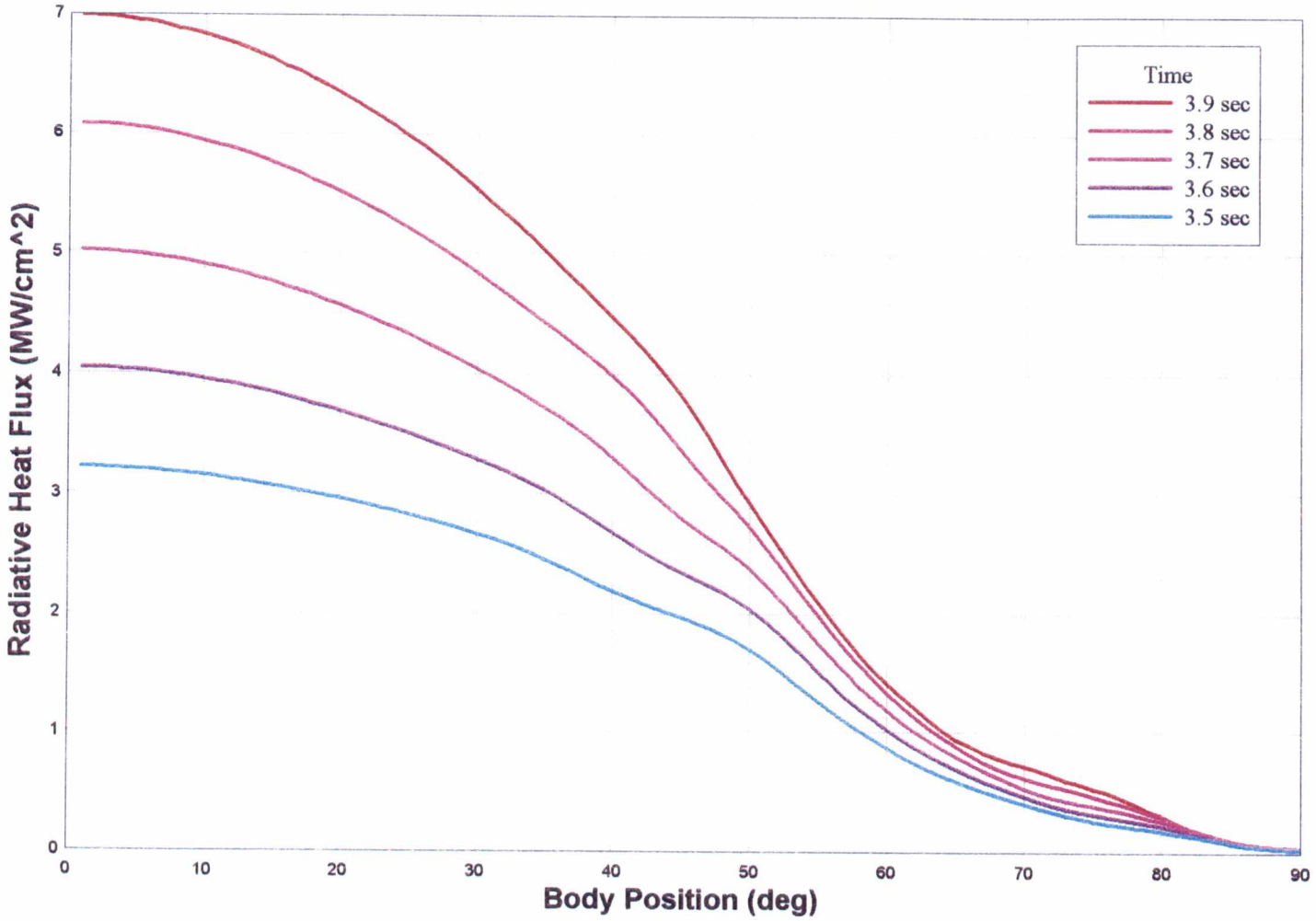


Figure 5.45 - Heat Flux vs. Body Position (Weak median body, 25 m radius, 30 km/s, 45°)

CHAPTER 6

CONCLUSION

6.1 Current Status

Prior to this work, meteor trajectory analysis was limited to conditions in which shock layer temperatures could not exceed 30,000 K. This limitation meant that analysis was restricted to meteors traveling at speeds below 20 km/s. The METEOR program presented here has used thermodynamic data from a 1961 Chance Vought report to extend this temperature range to 100,000 K, thereby allowing analysis of bodies traveling at speeds in excess of 40 km/s.

The improved code was used to conduct a parametric study of meteor entry, varying both entry conditions and body characteristics. Key findings were that meteor density is a primary determinant of airburst altitude (with more dense bodies reaching lower altitudes), while heat of ablation and mechanical yield strength have a less pronounced influence.

6.2 Future Work

While this new work has greatly extended the capabilities of the previous program, it is still not a complete model of meteor entry. Some parts of the program use approximations that have not been fully analyzed or could be improved with more work.

The adjustment to the Green-Nicolet approximation was not meant to be a permanent fix. The chance of a failed convergence has been greatly reduced, but not eliminated. A better curve fit or an improved approximation could eliminate this problem in the future.

The heat transfer across the inside of the shock layer uses the Goulard approximation across the entire face of the body. However, it has only been validated for at the stagnation point. Further work with this could either validate that the approximation is reasonable across the entire face, or replace it with a better approximation.

In addition, by writing a current computer code to model the work done by the Chance Vought group, better data could be used for the curve fits or the curve fit could actually be replaced by the full code. Additionally, the temperature, pressure, enthalpy, and density ranges could possibly be extended even further than this model.

The mechanical fragmentation model could also be substantially improved from the simple analytical model currently employed. Incorporation of a fragmentation model similar to that used by Baldwin and Shaeffer (1971) using yield strength data derived from observation of recent meteor entries should receive high priority.

REFERENCES

REFERENCES

Akima, Hiroshi. "Algorithm 474: Bivariate Interpolation and Smooth Surface Fitting Based on Local Procedures [E2]," *Communications of the ACM*, Vol. 17, No. 1, January 1974.

Anderson, John D. *Hypersonic and High Temperature Gas Dynamics*. McGraw-Hill: New York, 1989.

Anderson, John D. *Modern Compressible Flow With Historical Perspective*. McGraw-Hill: New York, 1990.

Anonymous (Sewell, K. G.). "Thermodynamics Properties of High Temperature Air," *Chance Vought Research Center Report No. RE-1R-14*, Dallas, Texas: Chance Vought Research Center, June 1961.

Bailey, Harry E. "Programs for Computing Equilibrium Thermodynamic Properties of Gases," *NASA Technical Note D-3921*, April 1967.

Baldwin, Barrett and Yvonne Sheaffer. "Ablation and Breakup of Large Meteoroids during Atmospheric Entry," *Journal of Geophysical Research*, Vol. 76, No. 19, July 1971.

Biberman, L. M., S. Y. Bronin, and A. N. Lagar'kov. "Radiative and Convective Heat Transfer in Hypersonic Flow Around a Blunt Body," *Mech. Zidkosty Gasa*, Vol. 5, pp. 112-121, 1972.

Biberman, L. M., S. Y. Bronin, M. V. Brykin, and A. K. Mnatsakanyan. "Influence of the Ablation on the Heat Transfer Near the Stagnation-Point for a Blunt Body," 6th International Heat Transfer Conference, Vol. 3, pp. 397-401, Toronto, Ottawa, 1978a.

Biberman, L. M., S. Y. Bronin, M. V. Brykin, and A. K. Mnatsakanyan. "Influence of the Ablated Thermal Insulation Coat on the Heat Transfer at the Stagnation-Point of a Blunt Body," *Izv. SSSR Mekh. Zhidkosii Gasa*, Vol. 3, pp. 129-136, 1978b.

Biberman, L. M., S. Y. Bronin, and M. V. Brykin. "Moving of a Blunt Body Through the Dense Atmosphere Under Conditions of Severe Aerodynamic Heating and Ablation," *Acta Astronautica*, Vol. 7, pp. 53-65, 1980.

Brandt, P. B. "Partition Functions as They Pertain to Equilibrium Thermodynamic Properties of Gases," *Chance Vought Research Center Report No. RE-OR-2*, Dallas, Texas: Chance Vought Research Center, December 1959.

Bronshten, V. A. *Physics of Meteoric Phenomena*. D. Reidel Publishing Co.: Hingham, Massachusetts, 1983.

Chyba, Christopher F. "Terrestrial Mantle Siderophiles and the Lunar Impact Record," *Icarus*, Vol. 92, pp. 217-233, 1991.

Chyba, Christopher F., Paul J. Thomas, and Kevin J. Zahnle. "The 1908 Tunguska Explosion: Atmospheric Disruption of a Stony Asteroid," *Nature*, Vol. 361, pp. 40-44, January 1993.

Dodd, R. T. *Meteorites: A Petrological-Chemical Synthesis*. Cambridge University Press: New York, 1981.

Fay, James A., W. Craig Moffatt, and Ronald F. Probstein. "An Analytical Study of Meteor Entry," *AIAA Journal*, Vol. 2, No. 5, May 1964.

Fought, Richard M. *A Computer Model and Parametric Study of Earth-Atmospheric Entry of Meteors*. Graduate Thesis, The University of Tennessee: Knoxville, 1995.

Fought, Richard M., Roop N. Gupta, and James E. Lyne. "Extension of Curve Fits for Enthalpy of Equilibrium Air to 1000 Atmospheres," *AIAA Journal of Thermophysics and Heat Transfer*, Vol. 11, No. 1, pp. 126-128, January 1997.

Gehrels, Tom [Ed.]. *Hazards Due To Comets and Asteroids*. The University of Arizona Press: Tucson, Arizona, 1994.

Goulard, R. "Preliminary Estimates of Radiative Transfer Effects on Detached Shock Layer," *AIAA Journal*, Vol. 2, pp. 494-502, 1964.

Green, M. J. and W. E. Nicolet. "Aerothermodynamic Environment for Jovian Entry with Silica Heat Shield," *AIAA Paper 78-186*, paper presented at the 16th AIAA Aerospace Sciences Meeting, Huntsville, Alabama, January 1978.

Gupta, Roop N. "Navier-Stokes and Viscous Shock-layer Solutions for Radiating Hypersonic Flows," *AIAA Paper 87-1576*, June 1987.

Gupta, Roop N., Jerrold M. Yos, Richard A. Thompson, and Kam-Pui Lee. "A Review of Reaction Rates and Thermodynamic and Transport Properties for an 11-Species Air Model for Chemical and Thermal Nonequilibrium Calculation to 30,000 K," *NASA Reference Publication 1232*, 1990.

Gupta, Roop N., Kam-Pui Lee, Richard A. Thompson, and Jerrold M. Yos. "Calculations and Curve Fits of Thermodynamic and Transport Properties for Equilibrium Air to 30,000 K," *NASA Reference Publication 1260*, October 1991.

Hills, Jack G. and M. Patrick Goda. "The Fragmentation of Small Asteroids in the Atmosphere," *The Astronomical Journal*, Vol. 105, No. 3, pp. 1114-1144, March 1993.

Jackson, Kenneth R, Thomas E. Hull, and Wayne H. Enright. Dverk, www.netlib.org, 1976.

Lyne, J. Evans and Michael E. Tauber. "An Analysis of the Tunguska Event," *Nature*, Vol. 375, pp. 638-639, June 1995.

Lyne, J. Evans, Michael E. Tauber, and Rich M. Fought. "An Analytical Model of the Atmospheric Entry of Large Meteors and Its Application to the Tunguska Event," *The Journal of Geophysical Research*, Vol. 101, No. E10, pp. 23207-23212, October 1996.

Lyne, J. Evans, Michael E. Tauber, and Rich M. Fought. "A Computer Model of the Atmospheric Entry of the Tunguska Object," accepted by *Planetary and Space Science*, 1997.

Seeds, Michael A. *Foundations of Astronomy*. Wadsworth Publishing Company: Belmont, California, 1990.

Sekanina, Z. "The Tunguska Event: No Cometary Signature in Evidence," *The Astronomical Journal*, Vol. 88, No. 9, pp. 1382-1414, September 1983.

Srinivasan, S., J. C. Tannehill, and K. J. Weilmuenster. "Simplified Curve Fits for the Thermodynamic Properties of Equilibrium Air," *NASA Reference Publication 1181*, August 1987.

Stone, Richard. "The Last Great Impact on Earth," *Discover*, Vol. 17, No. 9, pp. 60-71, September 1996.

Sutton, K. and L. C. Hartung. "Equilibrium Radiative Heating Tables for Earth Entry," *NASA Technical Memorandum 102652*, May 1990.

Svetsov, V. V., I. V. Nemtchinov, and A. V. Teterev. "Disintegration of Large Meteoroids in Earth's Atmosphere: Theoretical Models," *Icarus*, Vol. 116, pp. 131-153, March 1995.

Tannehill, J. C. and P. H. Mugge. "Improved Curve Fit for the Thermodynamic Properties of Equilibrium Air Suitable for Numerical Computation Using Time-Dependent or Shock-Capturing Methods," *NASA Conference Report CR-2470*, October 1974.

Tauber, Michael E. "Atmospheric Entry into Jupiter," *Journal of Spacecraft and Rockets*, Vol. 6, No. 10, pp. 1103-1109, October 1969.

Thomas, P. J., L. Brookshaw, and G. Starkey. "Deformation of Large Bolides During Atmospheric Passage: Comparison of SPH and Analytical Models," *Lunar and Planetary Science XXV*, pp. 1397-1398, 1994.

Turco, R. P., O. B. Toon, C. Park, R. C. Whitten, J. B. Pollack, and P. Noerdlinger. "An Analysis of the Physical, Chemical, Optical, and Historical Impacts of the 1908 Tunguska Meteor Fall," *Icarus*, Vol. 50, pp. 1-51, 1982.

Vinh, Nguyen X., Adolf Busemann, and Robert D. Culp. *Hypersonic and Planetary Entry Flight Mechanics*. The University of Michigan Press: Ann Arbor, 1980.

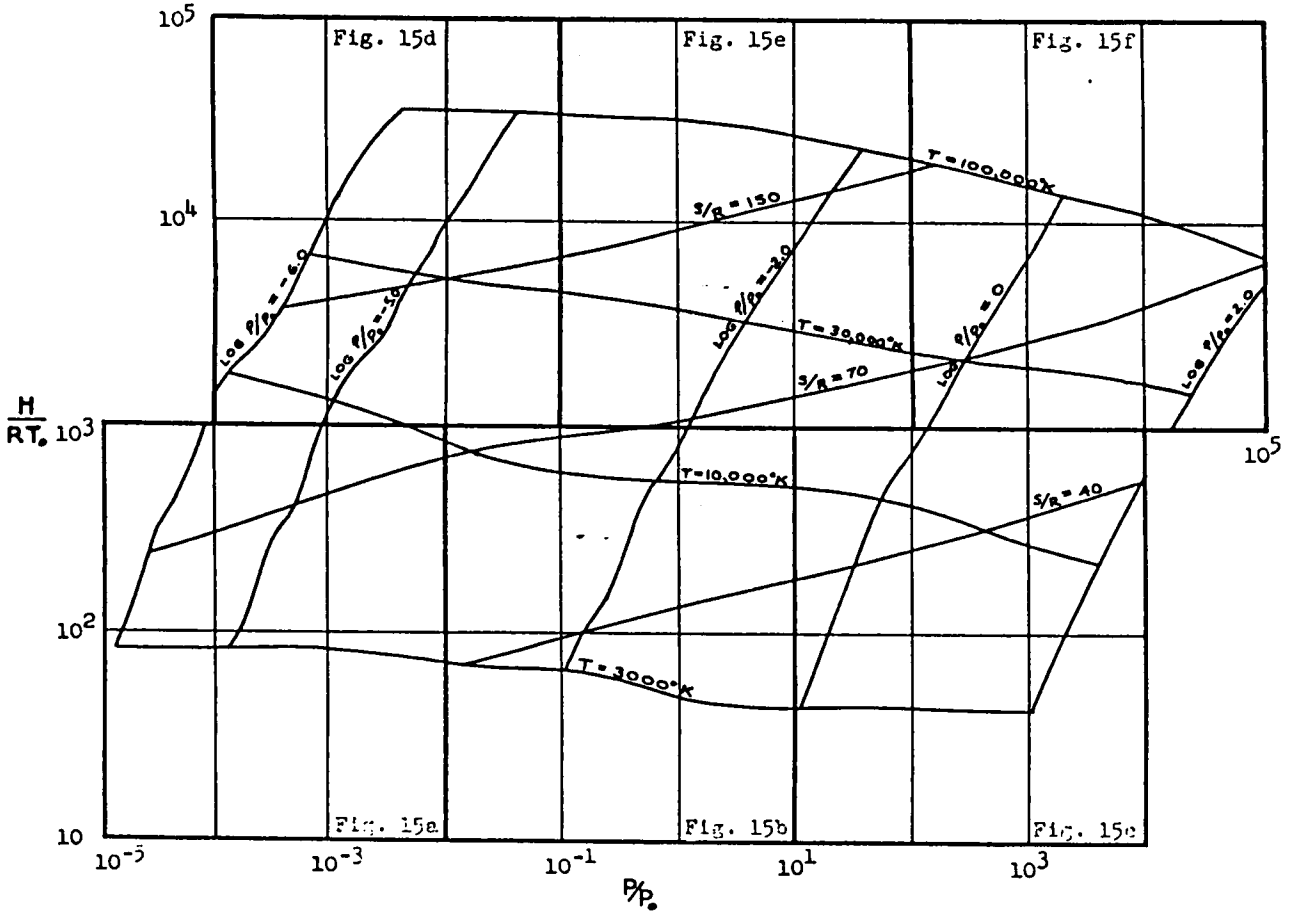
Wiesel, William E. *Spaceflight Dynamics*. McGraw-Hill: New York, 1989.

APPENDICES

APPENDIX A

Chance Vought Graphs

(Source: Anonymous (Sewell, K. G.). "Thermodynamics Properties of High Temperature Air," *Chance Vought Research Center Report No. RE-1R-14*, Dallas, Texas: Chance Vought Research Center, June 1961.)


Fig. 15d
Fig. 15e
Fig. 15f
Fig. 15a
Fig. 15b
Fig. 15c
10^5
10^4
10^3
10^2
10
$\frac{H}{RT_o}$
10^{-5}
10^{-3}
10^{-1}
10^1
10^3
10^5
P/P$_o$
T = 100,000°K
T = 30,000°K
T = 10,000°K
T = 3000°K
S/R = 130
S/R = 70
S/R = 40
Log ρ/ρ$_o$ = -6.0
Log ρ/ρ$_o$ = -4.0
Log ρ/ρ$_o$ = -2.0
Log ρ/ρ$_o$ = 0
Log ρ/ρ$_o$ = 2.0

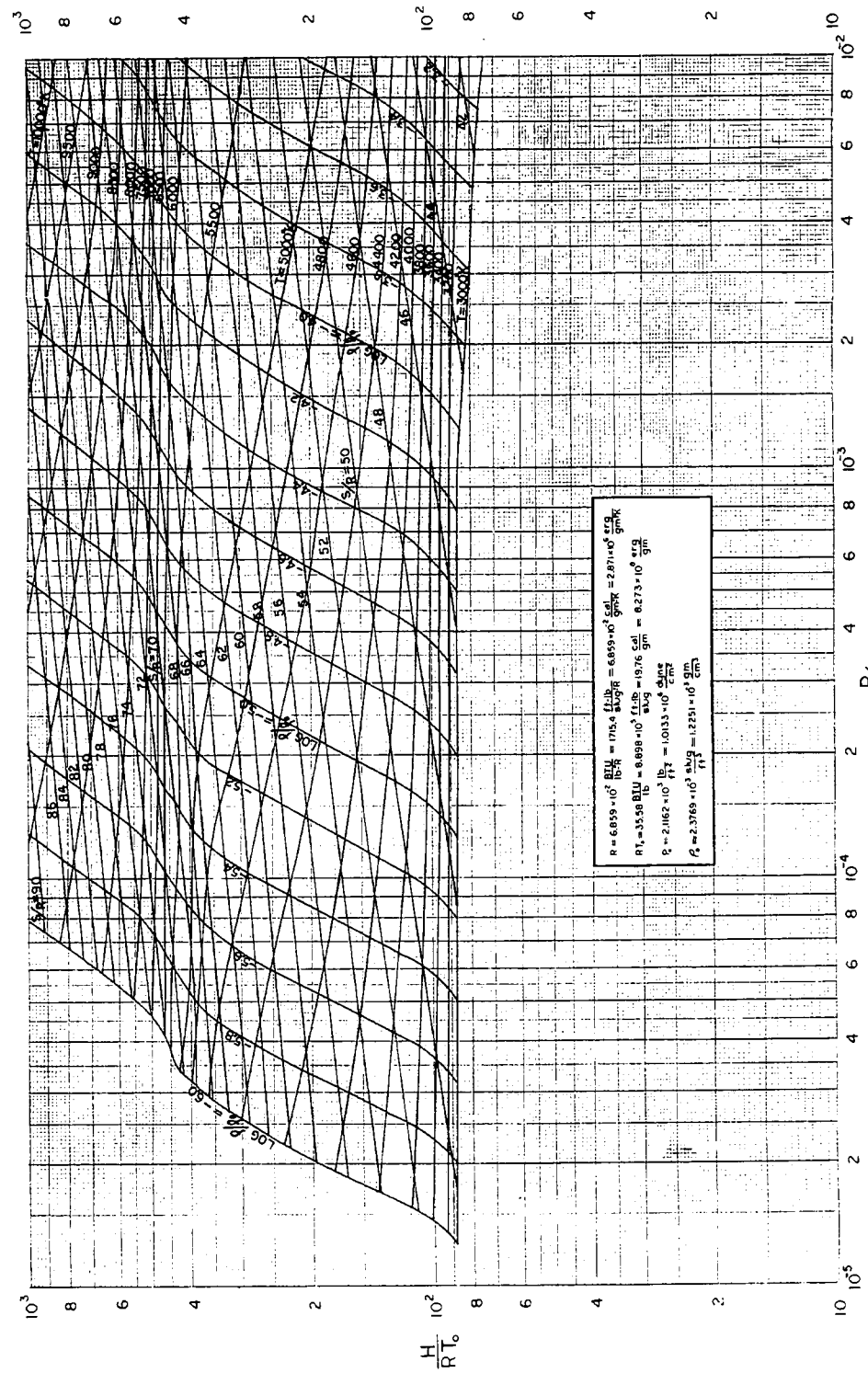


FIG. 15a THERMODYNAMIC STATE PROPERTIES OF AIR

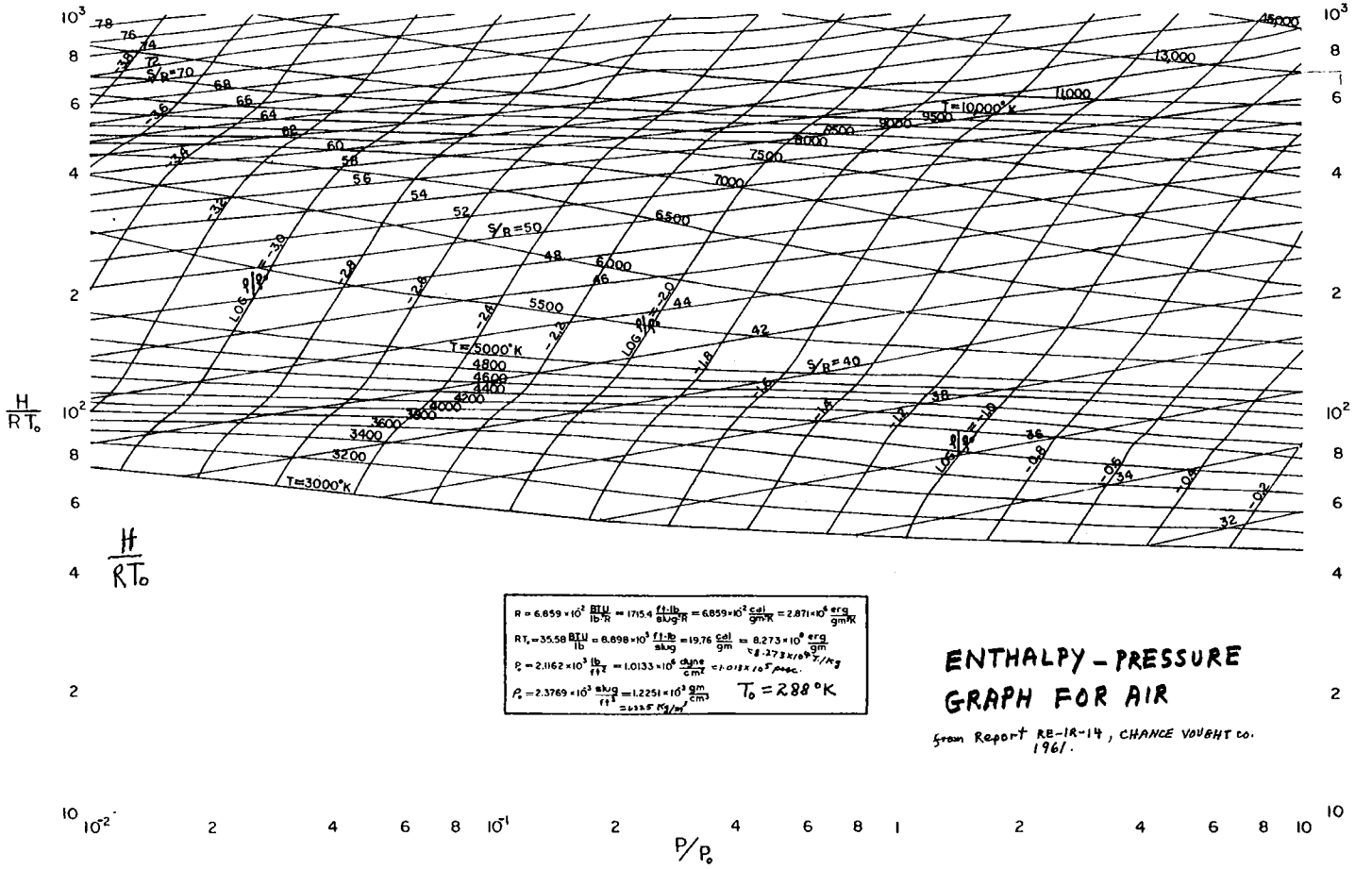


FIG. 15b THERMODYNAMIC STATE PROPERTIES OF AIR $10 < \frac{H}{RT_0} < 10^3$; $10^{-2} < \frac{P}{P_0} < 10$

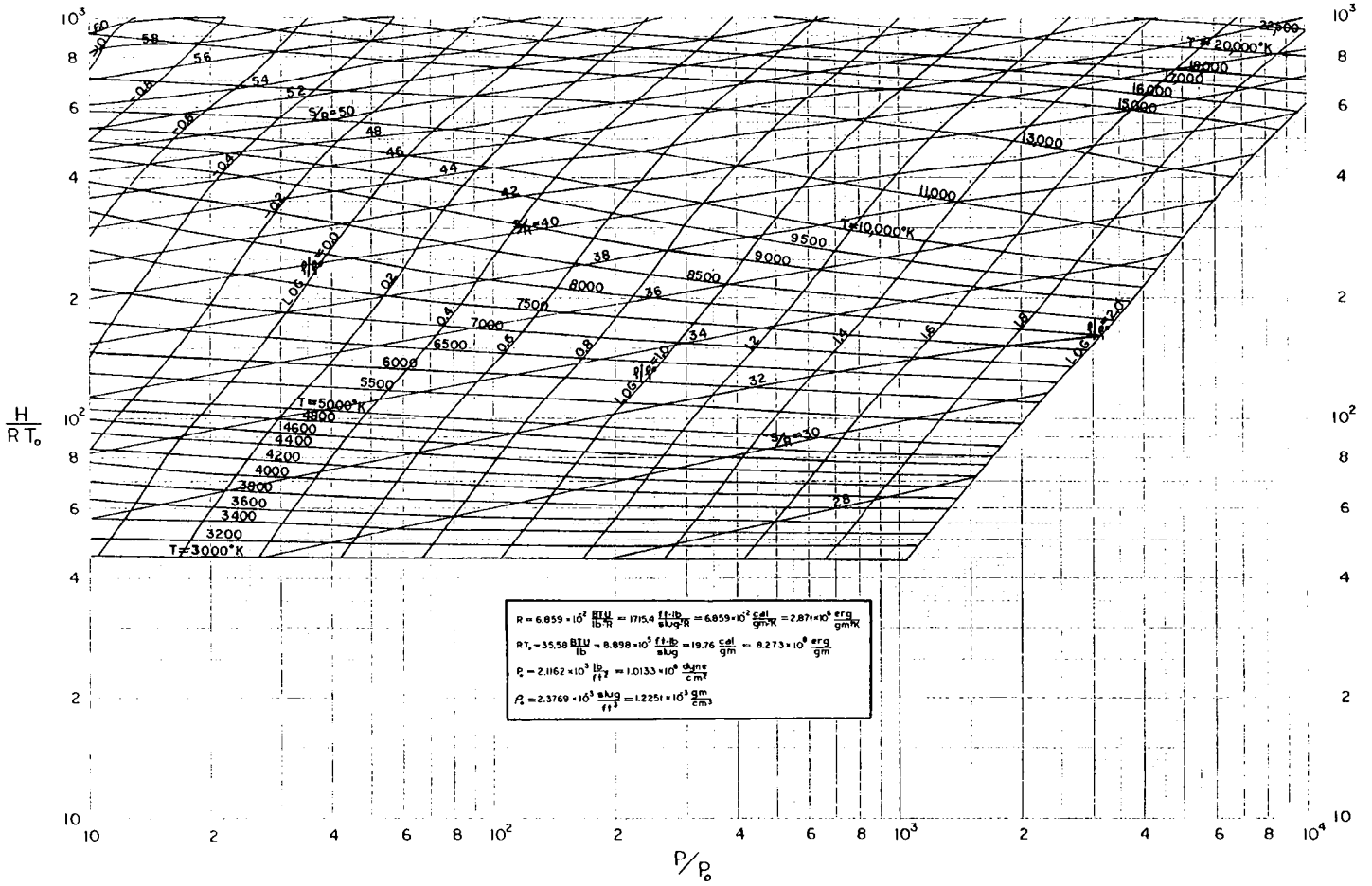


FIG. 15c THERMODYNAMIC STATE PROPERTIES OF AIR $10 < \frac{H}{RT_0} < 10^3$; $10 < \frac{P}{P_0} < 10^4$

FIG. 15c

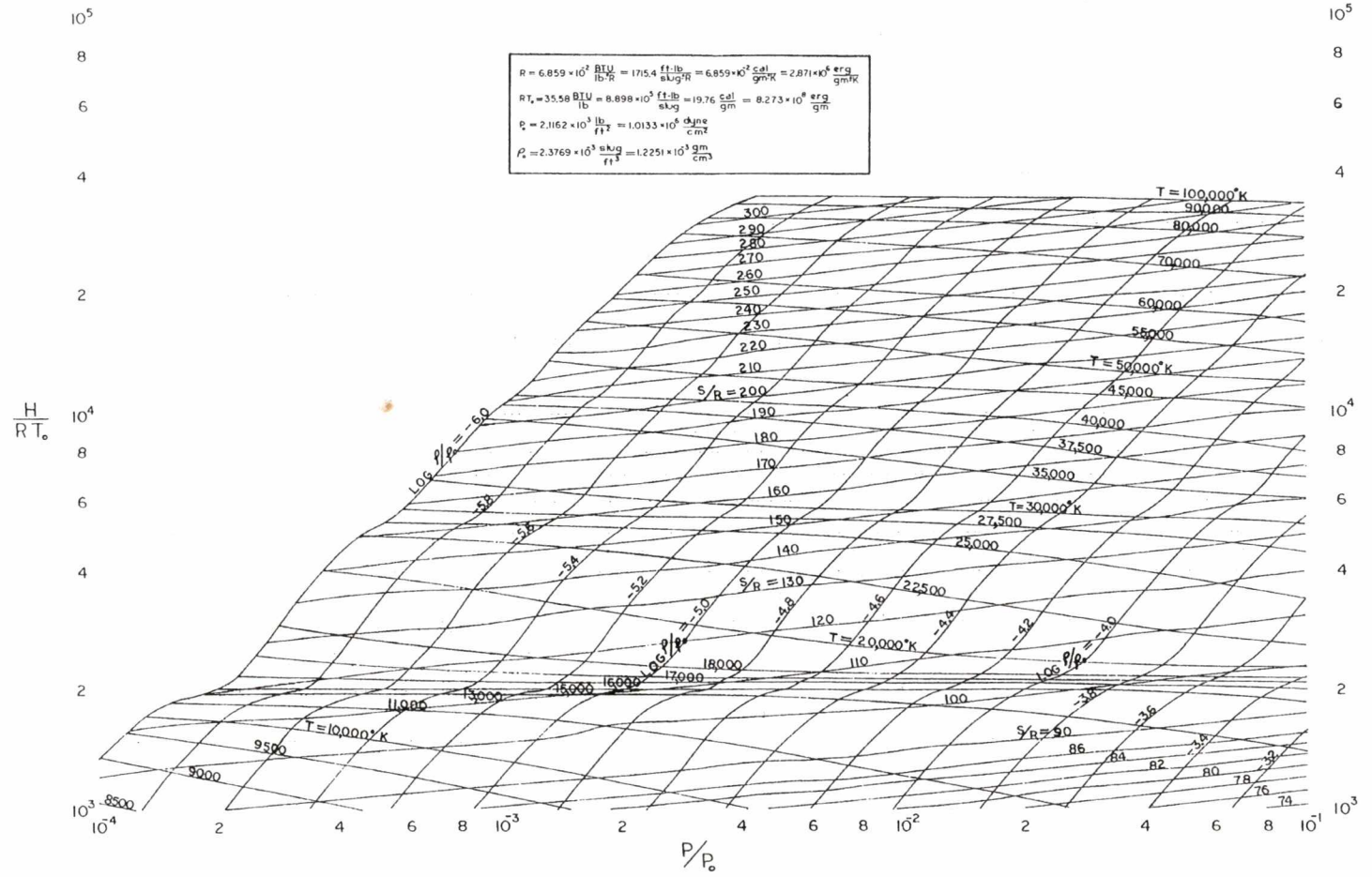


FIG. 15d THERMODYNAMIC STATE PROPERTIES OF AIR $10^{3} < \frac{H}{RT_0} < 10^{5}$; $10^{-4} < \frac{P}{P_0} < 10^{1}$

FIG. 15d

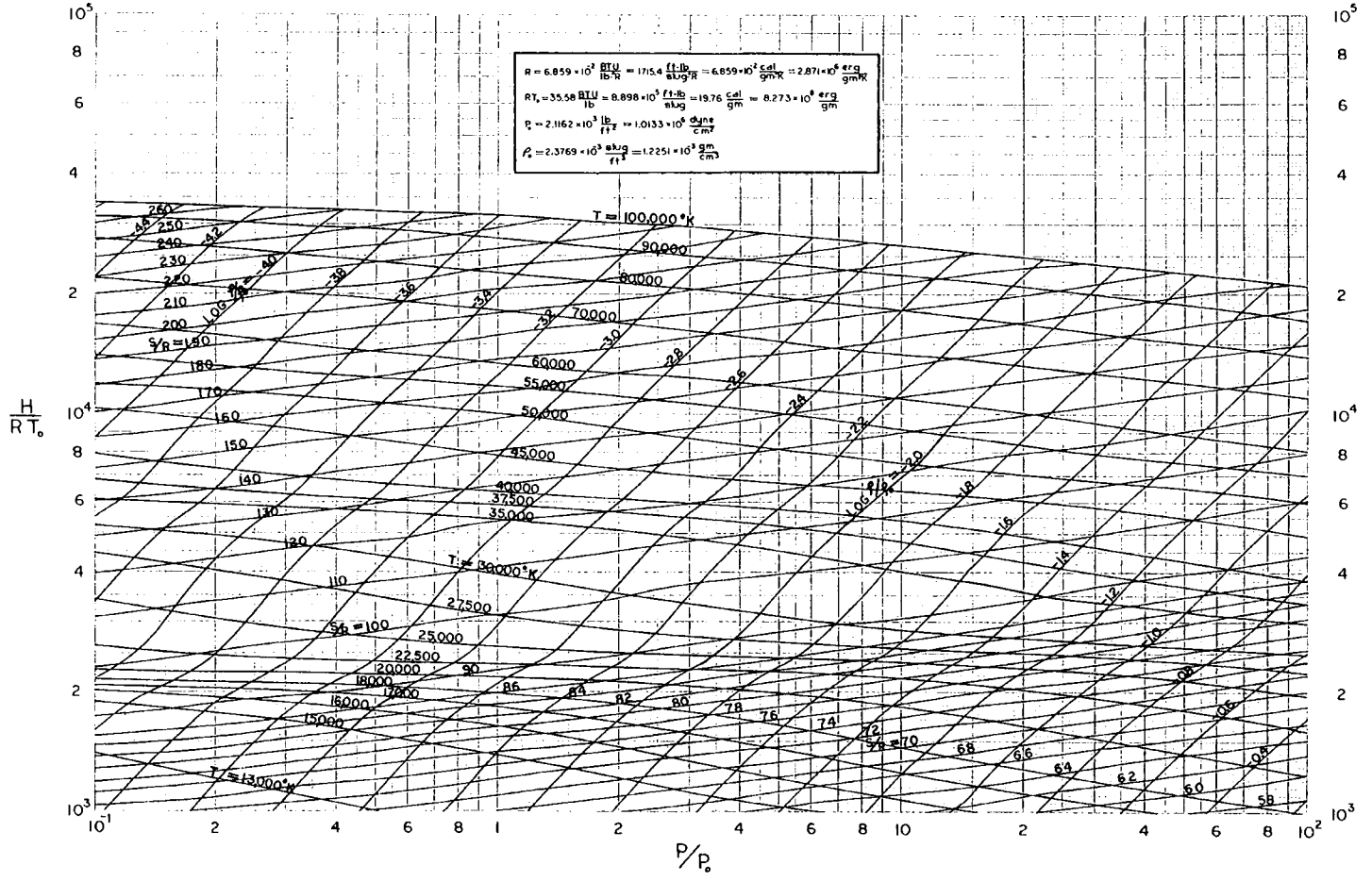


FIG. 15c THERMODYNAMIC STATE PROPERTIES OF AIR $10^3 < \frac{H}{RT_0} < 10^5$; $10^{-1} < \frac{P}{P_0} < 10^2$

FIG. 15e

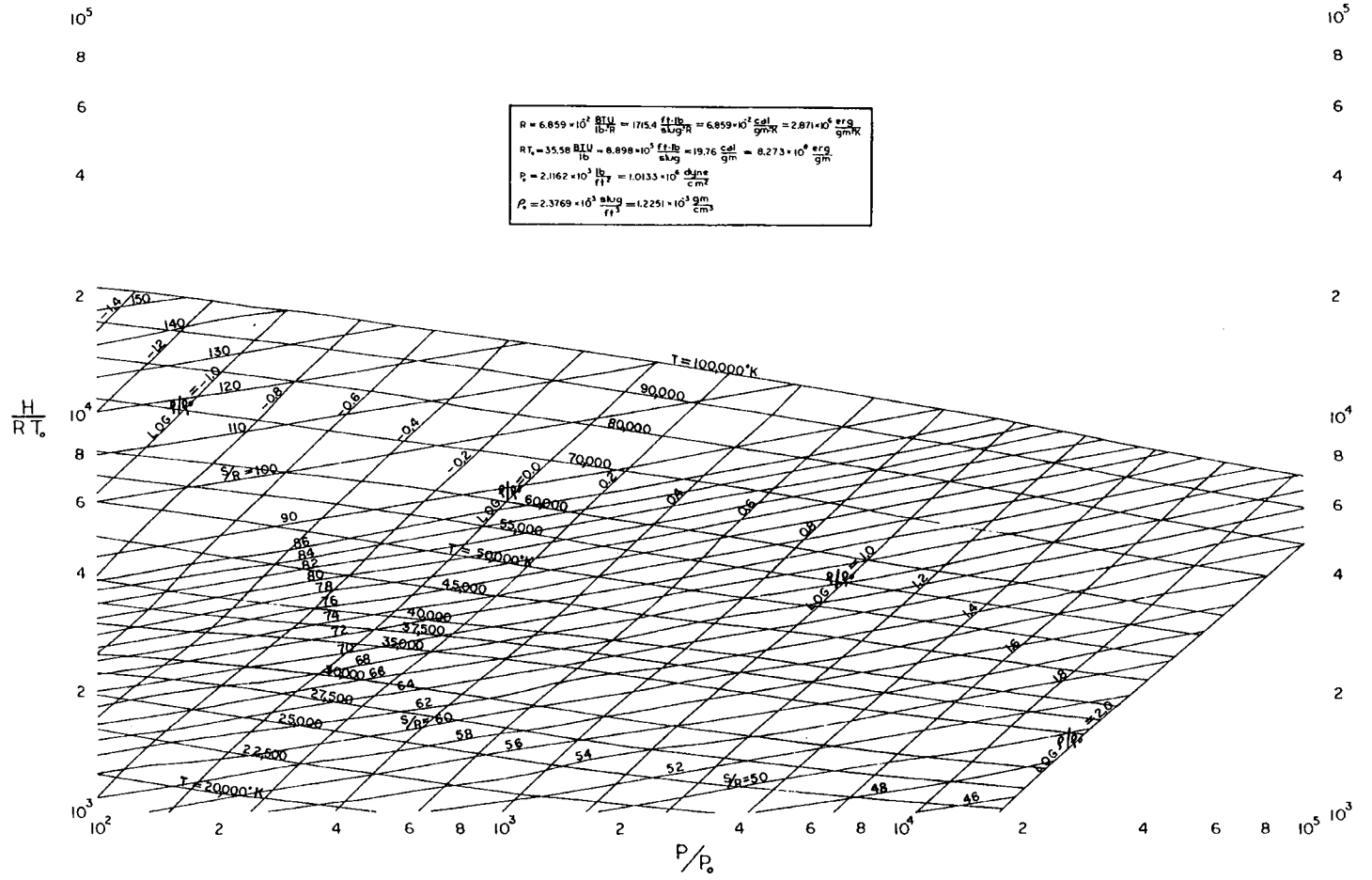


FIG. 15f THERMODYNAMIC STATE PROPERTIES OF AIR $10^3 < \frac{H}{RT_0} < 10^5$; $10^2 < \frac{P}{P_0} < 10^5$

FIG. 15f

APPENDIX B

Chance Vought Data Tables

Pressure Data (MPa)

Enthalpy (MJ/kg)

Density (kg/m^3)	3.722903	7.445805	8.273117	9.514084	12.40968	16.54623	28.95591	37.22903	41.36558	45.50214	57.91182	82.73117	124.0968	140.643
1.23E-06		1.32E-06	1.46E-06	1.58E-06	1.79E-06	2.07E-06	2.89E-06	3.85E-06	4.66E-06	5.12E-06	6.23E-06	7.93E-06	1.12E-05	1.3E-05
1.23E-05		1.34E-05	1.5E-05	1.67E-05	1.94E-05	2.25E-05	3.12E-05	4.09E-05	4.86E-05	5.51E-05	6.7E-05	8.71E-05	0.000125	0.000142
1.94E-05		2.14E-05	2.41E-05	2.67E-05	3.1E-05	3.62E-05	5E-05	6.48E-05	7.9E-05	8.97E-05	0.000107	0.000141	0.000197	0.000219
4.88E-05		5.37E-05	6.08E-05	6.89E-05	8E-05	9.22E-05	0.00013	0.000166	0.000203	0.000228	0.000282	0.000365	0.000522	0.000608
0.000123		0.000137	0.000154	0.000176	0.000208	0.000242	0.000339	0.000436	0.000517	0.000583	0.000735	0.000968	0.001368	0.001581
0.001225		0.001419	0.001621	0.001864	0.002189	0.002604	0.003729	0.004681	0.005472	0.006181	0.008005	0.01074	0.015807	0.018036
0.012251		0.014996	0.016516	0.019252	0.02371	0.028371	0.040733	0.050663	0.058262	0.065861	0.08714	0.119564	0.176306	0.20265
0.12251	0.101325	0.16212	0.176306	0.19961	0.250273	0.301949	0.44887	0.562354	0.628215	0.699143	0.942323	1.347623	2.0265	2.330475
1.2251	1.05378	1.722525	1.874513	2.127825	2.664848	3.333593	4.954793	6.16056	6.839438	7.548713	10.1325	15.4014	23.1021	26.54715
3.077312	2.654715	4.387373	4.82307	5.420888	6.8901	8.663288	12.86828	16.00935	17.93453	19.75838	26.3445	39.51675	60.795	69.40763
12.251	10.84178	18.2385	19.65705	22.49415	28.06703	35.86905	54.7155	69.10365	74.9805	82.57988	110.4443	167.1863	257.3655	293.8425
122.51	106.3913	189.9844	207.7163	235.5806	298.9088	385.035	597.8175	749.805	830.865	911.925	1185.503	1783.32	2735.775	3141.075

Enthalpy (MJ/kg)

Density (kg/m^3)	148.9161	153.0527	157.1892	165.4623	177.872	206.8279	330.9247	372.2903	413.6558	455.0214	496.387	827.3117	1654.623	2895.591
1.23E-06	1.4E-05	1.47E-05	1.56E-05	1.87E-05	2.2E-05	2.53E-05	3.67E-05	4.13E-05	4.66E-05	5.51E-05	6.08E-05	9.73E-05	0.000198	0.000428
1.23E-05	0.000153	0.00016	0.000168	0.000203	0.000236	0.000278	0.000405	0.000455	0.000507	0.000581	0.000659	0.001054	0.002148	0.004357
1.94E-05	0.000237	0.000251	0.000265	0.000327	0.000375	0.000451	0.000648	0.00073	0.000811	0.000942	0.001084	0.001702	0.003242	0.007093
4.88E-05	0.000659	0.000679	0.000709	0.000811	0.000952	0.001165	0.001702	0.001905	0.002158	0.002442	0.002736	0.004458	0.008917	0.017732
0.000123	0.001712	0.001773	0.001864	0.002027	0.002452	0.00303	0.004509	0.005016	0.005573	0.006282	0.007295	0.011652	0.024014	
0.001225	0.019252	0.01986	0.020772	0.022697	0.025838	0.033032	0.050561	0.056235	0.062619	0.069914	0.077007	0.128683	0.263445	
0.012251	0.216836	0.226968	0.233048	0.250273	0.278644	0.354638	0.559314	0.628215	0.699143	0.77007	0.856196	1.438815	2.938425	
0.12251	2.482463	2.563523	2.654715	2.8371	3.110678	3.85035	6.332813	7.09275	7.90335	8.71395	9.625875	16.51598	33.43725	
1.2251	28.16835	29.38425	30.19485	31.91738	34.95713	42.45518	70.9275	80.04675	88.65938	97.272	107.4045	182.385		
3.077312	74.47388	77.007	83.0865	84.09975	92.0031	111.4575	182.385	207.7163	232.5409	257.3655	278.6438	490.413		
12.251	314.1075	324.24	333.3593	359.7038	385.035	464.0685	770.07	871.395	972.72	1074.045	1175.37	2026.5		
122.51	3343.725	3465.315	3597.038	3774.356	4164.458	4893.998	8207.325	9423.225	10537.8	11652.38	12716.29			

Enthalpy (MJ/kg)

Pressure (MPa)

Temperature (K)	2.03E-06	5.07E-06	1.01E-05	2.03E-05	5.07E-05	0.000101	0.000203	0.000507	0.001013	0.002027	0.005066	0.010133	0.020265	0.050663	0.101325	0.20265	0.506625
3000	7.280343	7.263797	7.197612	7.176929	7.156246	7.11488	6.76741	6.453031	6.122106	5.708451	5.046601	4.798408	4.426117	4.21929	4.053827	3.971096	3.82218
3200	7.677452	7.611267	7.594721	7.528536	7.462351	7.445805	7.363074	7.123154	6.932872	6.74259	6.204838	5.791182	5.33616	4.823227	4.59158	4.450937	4.285475
3400	8.273117	8.107654	7.983558	7.942192	7.925646	7.842915	7.77673	7.627814	7.594721	7.429259	7.11488	6.76741	6.287569	5.609173	5.294795	5.038328	4.773588
3600	9.307256	8.893601	8.6785	8.438579	8.314482	8.273117	8.206932	8.091108	8.024923	7.917373	7.75191	7.445805	7.11488	6.577128	5.956644	5.625719	5.402345
3800	11.25144	10.3414	9.348622	9.265891	8.934966	8.769504	8.670226	8.52131	8.479945	8.289663	8.14902	7.942192	7.793276	7.363074	6.941145	6.444758	5.989737
4000	14.47795	12.49241	11.00325	10.58959	9.886375	9.431353	9.265891	9.017697	8.893601	8.686773	8.562676	8.479945	8.355848	8.024923	7.693999	7.263797	6.825321
4200		16.13258	14.14703	12.82333	11.33417	10.50686	10.05184	9.679547	9.431353	9.265891	9.017697	8.852235	8.769504	8.604041	8.273117	7.88428	7.263797
4400		21.92376	18.53178	16.04985	13.73337	12.40968	11.37554	10.58959	10.30003	9.845009	9.514084	9.431353	9.265891	9.100428	8.852235	8.479945	7.925646
4600		28.04587	23.99204	20.68279	16.95989	15.22253	13.56791	11.91329	11.16871	10.75505	10.25866	10.01047	9.679547	9.514084	9.307256	9.059063	8.65368
4800		31.8515	30.03141	25.89486	21.5101	18.86271	16.54623	13.98157	12.82333	11.91329	11.08598	10.67232	10.25866	10.01047	9.679547	9.431353	9.100428
5000		33.75432	32.927	31.43784	26.63944	23.33019	20.35187	17.45628	14.89161	13.60928	12.32694	11.58236	11.04461	10.50686	10.3414	10.01047	9.679547
5500		36.56718	35.65713	35.40894	34.99528	33.34066	31.10692	26.88763	23.82658	20.26914	17.20808	15.55346	14.0643	12.82333	12.24421	11.66509	11.14389
6000		38.38726	38.30453	38.2218	37.89087	37.31176	36.56718	34.91255	32.927	30.11415	25.4812	22.33742	19.19363	16.4635	14.89161	13.9402	12.82333
6500		41.36558	40.45554	39.79369	39.62823	39.2973	38.88365	38.2218	37.22903	35.98806	33.50612	30.44507	26.63944	22.25468	19.69002	17.62174	15.63619
7000		46.16399	43.76479	43.02021	42.11016	41.36558	40.95193	40.29008	39.71096	38.46999	37.64268	36.40171	34.00251	29.78322	25.81212	22.75107	19.44182
7500			51.29332	47.98408	45.58487	44.17844	43.51659	42.1929	41.77924	41.36558	40.29008	39.5455	38.46999	35.5744	32.34789	29.12137	24.65389
8000			65.19216	58.57367	52.69975	49.30778	46.9913	45.08849	44.67483	43.84752	43.43386	42.02743	41.11739	39.5455	37.22903	34.78846	30.5278
8500			86.86773	74.45805	63.53754	57.91182	53.03068	49.30778	47.90135	46.32945	45.50214	43.84752	43.02021	42.02743	40.70373	39.13184	35.73986
9000			112.1007	98.03643	81.07654	71.1488	63.20661	55.26442	52.69975	49.6387	47.98408	46.32945	45.9158	44.09571	43.43386	42.44109	39.79369
9500				121.2012	103.414	90.34244	77.7673	65.77128	57.49816	55.84354	52.12064	49.6387	48.39773	45.99853	45.50214	44.5921	43.02021
10000				139.8157	123.6831	110.0325	96.38181	79.58738	70.0733	64.11666	57.08451	53.77526	51.70698	48.64593	47.57042	46.32945	45.33668
11000				153.88	148.9161	143.5386	131.9562	115.41	100.1047	87.28138	74.87171	67.4259	61.63472	56.009	53.19614	51.45879	49.47324
13000				163.8077	162.5667	162.1531	161.3258	157.3547	153.0527	144.7795	129.0606	115.8236	100.1047	82.73117	75.28536	68.17048	62.21384
15000				169.1852	167.5306	167.117	166.7033	166.2896	165.876	165.4623	161.3258	155.9483	146.0205	128.2333	114.169	100.1047	85.2131
16000					177.872	174.5628	174.1491	173.7355	172.0808	171.2535	169.5989	165.876	160.4985	148.0888	134.8518	119.9602	99.93925
17000					191.5227	184.0768	180.3539	179.113	177.872	175.3901	174.5628	171.6672	169.1852	161.3258	150.9844	137.9956	119.1329
18000					221.7195	206.0006	193.5909	185.7315	183.6632	181.1813	177.872	176.2174	175.3901	170.0126	164.635	153.4663	136.5064
20000					322.6516	290.3864	259.7759	226.6834	208.4825	197.7275	190.2817	187.3861	186.1451	182.0086	179.113	174.5628	164.635
22500					405.3827	391.3184	371.4629	330.9247	297.0049	264.7397	229.9926	215.5147	203.105	197.7275	194.4182	190.2817	185.7315
25000					441.7844	434.3386	426.8928	412.8285	390.9048	364.0171	316.0331	281.286	252.3301	226.2697	215.101	206.8279	201.4504
27500						464.9492	455.4351	447.9893	438.4752	425.2382	397.9369	368.1537	332.993	288.7318	256.4666	236.1975	220.4786
30000						543.9574	509.624	479.0135	470.7403	464.5355	446.7483	431.8567	407.0373	364.0171	324.3062	292.041	256.4666
35000						802.4923	759.4721	679.2229	627.1023	569.1904	528.6522	503.0055	488.1139	470.7403	447.5756	421.929	373.1176
37500						875.2958	858.7495	795.8738	744.5805	690.8053	610.9697	562.5719	529.4795	504.6601	488.1139	466.6038	431.0294
40000						938.9988	914.1794	872.8138	839.7214	798.3558	721.4158	656.0582	599.801	553.0579	527.4112	507.5557	475.7042
45000							1083.778	1009.32	963.8181	941.4807	901.7697	858.3359	795.8738	711.488	647.3714	602.2829	554.2988
50000							1394.02	1282.333	1195.465	1108.598	1025.866	992.774	951.243	881.0869	827.3117	761.1267	676.741
55000							1609.121	1534.663	1460.205	1373.337	1240.968	1162.373	1092.051	1025.866	967.9547	914.1794	831.4482
60000								1741.491	1662.896	1604.985	1489.161	1389.884	1307.152	1187.192	1108.598	1046.549	967.9547
70000								2287.517	2192.376	2076.552	1915.227	1820.086	1716.672	1609.121	1505.707	1399.811	1257.514
80000								2564.666	2548.12	2481.935	2370.248	2287.517	2163.42	2002.094	1861.451	1770.447	1621.531
90000								2779.767	2721.855	2672.217	2647.397	2589.486	2535.71	2424.023	2283.38	2151.01	1985.548
100000								2924.547	2903.864	2891.454	2829.406	2812.86	2742.538	2676.353	2630.851	2502.618	2353.702

Enthalpy (MJ/kg)

Pressure (MPa)

Temperature (K)	1.01325	2.0265	5.06625	10.1325	20.265	50.6625	101.325	126.6563	202.65	253.3125	303.975	506.625	1013.25	2026.5	4053	6079.5	8106	10132.5
3000	3.789087	3.764268	3.722903	3.722903	3.722903	3.681537	3.656718											
3200	4.153105	4.136558	4.037281	4.012462	3.987642	3.979369	3.971096											
3400	4.690857	4.616399	4.434391	4.368206	4.335113	4.302021	4.277201											
3600	5.212064	4.988689	4.798408	4.740496	4.674311	4.616399	4.558487	4.554351										
3800	5.791182	5.526442	5.278249	5.129332	5.046601	4.972143	4.955597	4.947324										
4000	6.328934	6.064195	5.799455	5.609173	5.460257	5.33616	5.286522	5.278249										
4200	6.908053	6.63504	6.271023	6.039375	5.873913	5.724997	5.625719	5.617446										
4400	7.594721	7.280343	6.908053	6.601947	6.304115	6.122106	5.99801	5.964917										
4600	8.107654	7.900827	7.387893	7.106607	6.76741	6.585401	6.436485	6.411666										
4800	8.769504	8.438579	7.983558	7.64436	7.280343	6.949418	6.76741	6.701225										
5000	9.431353	9.017697	8.52131	8.14902	7.88428	7.470624	7.27207	7.156246										
5500	10.75505	10.3414	9.762278	9.431353	9.100428	8.645407	8.273117	8.206932	8.008377									
6000	12.16148	11.66509	11.16871	10.58959	10.3414	9.845009	9.431353	9.348622	9.100428									
6500	14.47795	13.60928	12.65787	12.07875	11.58236	11.08598	10.67232	10.50686	10.32485	10.17593								
7000	17.62174	15.96712	14.64342	13.77474	13.15426	12.40968	11.83056	11.66509	11.33417	11.16871								
7500	21.92376	19.44182	16.95989	15.71892	14.89161	13.77474	13.07152	12.8647	12.49241	12.32694	12.24421							
8000	27.30129	23.78521	20.35187	18.28359	16.95989	15.55346	14.64342	14.39522	13.73337	13.56791	13.40245							
8500	32.43062	28.95591	24.61252	21.7583	19.52456	17.45628	16.38077	16.09121	15.34663	14.97434	14.76751							
9000	37.39449	34.08524	29.70049	26.06032	22.91653	20.1864	18.44905	18.03539	17.16672	16.87716	16.54623							
9500	41.11739	38.46999	34.2507	30.77599	27.13582	22.8338	20.68279	20.02094	18.94544	18.44905	18.18431							
10000	44.26117	42.27563	38.71819	34.91255	31.35511	26.39124	23.49565	22.75107	21.34464	20.72416	20.35187							
11000	48.39773	47.32223	45.41941	42.60655	39.21457	34.00251	30.5278	29.28683	27.21855	26.22578	25.4812	23.49565						
13000	59.40098	56.33993	54.35438	52.86522	51.12786	47.81862	44.67483	43.2684	40.45554	38.96638	37.80814	34.58163						
15000	76.11267	70.98334	65.27489	62.62749	59.64917	56.83631	55.01623	54.35438	51.78971	51.12786	49.96963	46.90857						
16000	88.52235	80.24923	72.63797	67.75683	64.53031	61.22106	58.73913	57.91182	56.58812	55.84354	54.76803	52.69975	47.81862					
17000	103.414	92.24525	80.99381	74.62351	70.32149	66.18493	63.53754	62.87569	60.22829	59.81463	59.56644	57.49816	53.77526					
18000	119.9602	105.8959	91.00428	82.40024	76.02994	70.98334	67.6741	66.51586	64.53031	64.11666	63.04115	61.22106	57.74636					
20000	152.2253	136.0928	115.8236	103.2485	91.8316	82.73117	77.85003	76.27814	73.46528	72.80343	72.38977	69.57691	66.18493					
22500	177.872	167.117	148.9161	132.7835	118.3056	103.0003	93.07256	90.83882	86.86773	84.79945	83.39302	81.07654	77.35364					
25000	196.4865	190.2817	177.4584	162.1531	146.4342	126.5787	112.928	109.3706	103.414	100.932	97.62278	94.31353	88.52235	84.38579				
27500	212.2054	206.4143	196.9002	185.9797	172.9081	150.1571	135.6791	130.3016	121.0357	117.8919	115.41	109.2051	102.173	95.5545				
30000	236.1975	225.8561	214.6874	206.8279	194.6664	175.3901	157.1892	152.639	141.4703	136.7546	134.0245	126.165	116.0718	108.3778				
35000	331.752	296.1776	264.3261	246.5389	232.4746	215.9283	201.864	196.9002	182.8359	177.872	173.3218	161.0776	146.4342	136.0928				
37500	395.455	349.9528	305.6917	277.1494	256.4666	235.7838	221.3059	215.101	203.105	197.7275	192.35	180.7676	163.8077	149.3298				
40000	446.7483	405.7964	354.5031	316.0331	297.0049	258.1212	240.7477	234.9565	222.1332	217.1693	212.6191	198.9685	177.0447	165.0487				
45000	529.4795	499.6963	454.1941	407.8647	364.4308	317.6877	289.5591	282.1133	265.1534	257.7076	251.0891	237.0248	216.342	196.9002	179.9403			
50000	624.6203	580.7728	541.8892	500.5236	455.0214	395.455	354.0894	340.8524	320.1696	306.1053	299.4868	281.286	257.2939	231.6473	210.9645			
55000	760.2994	693.2872	626.2749	579.1182	535.2707	475.7042	430.2021	413.2422	381.3907	370.6356	358.226	332.5793	299.0732	271.3582	244.0569			
60000	901.7697	831.4482	731.3435	670.1225	620.4838	558.4354	502.5918	483.9773	452.5395	437.6479	424.4109	391.7321	347.4709	315.2058	280.8723	264.7397		
70000	1162.373	1092.051	992.774	910.0428	827.3117	728.0343	661.8493	644.4758	603.1102	580.3591	566.7085	527.4112	463.2945	413.6558	366.0854	344.989	330.0974	
80000	1509.844	1389.884	1245.104	1149.963	1063.096	947.2719	843.8579	814.0747	760.2994	741.2713	717.2792	661.8493	594.0098	528.2385	465.3628	431.8567	412.0012	398.7642
90000	1853.178	1733.218	1563.619	1422.976	1315.426	1162.373	1067.232	1033.312	951.4084	919.9706	897.6332	826.4844	735.4801	654.4035	573.7407	535.6843	506.7284	490.1822
100000	2200.649	2055.87	1861.451	1737.355	1609.121	1414.703	1282.333	1240.968	1166.509	1125.144	1092.051	1009.32	901.7697	795.8738	694.9418	643.2348	610.556	587.3913

APPENDIX C

METEOR Program (Fortran 90)

```
PROGRAM METEOR
    IMPLICIT NONE

! Meteor Trajectory Analysis Code
! Last modified: June 3, 1997
!
! For information contact:
!
!   Dr. Evans Lyne
!   University of Tennessee, Knoxville
!   (423)974-5254


! This program was written by Evans Lyne and Rich Fought as a model
! of meteor entry. It has since been modified by Chris Thames to
! allow modeling of meteors traveling at greater speeds. Changes were
! also made to account for a density term in the pressure equation
! that was previously assumed negligible. Finally, the code was
! converted from FORTRAN 77 to Fortran 90. For a more thorough
! description see Rich Fought's thesis and Chris Thames' thesis
! in the University of Tennessee, Knoxville, library.

!     shock angle is determined using the method of FOUGHT & PALMER
!     shock angle at shoulder is 15 degrees. The code assumes a
!     cylinder of diameter 2r and length r with a hemispherical
!     nosecap at each end. Radiation is applied to the hemisphere.

!     VARIABLES
!
!   AA          coefficient                 m^2/kg
!   ablfrac     mass ablated to total mass  -
!   ablmass     mass lost to ablation       kg
!   alton       max altitude of atm         m
!   areasc      surface area                m^2
!   balcof      ballistic coefficient       kg/m^2
!   Cd          drag coefficient            -
!   cmass       body mass                   kg
!   cmass2      body mass after ablation    kg
!   comp        comparison value            -
!   Cp          specific heat, const press  J/(kg-K)
!   d2rdt2      d(dr/dt)/dt                 m/s^2
!   d2rdt2o     previous d(dr/dt)/dt        m/s^2
!   drdt        dr/dt                       m/s
!   drdto       previous dr/dt              m/s
!   dt          time step                   s
!   diff        diff ratio in heat tran     -
!   enth_Jkg    enthalpy                    J/kg
!   enth_stag   stagnation point enthalpy   J/kg
!   g           gravitational accel         m/s^2
!   gamma       trajectory angle            degrees
!   gammae      entry trajectory angle      degrees
!   goulard     Goulard number              -
!   goulardsp   Goulard number @ stag pt    -
!   G0          Earth grav accel @ surface  m/s^2
!   h           altitude                    m
!   he          entry altitude              m
!   idiff       counter                     -
!   ind         indicator for DVERK routine -
!   ith         angle at nose               degrees
!   ithmax      max angle to sweep across   degrees
!   iwrite      counter                     -
!   J           counter                     -
!   MachNorm    Mach # of vel norm to shock -
!   mdot        mass flw rate per unit area kg/(m^2-s)
!   mdotratio   non-dim mass flow rate      -
!   n           number of diff. eqtns. for  -
!               IVPRK routine
!   p1          atmospheric pressure        Pa
!   p2          pressure behind shock       Pa
!   pflag       crunch flag                 -
!   pstag       stagnation pressure         Pa
```

```
!   pyield      body yield strength          Pa
!   qabl        heat of ablation             J/kg
!   qdotr       rad heat transfer rate       W/cm^2
!   qdotrc      temp value holder            W/cm^2
!   qdotrcad    adibatic rad heat tran rate  W/cm^2
!   qdotredg    rad heat tran rate at edge   W/cm^2
!   qdotrlim    15% max rad heat tnsf rate   W/cm^2
!   qdotrnew    temporary value holder       W/cm^2
!   qdotrold    prev rad heat transfer rate  W/cm^2
!   qdotrsp     stag pt rad heat tran rate   W/cm^2
!   qt          cumulative energy over time  J
!   qt1         energy on surf over 1 deg    J
!   qt2         cumulative energy on surf    J
!   rb          body nose radius             m
!   rhoc        body density                 kg/m^3
!   rho1        atmospheric density          kg/m^3
!   rho2        density of gas inside shock  kg/m^3
!   rho1_stag   stag pt dens front of shock  kg/m^3
!   rho2_stag   stag pt dens behind shock    kg/m^3
!   rho_old     previous density             kg/m^3
!   rho_tol     rho2 solution tolerance      -
!   rn          initial body nose radius     m
!   R0          Earth radius                 m
!   scalefac    scaling factor               -
!   spdsnd      speed of sound               m/s
!   t           time                         s
!   t1          atmospheric temperature      K
!   t2          temperature behind shock     K
!   t2sp        stagnation temperature       K
!   t2edg       temp behind shock@top edge   K
!   tprint      max time before output       s
!   thetash_rad shockwave angle              rad
!   tnext       value of 't' where solution  -
!               is desired for IVPRK routine
!   tol         tolerance for error control  -
!               for IVPRK routine
!   v           velocity                     m/s
!   ve          entry velocity               m/s
!   v_norm      norm vel in front of shock   m/s
!   xke         kinetic energy               J
!   xke1        initial kinetic energy       J
!   xkeold      previous kinetic energy      J
!   xkerate     rate of kin enrgy change     W
!   Y[3]        array containing:
!       Y(1)        velocity                 m/s
!       Y(2)        trajectory angle         rads
!       Y(3)        altitude                 m
!
!*************************************************************

    ! Common Block Declarations
    COMMON /BLOCK1/ R0, AA

    ! Constants
    DOUBLE PRECISION, PARAMETER :: PI = 3.14159265359
    DOUBLE PRECISION, PARAMETER :: Cp = 1004.0
    DOUBLE PRECISION, PARAMETER :: GAMALT = 1.40

    INTEGER, PARAMETER :: LX = 35   ! TEMP_FIT
    INTEGER, PARAMETER :: LY = 43
    INTEGER, PARAMETER :: NN = 10
    INTEGER, PARAMETER :: DX = 28   ! DENS_FIT
    INTEGER, PARAMETER :: DY = 12
    INTEGER, PARAMETER :: DN = 10

    ! External Subroutines
    EXTERNAL FCN1, DVERK

    ! Declare Variables
    INTEGER J, ind, ith, ithmax, iwrite, n, idiff
    LOGICAL pflag
```

```
      DOUBLE PRECISION enth_stag, comp, rho_tol, he, ve, rn, mdot
      DOUBLE PRECISION mdotratio, spdsnd, gammae, dt, qabl, pyield
      DOUBLE PRECISION MachNorm, alton, Cd, tprint, cmass, xkel
      DOUBLE PRECISION xke, balcof, rb, d2rdt2, d2rdt2o, drdt
      DOUBLE PRECISION drdto, qt, qt2, qdotr, qdotrold, t, v, h
      DOUBLE PRECISION R0, AA, tol, qdotrlim, rho1, qt1, v_norm
      DOUBLE PRECISION p2, p1, enth_Jkg, t1, rho_old, rhoc, rho2
      DOUBLE PRECISION thetash_rad, pstag, t2sp, t2, rho1_stag
      DOUBLE PRECISION rho2_stag, t2edg, qdotrcad, goulard, qdotrc
      DOUBLE PRECISION goulardsp, qdotrnew, scalefac, diff, pyield_MPa
      DOUBLE PRECISION qdotredg, areasc, ablmass, ablfrac, cmass2
      DOUBLE PRECISION qdotrsp, xkeold, xkerate, gamma, tnext, g

      DOUBLE PRECISION Y, C, W
      DOUBLE PRECISION XX, YY, ZZ, XXX, YYY, ZZZ

      ! Declare Arrays
      DIMENSION Y(3), C(24), W(9,27)
      DIMENSION XX(LX), YY(LY), ZZ(LX,LY)
      DIMENSION XXX(DX), YYY(DY), ZZZ(DX,DY)

      ! Loading Vought Data - pressure, enthalpy, temperature
      OPEN(UNIT=10, FILE="thermo1.dat", STATUS='OLD')
      READ(10,*) XX(1:LX)
J1:   DO J=1, LY
          READ(10,*) YY(J), ZZ(1:LX,J)
      END DO J1
      CLOSE(10)

      ! Loading Vought Data - enthalpy, pressure, density
      OPEN(UNIT=20, FILE="thermo2.dat", STATUS='OLD')
      READ(20,*) XXX(1:DX)
J2:   DO J=1, DY
          READ(20,*) YYY(J), ZZZ(1:DX,J)
      END DO J2
      CLOSE(20)

      ! Open input file and get data
      OPEN(UNIT=9,FILE='entry.dat',STATUS='OLD')
      READ(9,*) he, ve, rhoc, rn, gammae, dt, qabl, pyield_MPa
      READ(9,*) alton, Cd, tprint, rho_tol
      CLOSE(9)

      ! Open output files
      OPEN(UNIT=1, FILE="traj1.out", STATUS='NEW')
      OPEN(UNIT=2, FILE="traj2.out", STATUS='NEW')

      ! Set common block values
      R0 = 6378135.0
      cmass = rhoc * PI * 7.0/3.0 * rn**3
      balcof = cmass / (Cd*PI*rn**2)
      AA = 1.0 / (2.0*balcof)

      ! Initialize values
      n = 3
      ind = 1
      ithmax = 90
      iwrite = 0
      d2rdt2 = 0.0
      d2rdt2o = 0.0
      drdt = 0.0
      drdto = 0.0
      h = he
      pflag = .FALSE.
      pyield = pyield_MPa * 1000000.0
      qt = 0.0
      qt1 = 0.0
      qt2 = 0.0
      qdotr = 0.0
      qdotrold = 0.0
      rb = rn
```

```
    tol = 0.001
    t = 0.0
    v = ve
    xke1 = 0.5 * cmass * ve**2
    xke = xke1
    Y(1) = ve
    Y(2) = PI * (gammae/180.0)
    Y(3) = he

    ! Calculate gravitational accel, density, and viscosity at altitude 'h'
    CALL ATMOS76(h, g, rho1, t1, p1, spdsnd)

! Loop until altitude is below 1 m or kinetic energy is at less than 0.01%
TRAJ: DO

    ! *** CALCULATE HEATING INFORMATION ***

        ! Max heat transfer, 15% limit, converted to W/cm^2,
        ! from experimentation and Anderson [Hypersonic, p. 289]
        qdotrlim = 0.0001 * 0.15 * (0.5*rho1*V**3)

        ! Save previous value
        qdotrold = qdotr

        ! Step around nose of body calculating temperature and qdot
I1:     DO ith=0, ithmax, 1

            ! Calculate shock angle using an experimental value for approx. shape of shoc
k
            ! [Fought, pg. 23]
            thetash_rad = (-0.833333*(ith/90.0)*PI/2.0) + 1.5708

            ! Determine normal velocity outside shock
            v_norm = v * SIN(thetash_rad)

            ! Make an initial approximate calculation of pressure and enthalpy inside
            ! the shock
            p2 = p1 + rho1 * v_norm**2
            enth_Jkg = Cp*t1 + v_norm**2 / 2.0

            ! Initialize
            rho_old = 0.0

            ! Loop to determine correct pressure, enthalpy, density combination
PRESS:      DO

                ! One option for points using Vought curve fits if in range and
                ! the other for the rest
                IF (enth_Jkg > 7445805.12) THEN

                    ! Vought density curve fit
                    CALL DENS_FIT(p2, enth_Jkg, rho2, XXX, YYY, ZZZ, DX, DY, DN)

                ELSE

                    ! Eqtns from Anderson [Compressible, pg. 106]
                    MachNorm = v/spdsnd * SIN(thetash_rad)
                    rho2 = rho1 * ((GAMALT + 1.0)*MachNorm**2)/((GAMALT - 1.0)*MachNorm**
2 + 2.0)

                END IF

                ! Determine if density is within tolerance
                comp = ABS((rho2-rho_old)/rho2)
                IF (comp < rho_tol) THEN
                    EXIT PRESS
                END IF

                ! Save previous value
                rho_old = rho2
```

```
            ! Resolve pressure and enthalpy behind the shock [Anderson, Hypersonics,
pg. 508]
            p2 = p1 + rho1 * v_norm**2 * (1.0 - rho1/rho2)
            enth_Jkg = Cp*t1 + v_norm**2 / 2.0 * (1.0 - (rho1/rho2)**2)

        END DO PRESS

        ! Determine whether to use max limit b/c atm is too thin for reasonable value
s
        IF (h > alton) THEN

            qdotr = qdotrlim

        ELSE

            ! Determine temperature inside shock using either Vought or Fought curve
fit
            IF (enth_Jkg > 7445805.12) THEN

                ! Vought temperature curve fit
                CALL TEMP_FIT(enth_Jkg, p2, t2, XX, YY, ZZ, LX, LY, NN)

            ELSE

                ! Fought curve fit
                CALL TEMPNEW(enth_Jkg, p2, t2)

            END IF

            ! Print time, angle, and temp if time, t, exceeds tprint
            IF (t > tprint) THEN
                WRITE(6,*) t, ith, t2
            END IF

            ! ** USE GOULARD NUMBER TO ACCOUNT FOR RADIATIVE COOLING OF SHOCK LAYER *
*

            ! Apply Stefan-Boltzman equation [Incopera, pg. 10]
            qdotrcad = 5.67 * t2**4 / (1.0e12)

            ! Modify Stefan-Boltzman result according to Goulard number
            ! from Lyne [Geophysical Research, pg. 23208]
            goulard = 10000.0 * 4.0 * qdotrcad/(rho1*v**3)
            qdotrc = qdotrcad / (1.0 + 3.33*goulard**0.7)
            qdotr = qdotrc

            ! Determine stagnation point and edge values
            IF (ith == 0) THEN

                pstag =  rho1 * V**2 * (1.0 - rho1/rho2)
                t2sp = t2
                enth_stag = enth_Jkg
                rho1_stag = rho1
                rho2_stag = rho2
                goulardsp = goulard

            ELSE IF (ith == ithmax) THEN

                t2edg = t2

            END IF

            ! APPLY 15% FLOW ENERGY LIMIT
            IF (qdotrc > qdotrlim) THEN
                qdotr = qdotrlim
            END IF

            ! ** APPLY GREEN AND NICOLET'S CORRECTION TO ACCOUNT FOR BLOWING **
            ! ** OF WALL CAUSING REDUCED RADIATIVE HEATING (FOR SIO2)        **

            ! qdotr is in W/cm^2, heat of ablation is in Joules/kg. This
```

```
                ! results in a conversion factor of 10000
                qdotrnew = qdotrc
                idiff = 0

                ! Loop until (diff > 0.02) and (idiff < 200)
DIFF1:          DO
                    mdot = (qdotrnew/qabl) * 10000.0
                    mdotratio = mdot / (rho1*V)
                    IF (mdotratio < 0.04802734866) THEN
                        scalefac = 1.0
                    ELSE IF (mdotratio > 0.754793198787) THEN
                        scalefac = 0.5
                    ELSE
                        scalefac = 1.0095555-0.11552317*mdotratio &
                            -1.7206431*mdotratio**2-0.53616033*mdotratio**3 &
                            +4.1602124*mdotratio**4-2.2932833*mdotratio**5
                    END IF

                    qdotrold = qdotrnew
                    qdotrnew = scalefac * qdotr
                    diff = ABS(qdotrold - qdotrnew) / qdotrold
                    idiff = idiff + 1

                    IF (idiff == 1000) THEN
                        WRITE(6,*)202020
                    END IF

                    IF ((diff < 0.02) .OR. (idiff > 200)) THEN
                        EXIT DIFF1
                    END IF

                END DO DIFF1

                qdotrc = qdotrnew
                qdotr = qdotrc

            END IF

            IF (ith == 0) THEN

                ! Calculate stagnation point value
                qdotrsp = qdotr

            ELSE

                ! Calculate edge point value
                IF (ith == ithmax)  qdotredg = qdotr

                ! Multiply heating rate by surface area to get heat input
                areasc = 2.0*PI * rb**2 * (COS((ith-1)/180.0*PI) - COS(ith/180.0*PI))
                qt1 = 1.0e4 * qdotr * dt * areasc
                qt2 = qt2 + qt1
                qt1 = 0.0

            END IF

        END DO I1

        ! Sum heat input
        qt = qt2 + qt
        qt2 = 0.0

        ! Determine the amount of mass ablating
        ablmass = qt / qabl
        ablfrac = ablmass / cmass

        ! Determine new mass of meteor
        cmass2 = cmass * (1 - ablfrac)
        IF (cmass2 < 0.0) THEN
            cmass2 = 0.0
        END IF
```

```
        ! *** CALCULATE FRACTURE MECHANICS INFORMATION ***

        ! calculate radius increase according to Zahnle's model
        ! once material yield limit is exceeded and using the
        ! Trapezoid Rule to solve the diff-eqtn
        IF ((pyield < pstag/2.0) .OR. (pflag)) THEN

            d2rdt2 = (Cd * rho1 * V**2 / rhoc / rb) / 2.0
            drdt = drdt - 0.5 * (d2rdt2 + d2rdt2o) * dt
            rb = rb - 0.5 * (drdt + drdto) * dt

            drdto = drdt
            d2rdt2o = d2rdt2
            pflag = .TRUE.

        ELSE

            ! Radius change due to ablation
            rb = ((3.0*cmass2)/(7.0*PI*rhoc))**(1.0/3.0)

        END IF

        ! Calculate new ballistic coef and AA coef
        balcof = cmass2 / (Cd * PI * rb**2)
        IF (cmass2 > 0.0) THEN
            AA = 1.0 / (2.0 * balcof)
        END IF

        ! Save previous kinetic energy and calculate new value
        xkeold = xke
        xke = 0.5 * cmass2 * Y(1)**2

        ! Calculate rate of energy loss
        xkerate = (xkeold - xke) / dt

        ! Convert trajectory angle to degrees
        gamma = Y(2) * 180.0 / PI

        ! Print title line every 10 iterations
        iwrite = iwrite + 1
        IF (iwrite == 1) THEN
            WRITE(1,91)
            WRITE(2,93)
        END IF

        ! Output data at each iteration
        WRITE(1,90) t, Y(3)/1000.0, Y(1)/1000.0, xke/xke1*100.0, t2sp, pstag/1000000.0, &
                    cmass2/1000.0, rb
90      FORMAT(F5.2, 2X, F7.2, 2X, F7.2, 2X, F6.1, 2X, F9.0, 2X, F10.4, 2X, &
               F12.2, 2X, F7.2)
91      FORMAT(' TIME     alt      vel     %ke       t2sp      pstag        mass   radius
')
        WRITE(2,92) t, Y(3)/1000.0, qdotrsp, enth_stag/1000000.0, rho1_stag, rho2_stag, g
oulardsp
92      FORMAT(F5.2, 2X, F7.2, 2X, F13.4, 2X, F9.4, 3X, F10.6, 3X, F10.6, 2X, F10.4)
93      FORMAT(' Time     alt          Qdot      enth         rho1      rho2    goulard')

        ! Increment time
        tnext = t + dt

        ! *** CALCULATE TRAJECTORY INFORMATION ***

        ! Call DVERK - first order differential equation solver to
        ! solve the equations of motion; automatically increments
        ! time, t, by dt
        CALL DVERK(n, FCN1, t, Y, tnext, tol, ind, C, 9, W)

        ! Set the new altitude
        h = Y(3)
```

```
        ! If the altitude is less than 1 meter or kinetic energy less than
        ! 0.01% then end
        IF ((h < 1.0) .OR. (xke/xke1 < 0.01)) THEN
            EXIT TRAJ
        END IF

        ! Determine atmospheric conditions at new altitude
        CALL ATMOS76(h, g, rho1, t1, p1, spdsnd)

        ! Set the new velocity
        v = Y(1)

        ! Screen Output
        ! WRITE(*,*) Y(3)/1000.0

    END DO TRAJ

END PROGRAM METEOR


!*************************************************************************
! FCN1
!   Routine to evaluate the righthand side of the differential equations
!   of ballistic trajectory motion.
!*************************************************************************

SUBROUTINE FCN1(N,t,Y,YPRIME)
    IMPLICIT NONE

    INTEGER N
    DOUBLE PRECISION t, Y, YPRIME

    DIMENSION Y(N), YPRIME(N)

    ! Common Block
    COMMON /BLOCK1/ R0, AA
    DOUBLE PRECISION R0, AA

    ! Declare Variables
    DOUBLE PRECISION v, h, sgamma, cgamma, R, g
    DOUBLE PRECISION a1, a2, v2, rho1, p1, t1, spdsnd

    ! Set values for differential equations
    v = Y(1)
    h = Y(3)
    sgamma = SIN(Y(2))
    cgamma = COS(Y(2))
    R = R0 + h

    ! Determine density at current altitude
    CALL ATMOS76(h,g,rho1,t1,p1,spdsnd)

    a1 = rho1 * AA
    v2 = v**2
    a2 = g - v2/R

    ! Differential Equations of Motion [Vinh et. al., pg. 28]
    YPRIME(1) = -a1 * v2 - g * sgamma
    YPRIME(2) = -a2 * cgamma/v
    YPRIME(3) = v * sgamma

RETURN
END SUBROUTINE FCN1


!*************************************************************************
! DENS_FIT
!    Routine to solve density from pressure and enthalpy using a
!    curvefit of the Vought data. The routine requires an iterative
!    solution of solving pressure from enthalpy and density until
!    the desired pressure is found.
```

```
!***********************************************************************

SUBROUTINE DENS_FIT(p_Pa, h_Jkg, rho, X, Y, Z, LX, LY, N)
    IMPLICIT NONE

    DOUBLE PRECISION p_Pa, h_Jkg, rho, X, Y, Z
    INTEGER LX, LY, N
    ! p_Pa in Pa
    ! h_Jkg in J/kg
    ! rho in kg/m^3

    ! Constants
    INTEGER, PARAMETER :: IU = 0

    ! Declare Variables
    LOGICAL DONE, FIND
    INTEGER I, LX1, LX2, LY1, LY2
    DOUBLE PRECISION U, V, W, p, h
    DOUBLE PRECISION VSTEP, dens_min, dens_max, temp

    ! Declare Arrays
    DIMENSION X(LX), Y(LY), Z(LX,LY), U(N), V(N), W(N)

    ! Convert inputs
    p = p_Pa / 101325.0      ! atm
    h = h_Jkg / 82731.168    ! Vought non-dimensional

    ! Determine valid range of data from data arrays
    if ((h >= 45.0) .AND. (h < 90.0)) then

        dens_min = 0.12251
        dens_max = 122.51
        LX1 = 1
        LY1 = 8
        LX2 = 24
        LY2 = 12

    else if ((h >= 90.0) .AND. (h <= 6000.0)) then

        dens_min = 0.0000012251
        dens_max = 122.51
        LX1 = 2
        LY1 = 1
        LX2 = 25
        LY2 = 12

    else if ((h > 6000.0) .AND. (h <= 10000.0)) then

        dens_min = 0.0000012251
        dens_max = 12.251
        LX1 = 2
        LY1 = 1
        LX2 = 26
        LY2 = 11

    else if ((h > 10000.0) .AND. (h <= 20000.0)) then

        dens_min = 0.0000012251
        dens_max = 0.12251
        LX1 = 2
        LY1 = 1
        LX2 = 27
        LY2 = 8

    else if ((h > 20000.0) .AND. (h <= 35000.0)) then

        dens_min = 0.0000012251
        dens_max = 0.000048772
        LX1 = 2
        LY1 = 1
        LX2 = 28
```

```
        LY2 = 4

    else

        ! Out of range, but try to solve anyway
        write(1,*) "Enthalpy out of range at h = ", h * 0.082731168, " MJ/kg"
        dens_min = 0.0000012251
        dens_max = 122.51
        LX1 = 2
        LY1 = 1
        LX2 = 24
        LY2 = 12

    endif

    ! Set range of possible densities
    V(1) = dens_min
    V(N) = dens_max

    ! Loop until solution found
    DONE = .FALSE.
    do 100 while (.NOT. DONE)

        ! Calculate range steps
        VSTEP = (V(N) - V(1))/(N-1.0)
        U(1) = h
        U(N) = h
        do 10 I=2,N-1
            V(I) = V(1) + (I-1)*VSTEP
            U(I) = h
10      continue

        ! Get pressure data using curve fit code
        call ITPLBV(IU, LX2-LX1+1, LY2-LY1+1, X(LX1:LX2), Y(LY1:LY2), &
                    Z(LX1:LX2,LY1:LY2), N, U, V, W)

        FIND = .FALSE.
        ! Account for out of range cases
        if (p > W(N)) then
            write(1,*) "OUT OF RANGE AT H = ", h * 0.082731168, " MJ/kg AND P = ",p*0.101
325, " MPa"
            EXIT
            V(1) = V(N)
            V(N) = 2 * V(N)
            FIND = .TRUE.
        end if

        if (p < W(1)) then
            write(1,*) "OUT OF RANGE AT H = ", h * 0.082731168, " MJ/kg AND P = ", P*0.10
1325, " MPa"
            EXIT
            V(N) = V(1)
            V(1) = dens_min
            FIND = .TRUE.
        end if

        ! Step through new pressure values, W()
        do 30 I=2,N

            ! Determine if correct value
            temp = abs((P - W(I))/P)
            if (temp < 0.001) then
                rho = V(I)
                FIND = .TRUE.
                DONE = .TRUE.
            end if

            ! Determine new range
            if (P <= W(I)) then
                if (FIND .EQV. .FALSE.) then
                    V(N) = V(I)
```

```
                    V(1) = V(I-1)
                    FIND = .TRUE.
                end if
            end if
30      end do

100 end do

RETURN
END SUBROUTINE DENS_FIT


!**********************************************************************
! TEMP_FIT
!    Routine to solve temperature from pressure and enthalpy using a
!    curve fit of the Vought data. The routine requires an iterative
!    solution of solving enthalpy from temperature and pressure until
!    the desired enthalpy is found.
!**********************************************************************

SUBROUTINE TEMP_FIT(h_Jkg, p_Pa, t, X, Y, Z, LX, LY, N)
    IMPLICIT NONE

    DOUBLE PRECISION h_Jkg, p_Pa, t, X, Y, Z
    INTEGER LX, LY, N
    ! h_Jkg in J/kg
    ! p_Pa in Pa
    ! T in Kelvin

    ! Constants
    INTEGER, PARAMETER :: IU = 0

    ! Declared Variables
    LOGICAL DONE, FIND
    INTEGER I, LX1, LX2, LY1, LY2
    DOUBLE PRECISION U, V, W, p, h
    DOUBLE PRECISION VSTEP, temp_min, temp_max, temp

    ! Declared Arrays
    DIMENSION X(LX), Y(LY), Z(LX,LY), U(N), V(N), W(N)

    ! Convert Inputs
    h = h_Jkg / 82731.168  ! Vought non-dimensional
    p = p_Pa / 101325.0    ! atm

    ! Determine valid range of data from data arrays
    if ((p >= 0.00002) .AND. (p < 0.00005)) then

        temp_min = 3000.0
        temp_max = 4000.0
        LX1 = 1
        LY1 = 1
        LX2 = 24
        LY2 = 6

    else if ((p >= 0.00005) .AND. (p < 0.0001)) then

        temp_min = 3000.0
        temp_max = 7000.0
        LX1 = 2
        LY1 = 1
        LX2 = 24
        LY2 = 15

    else if ((p >= 0.0001) .AND. (p < 0.0002)) then

        temp_min = 3000.0
        temp_max = 9000.0
        LX1 = 3
        LY1 = 1
        LX2 = 24
```

```
    LY2 = 19

else if ((p >= 0.0002) .AND. (p < 0.0005)) then

    temp_min = 3000.0
    temp_max = 15000.0
    LX1 = 4
    LY1 = 1
    LX2 = 24
    LY2 = 24

else if ((p >= 0.0005) .AND. (p < 0.001)) then

    temp_min = 3000.0
    temp_max = 25000.0
    LX1 = 5
    LY1 = 1
    LX2 = 24
    LY2 = 30

else if ((p >= 0.001) .AND. (p < 0.002)) then

    temp_min = 3000.0
    temp_max = 40000.0
    LX1 = 6
    LY1 = 1
    LX2 = 24
    LY2 = 35

else if ((p >= 0.002) .AND. (p < 0.005)) then

    temp_min = 3000.0
    temp_max = 55000.0
    LX1 = 7
    LY1 = 1
    LX2 = 24
    LY2 = 38

else if ((p >= 0.005) .AND. (p <= 1000.0)) then

    temp_min = 3000.0
    temp_max = 100000.0
    LX1 = 8
    LY1 = 1
    LX2 = 24
    LY2 = 43

else if ((p > 1000.0) .AND. (p <= 1250.0)) then

    temp_min = 3600.0
    temp_max = 100000.0
    LX1 = 8
    LY1 = 4
    LX2 = 25
    LY2 = 43

else if ((p > 1250.0) .AND. (p <= 2000.0)) then

    temp_min = 5500.0
    temp_max = 100000.0
    LX1 = 8
    LY1 = 12
    LX2 = 26
    LY2 = 43

else if ((p > 2000.0) .AND. (p <= 2500.0)) then

    temp_min = 6500.0
    temp_max = 100000.0
    LX1 = 8
    LY1 = 14
```

```
    LX2 = 27
    LY2 = 43

else if ((p > 2500.0) .AND. (p <= 3000.0)) then

    temp_min = 7500.0
    temp_max = 100000.0
    LX1 = 8
    LY1 = 16
    LX2 = 28
    LY2 = 43

else if ((p > 3000.0) .AND. (p <= 5000.0)) then

    temp_min = 11000.0
    temp_max = 100000.0
    LX1 = 8
    LY1 = 22
    LX2 = 29
    LY2 = 43

else if ((p > 5000.0) .AND. (p <= 10000.0)) then

    temp_min = 16000.0
    temp_max = 100000.0
    LX1 = 8
    LY1 = 25
    LX2 = 30
    LY2 = 43

else if ((p > 10000.0) .AND. (p <= 20000.0)) then

    temp_min = 25000.0
    temp_max = 100000.0
    LX1 = 8
    LY1 = 30
    LX2 = 31
    LY2 = 43

else if ((p > 20000.0) .AND. (p <= 40000.0)) then

    temp_min = 45000.0
    temp_max = 100000.0
    LX1 = 8
    LY1 = 36
    LX2 = 32
    LY2 = 43

else if ((p > 40000.0) .AND. (p <= 60000.0)) then

    temp_min = 60000.0
    temp_max = 100000.0
    LX1 = 8
    LY1 = 39
    LX2 = 33
    LY2 = 43

else if ((p > 60000.0) .AND. (p <= 80000.0)) then

    temp_min = 70000.0
    temp_max = 100000.0
    LX1 = 8
    LY1 = 40
    LX2 = 34
    LY2 = 43

else if ((p > 80000.0) .AND. (p <= 100000.0)) then

    temp_min = 80000.0
    temp_max = 100000.0
    LX1 = 8
```

```
        LY1 = 41
        LX2 = 35
        LY2 = 43

    else

        ! Out of range, but try to solve anyway
        write(1,*) "Pressure out of range at P = ", p * 0.101325, " MPa"
        temp_min = 3000.0
        temp_max = 100000.0
        LX1 = 8
        LY1 = 1
        LX2 = 24
        LY2 = 43

    endif

    ! Set a range of possible temperatures
    V(1) = temp_min
    V(N) = temp_max

    ! Loop until solution found
    DONE = .FALSE.
    do 100 while (.NOT. DONE)

        ! Calculate range steps
        VSTEP = (V(N) - V(1))/(N-1.0)
        U(1) = P
        U(N) = P
        do 10 I=2,N-1
            V(I) = V(1) + (I-1)*VSTEP
            U(I) = p
10      continue

        ! Get enthalpy data using curve fit code
        call ITPLBV(IU, LX2-LX1+1, LY2-LY1+1, X(LX1:LX2), Y(LY1:LY2), &
                    Z(LX1:LX2,LY1:LY2), N, U, V, W)

        FIND = .FALSE.
        ! Account for out of range cases
        if (h > W(N)) then
            write(1,*) "OUT OF RANGE AT p = ", P*0.101325, " MPa AND h = ",h*19.76/238.84
5896627, " MJ/kg"
            EXIT
            V(1) = V(N)
            V(N) = 2 * V(N)
            FIND = .TRUE.
        end if

        if (h < W(1)) then
            write(1,*) "OUT OF RANGE AT p = ", P*0.101325, " MPa AND h = ", h*19.76/238.8
45896627, " MJ/kg"
            EXIT
            V(N) = V(1)
            V(1) = temp_min
            FIND = .TRUE.
        end if

        ! Step through new enthalpy values, W()
        do 30 I=2,N

            ! Determine if correct value
            temp = abs((h - W(I))/h)
            if (temp < 0.001) then
                t = V(I)
                FIND = .TRUE.
                DONE = .TRUE.
            end if

            ! Determine new range
            if (h <= W(I)) then
```

```
                if (FIND .EQV. .FALSE.) then
                    V(N) = V(I)
                    V(1) = V(I-1)
                    FIND = .TRUE.
                end if
            end if
30      end do

100 end do

RETURN
END SUBROUTINE TEMP_FIT


!**********************************************************************
! TEMPNEW
!   This is a routine written by Rich Fought to return the temperature
!   for a given pressure and enthalpy. It is based on Roop Gupta's
!   work and is valid from 500 K to 32600 K and 10^-4 atm to 1000 atm.
!**********************************************************************

SUBROUTINE TEMPNEW(H,P,T)
    IMPLICIT NONE

    DOUBLE PRECISION H, P, T
    ! H in J/kg
    ! P in Pa
    ! T in K

    DOUBLE PRECISION CHIS, THETAS, BPS
    DOUBLE PRECISION HDATANEW, A, THETAMAX, CHI
    INTEGER I, J, BP, ZN

    DIMENSION HDATANEW(36,6), A(6), THETAMAX(8), CHI(2)
    DIMENSION BP(2)

    OPEN(UNIT=20, FILE='hdatanew.dat', STATUS='OLD')

    DO 1 I=1,36
        READ(20,*) (HDATANEW(I,J), J=1,6)
1   CONTINUE

    DO 2 I=1,8
        READ(20,*) THETAMAX(I)
2   CONTINUE

    CLOSE(20)

    BPS=LOG10(P/101325.0)

    THETAS=LOG(H/4186800.0)

    IF (BPS.GT.0) THEN
        BP(2)=AINT(BPS)+1
        BP(1)=AINT(BPS)
        IF (THETAS.GT.THETAMAX(BP(2)+5)) THEN
            BP(2)=BP(1)
            BP(1)=BP(1)-1
        END IF
    ELSE
        BP(2)=AINT(BPS)
        BP(1)=AINT(BPS)-1
        IF (THETAS.GT.THETAMAX(BP(2)+5)) THEN
            BP(2)=BP(1)
            BP(1)=BP(1)-1
        END IF
        IF (BPS.LT.-4) THEN
            BP(1)=-3
            BP(2)=-4
        END IF
    END IF
```

```
DO 3 I=1,2

    IF (BP(I).EQ.-4) THEN
        IF (THETAS.LE.-2.085) THEN
            CHI(I)=THETAS-0.87547
            GOTO 4
        END IF
        IF (THETAS.GT.-2.085.AND.THETAS.LE.-0.0846) ZN=1
        IF (THETAS.GT.-0.0846.AND.THETAS.LE.1.284) ZN=2
        IF (THETAS.GT.1.284.AND.THETAS.LE.2.227) ZN=3
        IF (THETAS.GT.2.227.AND.THETAS.LE.2.3065) ZN=4
        IF (THETAS.GT.2.3065.AND.THETAS.LE.3.607) ZN=5
        IF (THETAS.GT.3.607.AND.THETAS.LE.4.172) ZN=6
        IF (THETAS.GT.4.172.AND.THETAS.LE.4.652) ZN=7
        IF (THETAS.GT.4.652) THEN
            WRITE(6,*)'Enthalpy out of range'
            GOTO 4
        END IF
    END IF

    IF (BP(I).EQ.-3) THEN
        IF (THETAS.LE.-2.087) THEN
            CHI(I)=THETAS-0.87547
            GOTO 4
        END IF
        IF (THETAS.GT.-2.087.AND.THETAS.LE.-0.2788) ZN=8
        IF (THETAS.GT.-0.2788.AND.THETAS.LE.0.9678) ZN=9
        IF (THETAS.GT.0.9678.AND.THETAS.LE.2.247) ZN=10
        IF (THETAS.GT.2.247.AND.THETAS.LE.3.607) ZN=11
        IF (THETAS.GT.3.607.AND.THETAS.LE.4.005) ZN=12
        IF (THETAS.GT.4.005.AND.THETAS.LE.4.686) ZN=13
        IF (THETAS.GT.4.686) THEN
            WRITE(6,*)'Enthalpy out of range'
            GOTO 4
        END IF
    END IF

    IF (BP(I).EQ.-2) THEN
        IF (THETAS.LE.-2.086) THEN
            CHI(I)=THETAS-0.87547
            GOTO 4
        END IF
        IF (THETAS.GT.-2.086.AND.THETAS.LE.0.1125) ZN=14
        IF (THETAS.GT.0.1125.AND.THETAS.LE.1.498) ZN=15
        IF (THETAS.GT.1.498.AND.THETAS.LE.2.748) ZN=16
        IF (THETAS.GT.2.748.AND.THETAS.LE.3.779) ZN=17
        IF (THETAS.GT.3.779.AND.THETAS.LE.4.689) ZN=18
        IF (THETAS.GT.4.689) THEN
            WRITE(6,*)'Enthalpy out of range'
            GOTO 4
        END IF
    END IF

    IF (BP(I).EQ.-1) THEN
        IF (THETAS.LE.-2.093) THEN
            CHI(I)=THETAS-0.87547
            GOTO 4
        END IF
        IF (THETAS.GT.-2.093.AND.THETAS.LE.0.352) ZN=19
        IF (THETAS.GT.0.352.AND.THETAS.LE.1.837) ZN=20
        IF (THETAS.GT.1.837.AND.THETAS.LE.3.644) ZN=21
        IF (THETAS.GT.3.644.AND.THETAS.LE.4.648) ZN=22
        IF (THETAS.GT.4.648) THEN
            WRITE(6,*)'Enthalpy out of range'
            GOTO 4
        END IF
    END IF

    IF (BP(I).EQ.0) THEN
        IF (THETAS.LE.-2.096) THEN
```

```
            CHI(I)=THETAS-0.87547
            GOTO 4
        END IF
        IF (THETAS.GT.-2.096.AND.THETAS.LE.0.485) ZN=23
        IF (THETAS.GT.0.485.AND.THETAS.LE.2.269) ZN=24
        IF (THETAS.GT.2.269.AND.THETAS.LE.3.659) ZN=25
        IF (THETAS.GT.3.659.AND.THETAS.LE.4.365) ZN=26
        IF (THETAS.GT.4.365) THEN
            WRITE(6,*)'Enthalpy out of range'
            GOTO 4
        END IF
    END IF

    IF (BP(I).EQ.1) THEN
        IF (THETAS.LE.-2.097) THEN
            CHI(I)=THETAS-0.87547
            GOTO 4
        END IF
        IF (THETAS.GT.-2.097.AND.THETAS.LE.0.555) ZN=27
        IF (THETAS.GT.0.555.AND.THETAS.LE.2.265) ZN=28
        IF (THETAS.GT.2.265.AND.THETAS.LE.3.468) ZN=29
        IF (THETAS.GT.3.468.AND.THETAS.LE.4.043) ZN=30
        IF (THETAS.GT.4.043) THEN
            WRITE(6,*)'Enthalpy out of range'
            GOTO 4
        END IF
    END IF

    IF (BP(I).EQ.2) THEN
        IF (THETAS.LE.-2.117) THEN
            CHI(I)=THETAS-0.87547
            GOTO 4
        END IF
        IF (THETAS.GT.-2.117.AND.THETAS.LE.1.041) ZN=31
        IF (THETAS.GT.1.041.AND.THETAS.LE.2.515) ZN=32
        IF (THETAS.GT.2.515.AND.THETAS.LE.3.861) ZN=33
        IF (THETAS.GT.3.861) THEN
            WRITE(6,*)'Enthalpy out of range'
            GOTO 4
        END IF
    END IF

    IF (BP(I).EQ.3) THEN
        IF (THETAS.LE.-2.137) THEN
            CHI(I)=THETAS-0.87547
            GOTO 4
        END IF
        IF (THETAS.GT.-2.137.AND.THETAS.LE.2.369) ZN=34
        IF (THETAS.GT.2.369.AND.THETAS.LE.2.596) ZN=35
        IF (THETAS.GT.2.596.AND.THETAS.LE.3.764) ZN=36
        IF (THETAS.GT.3.764) THEN
            WRITE(6,*)'Enthalpy out of range'
            GOTO 4
        END IF
    END IF

    DO 5 J=1,6
        A(J)=HDATANEW(ZN,J)
5   CONTINUE

    CHI(I)=A(1)*THETAS**5+A(2)*THETAS**4+A(3)*THETAS**3 &
        +A(4)*THETAS**2+A(5)*THETAS+A(6)

4   CONTINUE

3 CONTINUE

  CHIS=(BPS-BP(1))/(BP(2)-BP(1))*(CHI(2)-CHI(1))+CHI(1)

  T=10000.0*EXP(CHIS)
```

```
RETURN
END SUBROUTINE TEMPNEW


!-------------------------------------------------------------------------------
!                              U.S. STANDARD ATMOSPHERE 1976
!
!-------------------------------------------------------------------------------
!
SUBROUTINE ATMOS76(ALT,RESG,RESR,REST,RESP,RESCS)
    IMPLICIT NONE

    DOUBLE PRECISION ALT, RESG, RESR, REST, RESP, RESCS

    INTEGER I
    DOUBLE PRECISION KTSL, NSL, MUSL, M0, MSL, MFPSL, MCFSL
    DOUBLE PRECISION RESM, RESN, AVO, UGAS, PI, G0, G0P, P0, R0, T0
    DOUBLE PRECISION SUTH, BETA, GAMMA, ZSL, TSL, PSL, RSL
    DOUBLE PRECISION GSL, HPSL, CSSL, SIGMA, Z, RESH, RESTM, RESHP, RESMU
    DOUBLE PRECISION RESKT, RESMFP, RESMCF, TEMPM, WEIGHT, TNDENS, PRESS
    DOUBLE PRECISION THERM, VISC, FR, FPATH, SOUND, PSCALE, GRAVITY
    DOUBLE PRECISION DENSITY, GPALT

    DOUBLE PRECISION FI, HB, TMB, PB, MI, LMB

    DIMENSION FI(10), HB(8), TMB(7), PB(7), MI(10), LMB(10)

!   CONSTANTS  ---  CATEGORY I

      DATA AVO,UGAS,PI / 6.022169E+26, 8.31432E+03, 3.141592654 /
      DATA (MI(I),I=1,10) / 28.0134, 31.9988, 39.948, 44.00995, 20.183,  &
                            4.0026, 83.80, 131.30, 16.04303, 2.01594 /

!     CONSTANTS  ---  CATEGORY II

      DATA G0, G0P, P0, R0, T0, SUTH, BETA, GAMMA, SIGMA            &
       / 9.80665, 9.80665, 1.01325E+05, 6356766.0, 288.15 , 110.4,  &
         1.458E-06, 1.40, 3.65E-10 /
      DATA (FI(I),I=1,10) / 0.78084, 0.209476, 9.34E-03, 3.14E-04,  &
                            1.818E-05, 5.24E-06, 1.14E-06, 8.7E-08, &
                            2.0E-06, 5.0E-07 /
      DATA (HB(I),I=1,8)  / 0.0, 11.0, 20.0, 32.0, 47.0, 51.0, 71.0, 84.8520 /
      DATA (LMB(I),I=1,7) / -6.5, 0.0, 1.0, 2.8, 0.0, -2.8, -2.0 /

!     COMPUTE SEALEVEL VALUES

      ZSL   = 0.0
      MSL   = WEIGHT(FI,MI)
      M0    = MSL
      CALL TENDS(HB,LMB,T0,TMB)
      CALL PENDS(G0P,HB,LMB,M0,P0,PB,TMB,UGAS)
      TSL   = TEMPM(G0,G0P,HB,LMB,R0,TMB,ZSL)
      PSL   = PRESS(G0,G0P,HB,LMB,M0,PB,R0,TMB,UGAS,ZSL)
      RSL   = DENSITY(MSL,PSL,TSL,UGAS,G0,G0P,R0,ZSL)
      GSL   = GRAVITY(G0,G0P,R0,ZSL)
      HPSL  = PSCALE(GSL,MSL,TSL,UGAS)
      CSSL  = SOUND(GAMMA,MSL,TSL,UGAS)
      MUSL  = VISC(BETA,SUTH,TSL)
      KTSL  = THERM(TSL)
      NSL   = TNDENS(AVO,PSL,TSL,UGAS)
      MFPSL = FPATH(NSL,PI,SIGMA)
      MCFSL = FR(AVO,MSL,NSL,PSL,PI,SIGMA,TSL,UGAS)

!     COMPUTE PHYSICAL QUANTITIES FOR THE GIVEN ALTITUDE

      Z=ALT
      RESH  = GPALT(G0,G0P,R0,Z)
      RESM  = M0
      RESTM = TEMPM(G0,G0P,HB,LMB,R0,TMB,Z)
      REST  = RESTM
      RESP  = PRESS(G0,G0P,HB,LMB,M0,PB,R0,TMB,UGAS,Z)
```

```
      RESR   = DENSITY(RESM,RESP,REST,UGAS,G0,G0P,R0,Z)
      RESG   = GRAVITY(G0,G0P,R0,Z)
      RESHP  = PSCALE(RESG,RESM,REST,UGAS)
      RESCS  = SOUND(GAMMA,RESM,REST,UGAS)
      RESMU  = VISC(BETA,SUTH,REST)
      RESKT  = THERM(REST)
      RESN   = TNDENS(AVO,RESP,REST,UGAS)
      RESMFP = FPATH(RESN,PI,SIGMA)
      RESMCF = FR(AVO,RESM,RESN,RESP,PI,SIGMA,REST,UGAS)

RETURN
END SUBROUTINE ATMOS76
!-------------------------------------------------------------------------------

FUNCTION GPALT(G0,G0P,R0,Z)
    IMPLICIT NONE

    DOUBLE PRECISION GPALT, G0, G0P, R0, Z

    DOUBLE PRECISION GAMMA

      GAMMA = G0/G0P

      GPALT = GAMMA*R0*Z/(R0+Z)

RETURN
END FUNCTION GPALT
!-------------------------------------------------------------------------------

FUNCTION WEIGHT(FI,MI)
    IMPLICIT NONE

    DOUBLE PRECISION WEIGHT, FI, MI
    DIMENSION FI(10), MI(10)

    INTEGER I
    DOUBLE PRECISION SUM1, SUM2

      SUM1 = 0.0
      SUM2 = 0.0

      DO 10 I = 1,10
        SUM1 = SUM1+FI(I)
        SUM2 = SUM2+MI(I)*FI(I)
   10 CONTINUE
      WEIGHT = SUM2/SUM1

RETURN
END FUNCTION WEIGHT
!-------------------------------------------------------------------------------

SUBROUTINE TENDS(HB,LMB,T0,TMB)
    IMPLICIT NONE

    DOUBLE PRECISION HB, LMB, T0, TMB
    DIMENSION LMB(7), HB(8), TMB(7)

    INTEGER I

      TMB(1) = T0
      DO 10 I = 2,7
        TMB(I) = TMB(I-1) + LMB(I-1)*(HB(I)-HB(I-1))
   10 CONTINUE

RETURN
END SUBROUTINE TENDS
!-------------------------------------------------------------------------------

SUBROUTINE PENDS(G0P,HB,LMB,M0,P0,PB,TMB,UGAS)
    IMPLICIT NONE
```

```
      DOUBLE PRECISION G0P, HB, LMB, M0, P0, PB, TMB, UGAS
      DIMENSION HB(8), LMB(7), PB(7), TMB(7)

      INTEGER I
      DOUBLE PRECISION AA, BB

        PB(1) = P0
        DO 10 I = 2,7
           IF (LMB(I-1) .NE. 0.0) THEN
              AA = TMB(I-1)/TMB(I)
              BB = G0P*M0/(1.E-03*UGAS*LMB(I-1))
              PB(I) = PB(I-1)*(AA**BB)
           ELSE
              AA = 1.0E+03*G0P*M0*(HB(I)-HB(I-1))/(UGAS*TMB(I-1))
              PB(I) = PB(I-1)*EXP(-AA)
           ENDIF
 10     CONTINUE

RETURN
END SUBROUTINE PENDS
!-------------------------------------------------------------------------------

FUNCTION TEMPM(G0,G0P,HB,LMB,R0,TMB,Z)
      IMPLICIT NONE

        DOUBLE PRECISION TEMPM, G0, G0P, HB, LMB, R0, TMB, Z
        DIMENSION HB(8), LMB(7), TMB(7)

        DOUBLE PRECISION GAMMA, HKM

        GAMMA = G0/G0P

        HKM = 1.E-03*GAMMA*R0*Z/(R0+Z)

        IF (HKM .LT. HB(2)) THEN
           TEMPM = TMB(1) + LMB(1)*(HKM-HB(1))
        ELSE IF ((HKM .GE. HB(2)) .AND. (HKM .LT. HB(3))) THEN
           TEMPM = TMB(2) + LMB(2)*(HKM-HB(2))
        ELSE IF ((HKM .GE. HB(3)) .AND. (HKM .LT. HB(4))) THEN
           TEMPM = TMB(3) + LMB(3)*(HKM-HB(3))
        ELSE IF ((HKM .GE. HB(4)) .AND. (HKM .LT. HB(5))) THEN
           TEMPM = TMB(4) + LMB(4)*(HKM-HB(4))
        ELSE IF ((HKM .GE. HB(5)) .AND. (HKM .LT. HB(6))) THEN
           TEMPM = TMB(5) + LMB(5)*(HKM-HB(5))
        ELSE IF ((HKM .GE. HB(6)) .AND. (HKM .LT. HB(7))) THEN
           TEMPM = TMB(6) + LMB(6)*(HKM-HB(6))
        ELSE IF ((HKM .GE. HB(7)) .AND. (HKM .LE. HB(8))) THEN
           TEMPM = TMB(7) + LMB(7)*(HKM-HB(7))
        ENDIF

        IF(HKM.GT.84.852.AND.HKM.LT.92.630) TEMPM=187.
        IF(HKM.GT.92.63.AND.HKM.LT.97.482)TEMPM=(HKM-92.63)*1.1418+187.74
        IF(HKM.GT.97.482.AND.HKM.LT.101.358)TEMPM=(HKM-97.482)*2.31+193.3
        IF(HKM.GT.101.358.AND.HKM.LT.105.229)TEMPM=(HKM-101.36)*3.98+202.2
        IF(HKM.GT.105.229.AND.HKM.LT.108.129)TEMPM=(HKM-105.23)*7.71+217.6
        IF(HKM.GT.108.129)TEMPM=(HKM-108.129)*12.31+240.

RETURN
END FUNCTION TEMPM
!-------------------------------------------------------------------------------

FUNCTION PRESS(G0,G0P,HB,LMB,M0,PB,R0,TMB,UGAS,Z)
      IMPLICIT NONE

        DOUBLE PRECISION PRESS, G0, G0P, HB, LMB, M0, PB, R0, TMB, UGAS, Z
        DIMENSION HB(8), LMB(7), PB(7), TMB(7)

      DOUBLE PRECISION GAMMA, HKM, AA, BB

        GAMMA = G0/G0P
```

```
      HKM = 1.E-03*GAMMA*R0*Z/(R0+Z)

      IF (HKM .LT. HB(2)) THEN
         IF (LMB(1) .NE. 0.0) THEN
            AA = TMB(1)/(TMB(1)+LMB(1)*(HKM-HB(1)))
            BB = G0P*M0/(1.E-03*UGAS*LMB(1))
            PRESS  = PB(1)*(AA**BB)
         ELSE
            AA = 1.E+03*G0P*M0*(HKM-HB(1))/(UGAS*TMB(1))
            PRESS  = PB(1)*EXP(-AA)
         ENDIF
      ELSE IF ((HKM .GE. HB(2)) .AND. (HKM .LT. HB(3))) THEN
         IF (LMB(2) .NE. 0.0) THEN
            AA = TMB(1)/(TMB(2)+LMB(1)*(HKM-HB(2)))
            BB = G0P*M0/(1.E-03*UGAS*LMB(2))
            PRESS  = PB(2)*(AA**BB)
         ELSE
            AA = 1.E+03*G0P*M0*(HKM-HB(2))/(UGAS*TMB(2))
            PRESS  = PB(2)*EXP(-AA)
         ENDIF
      ELSE IF ((HKM .GE. HB(3)) .AND. (HKM .LT. HB(4))) THEN
         IF (LMB(3) .NE. 0.0) THEN
            AA = TMB(3)/(TMB(3)+LMB(3)*(HKM-HB(3)))
            BB = G0P*M0/(1.E-03*UGAS*LMB(3))
            PRESS  = PB(3)*(AA**BB)
         ELSE
            AA = 1.E+03*G0P*M0*(HKM-HB(3))/(UGAS*TMB(3))
            PRESS  = PB(3)*EXP(-AA)
         ENDIF
      ELSE IF ((HKM .GE. HB(4)) .AND. (HKM .LT. HB(5))) THEN
         IF (LMB(4) .NE. 0.0) THEN
            AA = TMB(4)/(TMB(4)+LMB(4)*(HKM-HB(4)))
            BB = G0P*M0/(1.E-03*UGAS*LMB(4))
            PRESS  = PB(4)*(AA**BB)
         ELSE
            AA = 1.E+03*G0P*M0*(HKM-HB(4))/(UGAS*TMB(4))
            PRESS  = PB(4)*EXP(-AA)
         ENDIF
      ELSE IF ((HKM .GE. HB(5)) .AND. (HKM .LT. HB(6))) THEN
         IF (LMB(5) .NE. 0.0) THEN
            AA = TMB(5)/(TMB(5)+LMB(5)*(HKM-HB(5)))
            BB = G0P*M0/(1.E-03*UGAS*LMB(5))
            PRESS  = PB(5)*(AA**BB)
         ELSE
            AA = 1.E+03*G0P*M0*(HKM-HB(5))/(UGAS*TMB(5))
            PRESS  = PB(5)*EXP(-AA)
         ENDIF
      ELSE IF ((HKM .GE. HB(6)) .AND. (HKM .LT. HB(7))) THEN
         IF (LMB(6) .NE. 0.0) THEN
            AA = TMB(6)/(TMB(6)+LMB(6)*(HKM-HB(6)))
            BB = G0P*M0/(1.E-03*UGAS*LMB(6))
            PRESS  = PB(6)*(AA**BB)
         ELSE
            AA = 1.E+03*G0P*M0*(HKM-HB(6))/(UGAS*TMB(6))
            PRESS  = PB(6)*EXP(-AA)
         ENDIF
      ELSE IF ((HKM .GE. HB(7)) .AND. (HKM .LE. HB(8))) THEN
         IF (LMB(7) .NE. 0.0) THEN
            AA = TMB(7)/(TMB(7)+LMB(7)*(HKM-HB(7)))
            BB = G0P*M0/(1.E-03*UGAS*LMB(7))
            PRESS  = PB(7)*(AA**BB)
         ELSE
            AA = 1.E+03*G0P*M0*(HKM-HB(7))/(UGAS*TMB(7))
            PRESS  = PB(7)*EXP(-AA)
         ENDIF
      ELSE IF (HKM.GT.HB(8)) THEN
         PRESS=100.
      ENDIF

RETURN
END FUNCTION PRESS
```

```
!-------------------------------------------------------------------------

FUNCTION DENSITY(M,P,T,UGAS,G0,G0P,R0,Z)
    IMPLICIT NONE

      DOUBLE PRECISION M, P, T, UGAS, G0, G0P, R0, Z

      DOUBLE PRECISION DENSITY, HKM, TEMP

      DENSITY = P*M/(UGAS*T)

      HKM = 0.001*G0*Z*R0/(G0P*(R0 + Z))

      IF ((HKM .GT. 84.852) .AND. (HKM .LT. 112.957)) THEN
        TEMP = -0.0773 * (HKM - 84.852)
        DENSITY = 6.958E-6 * 10**TEMP
      ENDIF
      IF (HKM .GT. 112.957) THEN
        TEMP = -0.0531 * (HKM - 112.957)
        DENSITY = 4.289E-8 * 10**TEMP
      ENDIF
      IF (HKM .GT. 122.0) DENSITY = 0.0


RETURN
END FUNCTION DENSITY
!-------------------------------------------------------------------------

FUNCTION GRAVITY(G0,G0P,R0,Z)
    IMPLICIT NONE

    DOUBLE PRECISION GRAVITY, G0, G0P, R0, Z

      GRAVITY = G0*((R0/(R0+Z))**2)

RETURN
END FUNCTION GRAVITY
!-------------------------------------------------------------------------

FUNCTION VISC(BETA,SUTH,T)
    IMPLICIT NONE

    DOUBLE PRECISION VISC, BETA, SUTH, T

      VISC = BETA*(T**1.5)/(T+SUTH)

RETURN
END FUNCTION VISC
!-------------------------------------------------------------------------

FUNCTION THERM(T)
    IMPLICIT NONE

    DOUBLE PRECISION THERM, T

      THERM = 2.64638E-03*(T**1.5)/(T+245.4*(10.0**(-12/T)))

RETURN
END FUNCTION THERM
!-------------------------------------------------------------------------

FUNCTION SOUND(GAMMA,M,T,UGAS)
    IMPLICIT NONE

      DOUBLE PRECISION SOUND, GAMMA, M, T, UGAS

      SOUND = SQRT(GAMMA*UGAS*T/M)

RETURN
END FUNCTION SOUND
```

```
!-------------------------------------------------------------------------------

FUNCTION TNDENS(AVO,P,T,UGAS)
    IMPLICIT NONE

    DOUBLE PRECISION TNDENS, AVO, P, T, UGAS

      TNDENS = AVO*P/(UGAS*T)

RETURN
END FUNCTION TNDENS
!-------------------------------------------------------------------------------

FUNCTION FPATH(N,PI,SIGMA)
    IMPLICIT NONE

      DOUBLE PRECISION FPATH, N, PI, SIGMA

      FPATH = SQRT(2.0)/(2.*PI*(SIGMA**2.0)*N)

RETURN
END FUNCTION FPATH
!-------------------------------------------------------------------------------

FUNCTION FR(AVO,M,N,P,PI,SIGMA,T,UGAS)
    IMPLICIT NONE

      DOUBLE PRECISION FR, AVO, M, N, P, PI, SIGMA, T, UGAS

      FR = 4.0*AVO*(SIGMA**2.0)*SQRT(PI*(P**2.0)/(UGAS*T*M))

RETURN
END FUNCTION FR
!-------------------------------------------------------------------------------

SUBROUTINE COMPOSIT(FI,N,NI)
    IMPLICIT NONE

      DOUBLE PRECISION FI, N, NI
      DIMENSION FI(10), NI(10)

      INTEGER I

      DO 10 I = 1,10
         NI(I) = FI(I)*N
   10 CONTINUE

RETURN
END SUBROUTINE COMPOSIT
!-------------------------------------------------------------------------------

FUNCTION PSCALE(G,M,T,UGAS)
    IMPLICIT NONE

      DOUBLE PRECISION PSCALE, G, M, T, UGAS

      PSCALE = UGAS*T/(G*M)

RETURN
END FUNCTION PSCALE


!*******************************************************************************
! DVERK
!    This program is a first order differential equation solver. A more
!    thorough description is given below.
!*******************************************************************************

SUBROUTINE DVERK (n, fcn, x, y, xend, tol, ind, c, nw, w)
    IMPLICIT NONE
```

```
      INTEGER n, ind, nw, k
      DOUBLE PRECISION x, y(n), xend, tol, c(1), w(nw,9), temp

!***********************************************************************
!                                                                      *
! note added 11/14/85.                                                 *
!                                                                      *
! if you discover any errors in this subroutine, please contact        *
!                                                                      *
!        kenneth r. jackson                                            *
!        department of computer science                                *
!        university of toronto                                         *
!        toronto, ontario,                                             *
!        canada   m5s 1a4                                              *
!                                                                      *
!        phone: 416-978-7075                                           *
!                                                                      *
!        electronic mail:                                              *
!        uucp:   {cornell,decvax,ihnp4,linus,uw-beaver}!utcsri!krj     *
!        csnet:  krj@toronto                                           *
!        arpa:   krj.toronto@csnet-relay                               *
!        bitnet: krj%toronto@csnet-relay.arpa                          *
!                                                                      *
! dverk is written in fortran 66.                                      *
!                                                                      *
! the constants dwarf and rreb -- c(10) and c(11), respectively -- are *
! set for a vax in double precision. they should be reset, as          *
! described below, if this program is run on another machine.          *
!                                                                      *
! the c array is declared in this subroutine to have one element only, *
! although more elements are referenced in this subroutine. this       *
! causes some compilers to issue warning messages. there is, though,   *
! no error provided c is declared sufficiently large in the calling    *
! program, as described below.                                         *
!                                                                      *
! the following external statement for fcn was added to avoid a        *
! warning message from the unix f77 compiler. the original dverk       *
! comments and code follow it.                                         *
!                                                                      *
!***********************************************************************

      EXTERNAL FCN1, MAX
      DOUBLE PRECISION MAX

!***********************************************************************
!                                                                      *
!     purpose - this is a runge-kutta subroutine based on verner's     *
! fifth and sixth order pair of formulas for finding approximations to *
! the solution of a system of first order ordinary differential        *
! equations with initial conditions. it attempts to keep the global    *
! error proportional to a tolerance specified by the user. (the        *
! proportionality depends on the kind of error control that is used,   *
! as well as the differential equation and the range of integration.)   *
!                                                                      *
!     various options are available to the user, including different   *
! kinds of error control, restrictions on step sizes, and interrupts   *
! which permit the user to examine the state of the calculation (and   *
! perhaps make modifications) during intermediate stages.              *
!                                                                      *
!     the program is efficient for non-stiff systems. however, a good  *
! variable-order-adams method will probably be more efficient if the   *
! function evaluations are very costly. such a method would also be    *
! more suitable if one wanted to obtain a large number of intermediate *
! solution values by interpolation, as might be the case for example   *
! with graphical output.                                               *
!                                                                      *
!                                  hull-enright-jackson   1/10/76      *
!                                                                      *
!***********************************************************************
!                                                                      *
!     use - the user must specify each of the following                *
```

```
!     n  number of equations                                              *
!                                                                         *
!   fcn  name of subroutine for evaluating functions - the  subroutine    *
!          itself must also be provided by the user - it should be of     *
!          the following form                                             *
!             subroutine fcn(n, x, y, yprime)                             *
!             integer n                                                   *
!             double precision x, y(n), yprime(n)                         *
!                    *** etc ***                                          *
!          and it should evaluate yprime, given n, x and y                *
!                                                                         *
!     x  independent variable - initial value supplied by user            *
!                                                                         *
!     y  dependent variable - initial values of components y(1), y(2),    *
!          ..., y(n) supplied by user                                     *
!                                                                         *
!  xend  value of x to which integration is to be carried out - it may    *
!          be less than the initial value of x                            *
!                                                                         *
!   tol  tolerance - the subroutine attempts to control a norm of  the    *
!          local  error  in  such  a  way  that  the  global error is     *
!          proportional to tol. in some problems there will be enough     *
!          damping  of  errors, as well as some cancellation, so that     *
!          the global error will be less than tol. alternatively, the     *
!          control   can   be   viewed   as   attempting   to   provide a *
!          calculated value of y at xend which is the exact  solution     *
!          to  the  problem y' = f(x,y) + e(x) where the norm of e(x)      *
!          is proportional to tol.  (the norm  is  a  max  norm  with     *
!          weights  that  depend on the error control strategy chosen     *
!          by the user.  the default weight for the k-th component is     *
!          1/max(1,abs(y(k))),  which therefore provides a mixture of     *
!          absolute and relative error control.)                          *
!                                                                         *
!   ind  indicator - on initial entry ind must be set equal to  either    *
!          1  or  2. if the user does not wish to use any options, he     *
!          should set ind to 1 - all that remains for the user to  do     *
!          then  is  to  declare c and w, and to specify nw. the user     *
!          may  also  select  various  options  on  initial  entry  by    *
!          setting ind = 2 and initializing the first 9 components of     *
!          c as described in the next section.  he may also  re-enter     *
!          the  subroutine  with ind = 3 as mentioned again below. in     *
!          any event, the subroutine returns with ind equal to            *
!             3 after a normal return                                     *
!             4, 5, or 6 after an interrupt (see options c(8), c(9))      *
!             -1, -2, or -3 after an error condition (see below)          *
!                                                                         *
!     c  communications vector - the dimension must be greater than or    *
!          equal to 24, unless option c(1) = 4 or 5 is used, in which     *
!          case the dimension must be greater than or equal to n+30       *
!                                                                         *
!    nw  first dimension of workspace w -  must  be  greater  than  or    *
!          equal to n                                                     *
!                                                                         *
!     w  workspace matrix - first dimension must be nw and second must    *
!          be greater than or equal to 9                                  *
!                                                                         *
!    the subroutine  will  normally  return  with  ind  =  3,  having     *
! replaced the initial values of x and y with, respectively, the value    *
! of xend and an approximation to y at xend.  the  subroutine  can  be    *
! called  repeatedly  with new values of xend without having to change    *
! any other argument.  however, changes in tol, or any of the  options    *
! described below, may also be made on such a re-entry if desired.        *
!                                                                         *
!    three error returns are also possible, in which  case  x  and  y     *
! will be the most recently accepted values -                             *
!    with ind = -3 the subroutine was unable  to  satisfy  the  error     *
!       requirement  with a particular step-size that is less than or     *
!       equal to hmin, which may mean that tol is too small               *
!    with ind = -2 the value of hmin  is  greater  than  hmax,  which     *
!       probably  means  that the requested tol (which is used in the     *
```

```
!        calculation of hmin) is too small                              *
!     with ind = -1 the allowed maximum number of fcn evaluations  has  *
!        been  exceeded,  but  this  can only occur if option c(7), as  *
!        described in the next section, has been used                   *
!                                                                       *
!     there are several circumstances that will cause the calculations  *
! to  be  terminated,  along with output of information that will help  *
! the user determine the cause of  the  trouble.  these  circumstances  *
! involve  entry with illegal or inconsistent values of the arguments,  *
! such as attempting a normal  re-entry  without  first  changing  the  *
! value of xend, or attempting to re-enter with ind less than zero.     *
!                                                                       *
!***********************************************************************
!                                                                       *
!     options - if the subroutine is entered with ind = 1, the first 9  *
! components of the communications vector are initialized to zero, and  *
! the subroutine uses only default values  for  each  option.  if  the  *
! subroutine  is  entered  with ind = 2, the user must specify each of  *
! these 9 components - normally he would first set them all  to  zero,  *
! and  then  make  non-zero  those  that  correspond to the particular  *
! options he wishes to select. in any event, options may be changed on  *
! re-entry  to  the  subroutine  -  but if the user changes any of the  *
! options, or tol, in the course of a calculation he should be careful  *
! about  how  such changes affect the subroutine - it may be better to  *
! restart with ind = 1 or 2. (components 10 to 24 of c are used by the  *
! program  -  the information is available to the user, but should not  *
! normally be changed by him.)                                          *
!                                                                       *
!  c(1)  error control indicator - the norm of the local error is  the  *
!           max  norm  of  the  weighted  error  estimate  vector, the  *
!           weights being determined according to the value of c(1) -   *
!              if c(1)=1 the weights are 1 (absolute error control)     *
!              if c(1)=2 the weights are 1/abs(y(k))  (relative  error  *
!                 control)                                              *
!              if c(1)=3 the  weights  are  1/max(abs(c(2)),abs(y(k)))  *
!                 (relative  error  control,  unless abs(y(k)) is less  *
!                 than the floor value, abs(c(2)) )                     *
!              if c(1)=4 the weights are 1/max(abs(c(k+30)),abs(y(k)))  *
!                 (here individual floor values are used)               *
!              if c(1)=5 the weights are 1/abs(c(k+30))                 *
!              for all other values of c(1), including  c(1) = 0,  the  *
!                 default  values  of  the  weights  are  taken  to be  *
!                 1/max(1,abs(y(k))), as mentioned earlier              *
!           (in the two cases c(1) = 4 or 5 the user must declare  the   *
!           dimension of c to be at least n+30 and must initialize the  *
!           components c(31), c(32), ..., c(n+30).)                     *
!                                                                       *
!  c(2)  floor value - used when the indicator c(1) has the value 3     *
!                                                                       *
!  c(3)  hmin specification - if not zero, the subroutine chooses hmin  *
!           to be abs(c(3)) - otherwise it uses the default value       *
!              10*max(dwarf,rreb*max(weighted norm y/tol,abs(x))),      *
!           where dwarf is a very small positive  machine  number  and  *
!           rreb is the relative roundoff error bound                   *
!                                                                       *
!  c(4)  hstart specification - if not zero, the subroutine  will  use  *
!           an  initial  hmag equal to abs(c(4)), except of course for  *
!           the restrictions imposed by hmin and hmax  -  otherwise it  *
!           uses the default value of hmax*(tol)**(1/6)                 *
!                                                                       *
!  c(5)  scale specification - this is intended to be a measure of the  *
!           scale of the problem - larger values of scale tend to make  *
!           the method more reliable, first  by  possibly  restricting  *
!           hmax  (as  described  below) and second, by tightening the  *
!           acceptance requirement - if c(5) is zero, a default  value  *
!           of  1  is  used.  for  linear  homogeneous  problems  with  *
!           constant coefficients, an appropriate value for scale is a  *
!           norm  of  the  associated  matrix.  for other problems, an  *
!           approximation  to  an  average  value  of  a  norm  of  the  *
!           jacobian along the trajectory may be appropriate            *
!                                                                       *
```

```
!  c(6)  hmax specification - four cases are possible                    *
!           if c(6).ne.0 and c(5).ne.0, hmax is taken to be              *
!              min(abs(c(6)),2/abs(c(5)))                                *
!           if c(6).ne.0 and c(5).eq.0, hmax is taken to be  abs(c(6))   *
!           if c(6).eq.0 and c(5).ne.0, hmax is taken to be              *
!              2/abs(c(5))                                               *
!           if c(6).eq.0 and c(5).eq.0, hmax is given a default  value   *
!              of 2                                                      *
!                                                                        *
!  c(7)  maximum number of function evaluations  -  if  not  zero,  an   *
!           error  return with ind = -1 will be caused when the number   *
!           of function evaluations exceeds abs(c(7))                    *
!                                                                        *
!  c(8)  interrupt number  1  -  if  not  zero,  the  subroutine  will   *
!           interrupt  the  calculations  after  it  has  chosen  its    *
!           preliminary value of hmag, and just before choosing htrial   *
!           and  xtrial  in  preparation for taking a step (htrial may   *
!           differ from hmag in sign, and may  require  adjustment  if   *
!           xend  is  near) - the subroutine returns with ind = 4, and   *
!           will resume calculation at the point  of  interruption  if   *
!           re-entered with ind = 4                                      *
!                                                                        *
!  c(9)  interrupt number  2  -  if  not  zero,  the  subroutine  will   *
!           interrupt  the  calculations  immediately  after  it  has    *
!           decided whether or not to accept the result  of  the  most   *
!           recent  trial step, with ind = 5 if it plans to accept, or   *
!           ind = 6 if it plans to reject -  y(*)  is  the  previously   *
!           accepted  result, while w(*,9) is the newly computed trial   *
!           value, and w(*,2) is the unweighted error estimate vector.   *
!           the  subroutine  will  resume calculations at the point of   *
!           interruption on re-entry with ind = 5 or 6. (the user  may   *
!           change ind in this case if he wishes, for example to force   *
!           acceptance of a step that would otherwise be rejected,  or   *
!           vice versa. he can also restart with ind = 1 or 2.)          *
!                                                                        *
!************************************************************************
!                                                                        *
!  summary of the components of the communications vector                *
!                                                                        *
!     prescribed at the option       determined by the program          *
!           of the user                                                  *
!                                                                        *
!                                    c(10) rreb(rel roundoff err bnd)    *
!     c(1) error control indicator   c(11) dwarf (very small mach no)    *
!     c(2) floor value               c(12) weighted norm y               *
!     c(3) hmin specification        c(13) hmin                          *
!     c(4) hstart specification      c(14) hmag                          *
!     c(5) scale specification       c(15) scale                         *
!     c(6) hmax specification        c(16) hmax                          *
!     c(7) max no of fcn evals       c(17) xtrial                        *
!     c(8) interrupt no 1            c(18) htrial                        *
!     c(9) interrupt no 2            c(19) est                           *
!                                    c(20) previous xend                 *
!                                    c(21) flag for xend                 *
!                                    c(22) no of successful steps        *
!                                    c(23) no of successive failures     *
!                                    c(24) no of fcn evals               *
!                                                                        *
!  if c(1) = 4 or 5, c(31), c(32), ... c(n+30) are floor values          *
!                                                                        *
!************************************************************************
!                                                                        *
!  an overview of the program                                            *
!                                                                        *
!     begin initialization, parameter checking, interrupt re-entries     *
!  ......abort if ind out of range 1 to 6                                *
!  .     cases - initial entry, normal re-entry, interrupt re-entries    *
!  .     case 1 - initial entry (ind .eq. 1 or 2)                        *
!  v........abort if n.gt.nw or tol.le.0                                 *
!  .        if initial entry without options (ind .eq. 1)               *
!  .           set c(1) to c(9) equal to zero                           *
```

```
!  .        else initial entry with options (ind .eq. 2)                *
!  .           make c(1) to c(9) non-negative                           *
!  .           make floor values non-negative if they are to be used    *
!  .        end if                                                      *
!  .        initialize rreb, dwarf, prev xend, flag, counts             *
!  .     case 2 - normal re-entry (ind .eq. 3)                          *
!  .....abort if xend reached, and either x changed or xend not         *
!  .        re-initialize flag                                          *
!  .     case 3 - re-entry following an interrupt (ind .eq. 4 to 6)     *
!  v        transfer control to the appropriate re-entry point.......   *
!  .     end cases                                                   .  *
!  .  end initialization, etc.                                       .  *
!  .                                                                 v  *
!  .  loop through the following 4 stages, once for each trial step .  *
!  .     stage 1 - prepare                                           .  *
!**********error return (with ind=-1) if no of fcn evals too great .  *
!  .        calc slope (adding 1 to no of fcn evals) if ind .ne. 6   .  *
!  .        calc hmin, scale, hmax                                   .  *
!**********error return (with ind=-2) if hmin .gt. hmax             .  *
!  .        calc preliminary hmag                                    .  *
!**********interrupt no 1 (with ind=4) if requested.......re-entry.v  *
!  .        calc hmag, xtrial and htrial                             .  *
!  .     end stage 1                                                 .  *
!  v     stage 2 - calc ytrial (adding 7 to no of fcn evals)         .  *
!  .     stage 3 - calc the error estimate                           .  *
!  .     stage 4 - make decisions                                    .  *
!  .        set ind=5 if step acceptable, else set ind=6             .  *
!**********interrupt no 2 if requested.....................re-entry.v  *
!  .        if step accepted (ind .eq. 5)                               *
!  .           update x, y from xtrial, ytrial                          *
!  .           add 1 to no of successful steps                          *
!  .           set no of successive failures to zero                    *
!**************return(with ind=3, xend saved, flag set) if x .eq. xend *
!  .        else step not accepted (ind .eq. 6)                         *
!  .           add 1 to no of successive failures                       *
!**************error return (with ind=-3) if hmag .le. hmin            *
!  .        end if                                                      *
!  .     end stage 4                                                    *
!  .  end loop                                                          *
!  .                                                                    *
!  begin abort action                                                   *
!     output appropriate  message  about  stopping  the  calculations, *
!        along with values of ind, n, nw, tol, hmin,  hmax,  x,  xend, *
!        previous xend,  no of  successful  steps,  no  of  successive *
!        failures, no of fcn evals, and the components of y            *
!     stop                                                            *
!  end abort action                                                    *
!                                                                      *
!**********************************************************************


!     ***********************************************************************
!     * begin initialization, parameter checking, interrupt re-entries      *
!     ***********************************************************************

!  ......abort if ind out of range 1 to 6
         if (ind.lt.1 .OR. ind.gt.6) goto 500

!        cases - initial entry, normal re-entry, interrupt re-entries
         goto (5, 5, 45, 1111, 2222, 2222), ind
!        case 1 - initial entry (ind .eq. 1 or 2)
!  .........abort if n.gt.nw or tol.le.0
     5      if (n.gt.nw .OR. tol.le.0.d0) goto 500
            if (ind.eq. 2) goto 15
!              initial entry without options (ind .eq. 1)
!              set c(1) to c(9) equal to 0
               do 10 k = 1, 9
                  c(k) = 0.d0
    10         continue
               goto 35
    15      continue
```

```
!              initial entry with options (ind .eq. 2)
!              make c(1) to c(9) non-negative
               do 20 k = 1, 9
                  c(k) = abs(c(k))
   20          continue
!              make floor values non-negative if they are to be used
               if (c(1).ne.4.d0 .AND. c(1).ne.5.d0) goto 30
                  do 25 k = 1, n
                     c(k+30) = abs(c(k+30))
   25             continue
   30          continue
   35       continue
!           initialize rreb, dwarf, prev xend, flag, counts
            c(10) = 2.d0**(-56)
            c(11) = 1.d-35
!           set previous xend initially to initial value of x
            c(20) = x
            do 40 k = 21, 24
               c(k) = 0.d0
   40       continue
            goto 50
!        case 2 - normal re-entry (ind .eq. 3)
!  .........abort if xend reached, and either x changed or xend not
   45       if (c(21).ne.0.d0 .AND. (x.ne.c(20) .OR. xend.eq.c(20))) goto 500
!           re-initialize flag
            c(21) = 0.d0
            goto 50
!        case 3 - re-entry following an interrupt (ind .eq. 4 to 6)
!           transfer control to the appropriate re-entry point.........
!           this has already been handled by the computed goto        .
!        end cases                                                     v
   50    continue

!     end initialization, etc.

!     ***********************************************************************
!     * loop through the following 4 stages, once for each trial  step      *
!     * until the occurrence of one of the following                        *
!     *    (a) the normal return (with ind .eq. 3) on reaching xend in      *
!     *        stage 4                                                      *
!     *    (b) an error return (with ind .lt. 0) in stage 1 or stage 4      *
!     *    (c) an interrupt return (with ind  .eq.  4,  5  or  6),  if      *
!     *        requested, in stage 1 or stage 4                             *
!     ***********************************************************************

99999 continue

!        ***********************************************************************
!        * stage 1 - prepare - do calculations of  hmin,  hmax,  etc.,         *
!        * and some parameter  checking,  and  end  up  with  suitable         *
!        * values of hmag, xtrial and htrial in preparation for taking         *
!        * an integration step.                                                *
!        ***********************************************************************

!***********error return (with ind=-1) if no of fcn evals too great
            if (c(7).eq.0.d0 .OR. c(24).lt.c(7)) goto 100
               ind = -1
               return
  100       continue

!           calculate slope (adding 1 to no of fcn evals) if ind .ne. 6
            if (ind .eq. 6) goto 105
               call fcn(n, x, y, w(1,1))
               c(24) = c(24) + 1.d0
  105       continue

!           calculate hmin - use default unless value prescribed
            c(13) = c(3)
            if (c(3) .ne. 0.d0) goto 165
!              calculate default value of hmin
!              first calculate weighted norm y - c(12) - as specified
```

```
!              by the error control indicator c(1)
               temp = 0.d0
               if (c(1) .ne. 1.d0) goto 115
!                 absolute error control - weights are 1
                  do 110 k = 1, n
                     temp = max(temp, abs(y(k)))
  110             continue
                  c(12) = temp
                  goto 160
  115          if (c(1) .ne. 2.d0) goto 120
!                 relative error control - weights are 1/abs(y(k)) so
!                 weighted norm y is 1
                  c(12) = 1.d0
                  goto 160
  120          if (c(1) .ne. 3.d0) goto 130
!                 weights are 1/max(c(2),abs(y(k)))
                  do 125 k = 1, n
                     temp = max(temp, abs(y(k))/c(2))
  125             continue
                  c(12) = min(temp, 1.0d0)
                  goto 160
  130          if (c(1) .ne. 4.d0) goto 140
!                 weights are 1/max(c(k+30),abs(y(k)))
                  do 135 k = 1, n
                     temp = max(temp, abs(y(k))/c(k+30))
  135             continue
                  c(12) = min(temp, 1.0d0)
                  goto 160
  140          if (c(1) .ne. 5.d0) goto 150
!                 weights are 1/c(k+30)
                  do 145 k = 1, n
                     temp = max(temp, abs(y(k))/c(k+30))
  145             continue
                  c(12) = temp
                  goto 160
  150          continue
!                 default case - weights are 1/max(1,abs(y(k)))
                  do 155 k = 1, n
                     temp = max(temp, abs(y(k)))
  155             continue
                  c(12) = min(temp, 1.0d0)
  160          continue
               c(13) = 10.d0*max(c(11),c(10)*max(c(12)/tol,abs(x)))
  165       continue

!           calculate scale - use default unless value prescribed
            c(15) = c(5)
            if (c(5) .eq. 0.d0) c(15) = 1.d0

!           calculate hmax - consider 4 cases
!           case 1 both hmax and scale prescribed
               if (c(6).ne.0.d0 .AND. c(5).ne.0.d0) c(16) = min(c(6), 2.d0/c(5))
!           case 2 - hmax prescribed, but scale not
               if (c(6).ne.0.d0 .AND. c(5).eq.0.d0) c(16) = c(6)
!           case 3 - hmax not prescribed, but scale is
               if (c(6).eq.0.d0 .AND. c(5).ne.0.d0) c(16) = 2.d0/c(5)
!           case 4 - neither hmax nor scale is provided
               if (c(6).eq.0.d0 .AND. c(5).eq.0.d0) c(16) = 2.d0

!**********error return (with ind=-2) if hmin .gt. hmax
            if (c(13) .le. c(16)) goto 170
               ind = -2
               return
  170       continue

!           calculate preliminary hmag - consider 3 cases
            if (ind .gt. 2) goto 175
!           case 1 - initial entry - use prescribed value of hstart, if
!              any, else default
               c(14) = c(4)
               if (c(4) .eq. 0.d0) c(14) = c(16)*tol**(1./6.)
```

```
               goto 185
  175       if (c(23) .gt. 1.d0) goto 180
!           case 2 - after a successful step, or at most  one  failure,
!              use min(2, .9*(tol/est)**(1/6))*hmag, but avoid possible
!              overflow. then avoid reduction by more than half.
               temp = 2.d0*c(14)
               if (tol .lt. (2.d0/.9d0)**6*c(19)) temp = .9d0*(tol/c(19))**(1./6.)*c(14)
               c(14) = max(temp, .5d0*c(14))
               goto 185
  180       continue
!           case 3 - after two or more successive failures
               c(14) = .5d0*c(14)
  185       continue

!           check against hmax
            c(14) = min(c(14), c(16))

!           check against hmin
            c(14) = max(c(14), c(13))

!***********interrupt no 1 (with ind=4) if requested
            if (c(8) .eq. 0.d0) goto 1111
               ind = 4
               return
!           resume here on re-entry with ind .eq. 4    ........re-entry..
 1111       continue

!           calculate hmag, xtrial - depending on preliminary hmag, xend
            if (c(14) .ge. abs(xend - x)) goto 190
!              do not step more than half way to xend
               c(14) = min(c(14), .5d0*abs(xend - x))
               c(17) = x + sign(c(14), xend - x)
               goto 195
  190       continue
!              hit xend exactly
               c(14) = abs(xend - x)
               c(17) = xend
  195       continue

!           calculate htrial
            c(18) = c(17) - x

!        end stage 1

!        ***************************************************************
!        * stage 2 - calculate ytrial (adding 7 to no of  fcn  evals). *
!        * w(*,2), ... w(*,8)  hold  intermediate  results  needed  in *
!        * stage 3. w(*,9) is temporary storage until finally it holds *
!        * ytrial.                                                     *
!        ***************************************************************

            temp = c(18)/1398169080000.d0
!
            do 200 k = 1, n
               w(k,9) = y(k) + temp*w(k,1)*233028180000.d0
  200       continue
            call fcn(n, x + c(18)/6.0, w(1,9), w(1,2))
!
            do 205 k = 1, n
               w(k,9) = y(k) + temp*(   w(k,1)*74569017600.d0 + w(k,2)*298276070400.d0  )
  205       continue
            call fcn(n, x + c(18)*(4.0/15.0), w(1,9), w(1,3))

            do 210 k = 1, n
               w(k,9) = y(k) + temp*(   w(k,1)*1165140900000.d0     &
                                      - w(k,2)*3728450880000.d0     &
                                      + w(k,3)*3495422700000.d0 )
  210       continue
            call fcn(n, x + c(18)*(2.0/3.0), w(1,9), w(1,4))

            do 215 k = 1, n
```

```
               w(k,9) = y(k) + temp*( - w(k,1)*3604654659375.d0      &
                                      + w(k,2)*12816549900000.d0     &
                                      - w(k,3)*9284716546875.d0      &
                                      + w(k,4)*1237962206250.d0 )
  215       continue
            call fcn(n, x + c(18)*(5.0/6.0), w(1,9), w(1,5))

            do 220 k = 1, n
               w(k,9) = y(k) + temp*(   w(k,1)*3355605792000.d0      &
                                      - w(k,2)*11185352640000.d0     &
                                      + w(k,3)*9172628850000.d0      &
                                      - w(k,4)*427218330000.d0       &
                                      + w(k,5)*482505408000.d0  )
  220       continue
            call fcn(n, x + c(18), w(1,9), w(1,6))

            do 225 k = 1, n
               w(k,9) = y(k) + temp*( - w(k,1)*770204740536.d0       &
                                      + w(k,2)*2311639545600.d0      &
                                      - w(k,3)*1322092233000.d0      &
                                      - w(k,4)*453006781920.d0       &
                                      + w(k,5)*326875481856.d0  )
  225       continue
            call fcn(n, x + c(18)/15.0, w(1,9), w(1,7))

            do 230 k = 1, n
               w(k,9) = y(k) + temp*(   w(k,1)*2845924389000.d0      &
                                      - w(k,2)*9754668000000.d0      &
                                      + w(k,3)*7897110375000.d0      &
                                      - w(k,4)*192082660000.d0       &
                                      + w(k,5)*400298976000.d0       &
                                      + w(k,7)*201586000000.d0  )
  230       continue
            call fcn(n, x + c(18), w(1,9), w(1,8))

!           calculate ytrial, the extrapolated approximation and store
!              in w(*,9)
            do 235 k = 1, n
               w(k,9) = y(k) + temp*(   w(k,1)*104862681000.d0       &
                                      + w(k,3)*545186250000.d0       &
                                      + w(k,4)*446637345000.d0       &
                                      + w(k,5)*188806464000.d0       &
                                      + w(k,7)*15076875000.d0        &
                                      + w(k,8)*97599465000.d0   )
  235       continue

!           add 7 to the no of fcn evals
            c(24) = c(24) + 7.d0

!        end stage 2

!        ***************************************************************
!        * stage 3 - calculate the error estimate est. first calculate *
!        * the  unweighted  absolute  error  estimate vector (per unit *
!        * step) for the unextrapolated approximation and store it  in *
!        * w(*,2).  then  calculate the weighted max norm of w(*,2) as *
!        * specified by the error  control  indicator  c(1).  finally, *
!        * modify  this result to produce est, the error estimate (per *
!        * unit step) for the extrapolated approximation ytrial.       *
!        ***************************************************************

!           calculate the unweighted absolute error estimate vector
            do 300 k = 1, n
               w(k,2) = (   w(k,1)*8738556750.d0                      &
                          + w(k,3)*9735468750.d0                      &
                          - w(k,4)*9709507500.d0                      &
                          + w(k,5)*8582112000.d0                      &
                          + w(k,6)*95329710000.d0                     &
                          - w(k,7)*15076875000.d0                     &
                          - w(k,8)*97599465000.d0)/1398169080000.d0
  300       continue
```

```
!           calculate the weighted max norm of w(*,2) as specified by
!           the error control indicator c(1)
            temp = 0.d0
            if (c(1) .ne. 1.d0) goto 310
!              absolute error control
               do 305 k = 1, n
                  temp = max(temp,abs(w(k,2)))
  305          continue
               goto 360
  310       if (c(1) .ne. 2.d0) goto 320
!              relative error control
               do 315 k = 1, n
                  temp = max(temp, abs(w(k,2)/y(k)))
  315          continue
               goto 360
  320       if (c(1) .ne. 3.d0) goto 330
!              weights are 1/max(c(2),abs(y(k)))
               do 325 k = 1, n
                  temp = max(temp, abs(w(k,2)) / max(c(2), abs(y(k))) )
  325          continue
               goto 360
  330       if (c(1) .ne. 4.d0) goto 340
!              weights are 1/max(c(k+30),abs(y(k)))
               do 335 k = 1, n
                  temp = max(temp, abs(w(k,2)) / max(c(k+30), abs(y(k))) )
  335          continue
               goto 360
  340       if (c(1) .ne. 5.d0) goto 350
!              weights are 1/c(k+30)
               do 345 k = 1, n
                  temp = max(temp, abs(w(k,2)/c(k+30)))
  345          continue
               goto 360
  350       continue
!              default case - weights are 1/max(1,abs(y(k)))
               do 355 k = 1, n
                  temp = max(temp, abs(w(k,2)) / max(1.0, abs(y(k))) )
  355          continue
  360       continue

!           calculate est - (the weighted max norm of w(*,2))*hmag*scale
!              - est is intended to be a measure of the error  per  unit
!              step in ytrial
            c(19) = temp*c(14)*c(15)

!        end stage 3

!        ***********************************************************
!        * stage 4 - make decisions.                               *
!        ***********************************************************

!           set ind=5 if step acceptable, else set ind=6
            ind = 5
            if (c(19) .gt. tol) ind = 6

!***********interrupt no 2 if requested
            if (c(9) .eq. 0.d0) goto 2222
               return
!           resume here on re-entry with ind .eq. 5 or 6    ...re-entry..
 2222       continue

            if (ind .eq. 6) goto 410
!              step accepted (ind .eq. 5), so update x, y from xtrial,
!                 ytrial, add 1 to the no of successful steps, and set
!                 the no of successive failures to zero
               x = c(17)
               do 400 k = 1, n
                  y(k) = w(k,9)
  400          continue
               c(22) = c(22) + 1.d0
```

```
               c(23) = 0.d0
!**************return(with ind=3, xend saved, flag set) if x .eq. xend
               if (x .ne. xend) goto 405
                  ind = 3
                  c(20) = xend
                  c(21) = 1.d0
                  return
  405          continue
               goto 420
  410       continue
!                step not accepted (ind .eq. 6), so add 1 to the no of
!                   successive failures
               c(23) = c(23) + 1.d0
!**************error return (with ind=-3) if hmag .le. hmin
               if (c(14) .gt. c(13)) goto 415
                  ind = -3
                  return
  415          continue
  420       continue

!        end stage 4

      goto 99999
!     end loop

!  begin abort action
  500 continue

!      write(6,505) ind, tol, x, n, c(13), xend, nw, c(16), c(20),  &
!            c(22), c(23), c(24), (y(k), k = 1, n)
!  505 format( /// 1h0, 58hcomputation stopped in dverk with the following values - / 1h0
,
!       5hind =, i4, 5x, 6htol  =, 1pd13.6, 5x, 11hx         =,              &
!              1pd22.15                                                     &
!        / 1h , 5hn   =, i4, 5x, 6hhmin =, 1pd13.6, 5x, 11hxend      =,     &
!              1pd22.15                                                     &
!        / 1h , 5hnw  =, i4, 5x, 6hhmax =, 1pd13.6, 5x, 11hprev xend =,     &
!              1pd22.15                                                     &
!        / 1h0, 14x, 27hno of successful steps    =, 0pf8.0                 &
!        / 1h , 14x, 27hno of successive failures =, 0pf8.0                 &
!        / 1h , 14x, 27hno of function evals      =, 0pf8.0                 &
!        / 1h0, 23hthe components of y are                                  &
!        // (1h , 1p5d24.15)                                                )

RETURN

!  end abort action

END SUBROUTINE DVERK


!     ALGORITHM 474 COLLECTED ALGORITHMS FROM ACM.
!     ALGORITHM APPEARED IN COMM. ACM, VOL. 17, NO. 1,
!     P. 26.
SUBROUTINE ITPLBV(IU, LX, LY, X, Y, Z, N, U, V, W)
    IMPLICIT NONE

! BIVARIATE INTERPOLATION
! THIS SUBROUTINE INTERPOLATES, FROM VALUES OF THE FUNCTION
! GIVEN AT INPUT GRID POINTS IN AN X-Y PLANE AND FOR A GIVEN
! SET OF POINTS IN THE PLANE, THE VALUES OF A SINGLE-VALUED
! BIVARIATE FUNCTION Z = Z(X,Y).
! THE METHOD IS BASED ON A PIECE-WISE FUNCTION COMPOSED OF
! A SET OF BICUBIC POLYNOMIALS IN X AND Y.  EACH POLYNOMIAL
! IS APPLICABLE TO A RECTANGLE OF THE INPUT GRID IN THE X-Y
! PLANE.  EACH POLYNOMIAL IS DETERMINED LOCALLY.
! THE INPUT PARAMETERS ARE
! IU  = LOGICAL UNIT NUMBER OF STANDARD OUTPUT UNIT
! LX  = NUMBER OF INPUT GRID POINTS IN THE X COORDINATE
!       (MUST BE 2 OR GREATER)
! LY  = NUMBER OF INPUT GRID POINTS IN THE Y COORDINATE
```

```
!       (MUST BE 2 OR GREATER)
! X   = ARRAY OF DIMENSION LX STORING THE X COORDINATES
!       OF INPUT GRID POINTS (IN ASCENDING ORDER)
! Y   = ARRAY OF DIMENSION LY STORING THE Y COORDINATES
!       OF INPUT GRID POINTS (IN ASCENDING ORDER)
! Z   = DOUBLY-DIMENSIONED ARRAY OF DIMENSION (LX,LY)
!       STORING THE VALUES OF THE FUNCTION (Z VALUES)
!       AT INPUT GRID POINTS
! N   = NUMBER OF POINTS AT WHICH INTERPOLATION OF THE
!       Z VALUE IS DESIRED (MUST BE 1 OR GREATER)
! U   = ARRAY OF DIMENSION N STORING THE X COORDINATES
!       OF DESIRED POINTS
! V   = ARRAY OF DIMENSION N STORING THE Y COORDINATES
!       OF DESIRED POINTS
! THE OUTPUT PARAMETER IS
! W   = ARRAY OF DIMENSION N WHERE THE INTERPOLATED Z
!       VALUES AT DESIRED POINTS ARE TO BE DISPLAYED
!       SOME VARIABLES INTERNALLY USED ARE
! ZA  = DIVIDED DIFFERENCE OF Z WITH RESPECT TO X
! ZB  = DIVIDED DIFFERENCE OF Z WITH RESPECT TO Y
! ZAB = SECOND ORDER DIVIDED DIFFERENCE OF Z WITH
!       RESPECT TO X AND Y
! ZX  = PARTIAL DERIVATIVE OF Z WITH RESPECT TO X
! ZY  = PARTIAL DERIVATIVE OF Z WITH RESPECT TO Y
! ZXY = SECOND ORDER PARTIAL DERIVATIVE OF Z WITH
!       RESPECT TO X AND Y

! DECLARATION STATEMENTS
    INTEGER IU, LX, LY, N
    DOUBLE PRECISION X, Y, Z, U, V, W

    INTEGER IU0, LX0, LXM1, LXM2, LXP1, LY0, LYM1, LYM2, LYP1, N0
    INTEGER IX, IY, IXPV, IYPV, K, IMN, IMX, JX, JY, JXM2, JXML
    INTEGER JYM2, JYML, JX1, JY1
    DOUBLE PRECISION UK, VK, X3, X4, A3, Y3, Y4, B3, Z33, Z43, Z34
    DOUBLE PRECISION Z44, Z3A3, Z4A3, Z3B3, Z4B3, ZA3B3, X2, A2
    DOUBLE PRECISION Z23, Z24, Z3A2, Z4A2, X5, A4, Z53, Z54, Z3A4
    DOUBLE PRECISION A1, Z3A1, Z4A1, A5, Z3A5, Z4A5, Y2, B2, Z32
    DOUBLE PRECISION Z42, Z3B2, Z4B2, Y5, B4, Z35, Z45, Z3B4, Z4B4
    DOUBLE PRECISION ZA3B2, ZA3B4, B1, Z3B1, Z4B1, B5, Z3B5, Z4B5
    DOUBLE PRECISION ZA4B2, ZA4B4, ZA2B2, ZA2B4, W2, ZA, W3, SW
    DOUBLE PRECISION WX2, WX3, ZX, ZB, WY2, WY3, ZY, ZXY, ZAB, W1
    DOUBLE PRECISION W4, W5, Z4A4, ZA2B3, ZA4B3, ZX3B3
    DOUBLE PRECISION ZX34, ZX33, ZX4B3, ZX44, ZX43, ZY3A3, ZY43, ZY33
    DOUBLE PRECISION ZY4A3, ZY44, ZY34, A, ZXY33, B, ZXY43, C, ZXY34
    DOUBLE PRECISION D, ZXY44, E, A3SQ, B3SQ, P02, P03, P12, P13, P20
    DOUBLE PRECISION P21, P22, P23, P30, P31, P32, P33, DY, Q0, P00
    DOUBLE PRECISION P01, Q1, P10, P11, Q2, Q3, DX

    DIMENSION X(LX), Y(LY), Z(LX,LY), U(N), V(N), W(N)
    DIMENSION ZA(5,2), ZB(2,5), ZAB(3,3), ZX(4,4), ZY(4,4), &
        ZXY(4,4)
    EQUIVALENCE (Z3A1,ZA(1,1)), (Z3A2,ZA(2,1)), (Z3A3,ZA(3,1)),      &
        (Z3A4,ZA(4,1)), (Z3A5,ZA(5,1)), (Z4A1,ZA(1,2)), (Z4A2,ZA(2,2)), &
        (Z4A3,ZA(3,2)), (Z4A4,ZA(4,2)), (Z4A5,ZA(5,2)), (Z3B1,ZB(1,1)), &
        (Z3B2,ZB(1,2)), (Z3B3,ZB(1,3)), (Z3B4,ZB(1,4)), (Z3B5,ZB(1,5)), &
        (Z4B1,ZB(2,1)), (Z4B2,ZB(2,2)), (Z4B3,ZB(2,3)), (Z4B4,ZB(2,4)), &
        (Z4B5,ZB(2,5)), (ZA2B2,ZAB(1,1)), (ZA3B2,ZAB(2,1)),              &
        (ZA4B2,ZAB(3,1)), (ZA2B3,ZAB(1,2)), (ZA3B3,ZAB(2,2)),            &
        (ZA4B3,ZAB(3,2)), (ZA2B4,ZAB(1,3)), (ZA3B4,ZAB(2,3)),            &
        (ZA4B4,ZAB(3,3)), (ZX33,ZX(2,2)), (ZX43,ZX(3,2)),                &
        (ZX34,ZX(2,3)), (ZX44,ZX(3,3)), (ZY33,ZY(2,2)),                  &
        (ZY43,ZY(3,2)), (ZY34,ZY(2,3)), (ZY44,ZY(3,3)),                  &
        (ZXY33,ZXY(2,2)), (ZXY43,ZXY(3,2)), (ZXY34,ZXY(2,3)),            &
        (ZXY44,ZXY(3,3)), (P00,Z33), (P01,ZY33), (P10,ZX33),             &
        (P11,ZXY33)
    EQUIVALENCE (LX0,ZX(1,1)), (LXM1,ZX(4,1)), (LXM2,ZX(1,4)),       &
        (LXP1,ZX(4,4)), (LY0,ZY(1,1)), (LYM1,ZY(4,1)), (LYM2,ZY(1,4)),  &
        (LYP1,ZY(4,4)), (IX,ZXY(1,1)), (IY,ZXY(4,1)), (IXPV,ZXY(1,4)),  &
        (IYPV,ZXY(4,4)), (IMN,JX), (IMX,JY), (JXM2,JX1),                &
        (JYM2,JY1), (UK,DX), (VK,DY), (A1,A5,B1,B5,ZX(2,1),A,Q0),       &
```

```
     (A2,ZX(1,2),B,Q1), (A4,ZX(4,2),C,Q2), (B2,ZY(2,1),D,Q3),          &
     (B4,ZY(2,4),E), (X2,ZX(3,1),A3SQ), (X4,ZX(1,3)), (X5,ZX(4,3)),   &
     (Y2,ZX(2,4)), (Y4,ZY(3,1),B3SQ), (Y5,ZX(3,4),P02),               &
     (Z23,ZY(1,2),P03), (Z24,ZY(4,2),P12), (Z32,ZY(1,3),P13),         &
     (Z34,ZY(4,3),P20), (Z35,ZY(3,4),P21), (Z42,ZXY(2,1),P22),        &
     (Z43,ZXY(1,2),P23), (Z44,ZXY(3,1),P30), (Z45,ZXY(4,2),P31),      &
     (Z53,ZXY(1,3),P32), (Z54,ZXY(4,3),P33), (W2,WY2,W4),             &
     (W3,WY3,W1,W5), (WX2,ZXY(2,4)), (WX3,ZXY(3,4))

! PRELIMINARY PROCESSING
! SETTING OF SOME INPUT PARAMETERS TO LOCAL VARIABLES
      IU0 = IU
      LX0 = LX
      LXM1 = LX0 - 1
      LXM2 = LXM1 - 1
      LXP1 = LX0 + 1
      LY0 = LY
      LYM1 = LY0 - 1
      LYM2 = LYM1 - 1
      LYP1 = LY0 + 1
      N0 = N

! ERROR CHECK
      IF (LXM2.LT.0) GOTO 710
      IF (LYM2.LT.0) GOTO 720
      IF (N0.LT.1) GOTO 730

      DO 10 IX=2,LX0
          IF (X(IX-1)-X(IX)) 10, 740, 750
   10 CONTINUE

      DO 20 IY=2,LY0
          IF (Y(IY-1)-Y(IY)) 20, 770, 780
   20 CONTINUE

! INITIAL SETTING OF PREVIOUS VALUES OF IX AND IY
       IXPV = 0
       IYPV = 0

! MAIN DO-LOOP
      DO 700 K=1,N0
          UK = U(K)
          VK = V(K)

! ROUTINES TO LOCATE THE DESIRED POINT
! TO FIND OUT THE IX VALUE FOR WHICH
! (U(K).GE.X(IX-1)).AND.(U(K).LT.X(IX))
          IF (LXM2.EQ.0) GOTO 80
          IF (UK.GE.X(LX0)) GOTO 70
          IF (UK.LT.X(1)) GOTO 60
          IMN = 2
          IMX = LX0
   30     IX = (IMN+IMX)/2
          IF (UK.GE.X(IX)) GOTO 40
          IMX = IX
          GOTO 50

   40     IMN = IX + 1
   50     IF (IMX.GT.IMN) GOTO 30
          IX = IMX
          GOTO 90

   60     IX = 1
          GOTO 90

   70     IX = LXP1
          GOTO 90

   80     IX = 2

! TO FIND OUT THE IY VALUE FOR WHICH
```

```
! (V(K).GE.Y(IY-1)).AND.(V(K).LT.Y(IY))
   90   IF (LYM2.EQ.0) GOTO 150
        IF (VK.GE.Y(LY0)) GOTO 140
        IF (VK.LT.Y(1)) GOTO 130
        IMN = 2
        IMX = LY0
  100   IY = (IMN+IMX)/2
        IF (VK.GE.Y(IY)) GOTO 110
        IMX = IY
        GOTO 120

  110   IMN = IY + 1
  120   IF (IMX.GT.IMN) GOTO 100
        IY = IMX
        GOTO 160

  130   IY = 1
        GOTO 160

  140   IY = LYP1
        GOTO 160

  150   IY = 2

! TO CHECK IF THE DESIRED POINT IS IN THE SAME RECTANGLE
! AS THE PREVIOUS POINT.  IF YES, SKIP TO THE COMPUTATION
! OF THE POLYNOMIAL
  160   IF (IX.EQ.IXPV .AND. IY.EQ.IYPV) GOTO 690
        IXPV = IX
        IYPV = IY

! ROUTINES TO PICK UP NECESSARY X, Y, AND Z VALUES, TO
! COMPUTE THE ZA, ZB, AND ZAB VALUES, AND TO ESTIMATE THEM
! WHEN NECESSARY
        JX = IX
        IF (JX.EQ.1) JX = 2
        IF (JX.EQ.LXP1) JX = LX0
        JY = IY
        IF (JY.EQ.1) JY = 2
        IF (JY.EQ.LYP1) JY = LY0
        JXM2 = JX - 2
        JXML = JX - LX0
        JYM2 = JY - 2
        JYML = JY - LY0

! IN THE CORE AREA, I.E., IN THE RECTANGLE THAT CONTAINS
! THE DESIRED POINT
        X3 = X(JX-1)
        X4 = X(JX)
        A3 = 1.0/(X4-X3)
        Y3 = Y(JY-1)
        Y4 = Y(JY)
        B3 = 1.0/(Y4-Y3)
        Z33 = Z(JX-1,JY-1)
        Z43 = Z(JX,JY-1)
        Z34 = Z(JX-1,JY)
        Z44 = Z(JX,JY)
        Z3A3 = (Z43-Z33)*A3
        Z4A3 = (Z44-Z34)*A3
        Z3B3 = (Z34-Z33)*B3
        Z4B3 = (Z44-Z43)*B3
        ZA3B3 = (Z4B3-Z3B3)*A3

! IN THE X DIRECTION
        IF (LXM2.EQ.0) GOTO 230
        IF (JXM2.EQ.0) GOTO 170
        X2 = X(JX-2)
        A2 = 1.0/(X3-X2)
        Z23 = Z(JX-2,JY-1)
        Z24 = Z(JX-2,JY)
        Z3A2 = (Z33-Z23)*A2
```

```
      Z4A2 = (Z34-Z24)*A2
      IF (JXML.EQ.0) GOTO 180
  170 X5 = X(JX+1)
      A4 = 1.0/(X5-X4)
      Z53 = Z(JX+1,JY-1)
      Z54 = Z(JX+1,JY)
      Z3A4 = (Z53-Z43)*A4
      Z4A4 = (Z54-Z44)*A4
      IF (JXM2.NE.0) GOTO 190
      Z3A2 = Z3A3 + Z3A3 - Z3A4
      Z4A2 = Z4A3 + Z4A3 - Z4A4
      GOTO 190

  180 Z3A4 = Z3A3 + Z3A3 - Z3A2
      Z4A4 = Z4A3 + Z4A3 - Z4A2
  190 ZA2B3 = (Z4A2-Z3A2)*B3
      ZA4B3 = (Z4A4-Z3A4)*B3
      IF (JX.LE.3) GOTO 200
      A1 = 1.0/(X2-X(JX-3))
      Z3A1 = (Z23-Z(JX-3,JY-1))*A1
      Z4A1 = (Z24-Z(JX-3,JY))*A1
      GOTO 210

  200 Z3A1 = Z3A2 + Z3A2 - Z3A3
      Z4A1 = Z4A2 + Z4A2 - Z4A3
  210 IF (JX.GE.LXM1) GOTO 220
      A5 = 1.0/(X(JX+2)-X5)
      Z3A5 = (Z(JX+2,JY-1)-Z53)*A5
      Z4A5 = (Z(JX+2,JY)-Z54)*A5
      GOTO 240

  220 Z3A5 = Z3A4 + Z3A4 - Z3A3
      Z4A5 = Z4A4 + Z4A4 - Z4A3
      GOTO 240

  230 Z3A2 = Z3A3
      Z4A2 = Z4A3
      GOTO 180

! IN THE Y DIRECTION
  240 IF (LYM2.EQ.0) GOTO 310
      IF (JYM2.EQ.0) GOTO 250
      Y2 = Y(JY-2)
      B2 = 1.0/(Y3-Y2)
      Z32 = Z(JX-1,JY-2)
      Z42 = Z(JX,JY-2)
      Z3B2 = (Z33-Z32)*B2
      Z4B2 = (Z43-Z42)*B2
      IF (JYML.EQ.0) GOTO 260
  250 Y5 = Y(JY+1)
      B4 = 1.0/(Y5-Y4)
      Z35 = Z(JX-1,JY+1)
      Z45 = Z(JX,JY+1)
      Z3B4 = (Z35-Z34)*B4
      Z4B4 = (Z45-Z44)*B4
      IF (JYM2.NE.0) GOTO 270
      Z3B2 = Z3B3 + Z3B3 - Z3B4
      Z4B2 = Z4B3 + Z4B3 - Z4B4
      GOTO 270

  260 Z3B4 = Z3B3 + Z3B3 - Z3B2
      Z4B4 = Z4B3 + Z4B3 - Z4B2
  270 ZA3B2 = (Z4B2-Z3B2)*A3
      ZA3B4 = (Z4B4-Z3B4)*A3
      IF (JY.LE.3) GOTO 280
      B1 = 1.0/(Y2-Y(JY-3))
      Z3B1 = (Z32-Z(JX-1,JY-3))*B1
      Z4B1 = (Z42-Z(JX,JY-3))*B1
      GOTO 290

  280 Z3B1 = Z3B2 + Z3B2 - Z3B3
```

```
      Z4B1 = Z4B2 + Z4B2 - Z4B3
290   IF (JY.GE.LYM1) GOTO 300
      B5 = 1.0/(Y(JY+2)-Y5)
      Z3B5 = (Z(JX-1,JY+2)-Z35)*B5
      Z4B5 = (Z(JX,JY+2)-Z45)*B5
      GOTO 320

300   Z3B5 = Z3B4 + Z3B4 - Z3B3
      Z4B5 = Z4B4 + Z4B4 - Z4B3
      GOTO 320

310   Z3B2 = Z3B3
      Z4B2 = Z4B3
      GOTO 260

! IN THE DIAGONAL DIRECTIONS
320   IF (LXM2.EQ.0) GOTO 400
      IF (LYM2.EQ.0) GOTO 410
      IF (JXML.EQ.0) GOTO 350
      IF (JYM2.EQ.0) GOTO 330
      ZA4B2 = ((Z53-Z(JX+1,JY-2))*B2-Z4B2)*A4
      IF (JYML.EQ.0) GOTO 340
330   ZA4B4 = ((Z(JX+1,JY+1)-Z54)*B4-Z4B4)*A4
      IF (JYM2.NE.0) GOTO 380
      ZA4B2 = ZA4B3 + ZA4B3 - ZA4B4
      GOTO 380

340   ZA4B4 = ZA4B3 + ZA4B3 - ZA4B2
      GOTO 380

350   IF (JYM2.EQ.0) GOTO 360
      ZA2B2 = (Z3B2-(Z23-Z(JX-2,JY-2))*B2)*A2
      IF (JYML.EQ.0) GOTO 370
360   ZA2B4 = (Z3B4-(Z(JX-2,JY+1)-Z24)*B4)*A2
      IF (JYM2.NE.0) GOTO 390
      ZA2B2 = ZA2B3 + ZA2B3 - ZA2B4
      GOTO 390

370   ZA2B4 = ZA2B3 + ZA2B3 - ZA2B2
      GOTO 390

380   IF (JXM2.NE.0) GOTO 350
      ZA2B2 = ZA3B2 + ZA3B2 - ZA4B2
      ZA2B4 = ZA3B4 + ZA3B4 - ZA4B4
      GOTO 420

390   IF (JXML.NE.0) GOTO 420
      ZA4B2 = ZA3B2 + ZA3B2 - ZA2B2
      ZA4B4 = ZA3B4 + ZA3B4 - ZA2B4
      GOTO 420

400   ZA2B2 = ZA3B2
      ZA4B2 = ZA3B2
      ZA2B4 = ZA3B4
      ZA4B4 = ZA3B4
      GOTO 420

410   ZA2B2 = ZA2B3
      ZA2B4 = ZA2B3
      ZA4B2 = ZA4B3
      ZA4B4 = ZA4B3

! NUMERICAL DIFFERENTIATION   ---   TO DETERMINE PARTIAL
! DERIVATIVES ZX, ZY, AND ZXY AS WEIGHTED MEANS OF DIVIDED
! DIFFERENCES ZA, ZB, AND ZAB, RESPECTIVELY
420   DO 480 JY=2,3
          DO 470 JX=2,3

              W2 = ABS(ZA(JX+2,JY-1)-ZA(JX+1,JY-1))
              W3 = ABS(ZA(JX,JY-1)-ZA(JX-1,JY-1))
              SW = W2 + W3
```

```
                IF (SW.EQ.0.0) GOTO 430
                WX2 = W2/SW
                WX3 = W3/SW
                GOTO 440

  430           WX2 = 0.5
                WX3 = 0.5
  440           ZX(JX,JY) = WX2*ZA(JX,JY-1) + WX3*ZA(JX+1,JY-1)
                W2 = ABS(ZB(JX-1,JY+2)-ZB(JX-1,JY+1))
                W3 = ABS(ZB(JX-1,JY)-ZB(JX-1,JY-1))
                SW = W2 + W3
                IF (SW.EQ.0.0) GOTO 450
                WY2 = W2/SW
                WY3 = W3/SW
                GOTO 460

  450           WY2 = 0.5
                WY3 = 0.5
  460           ZY(JX,JY) = WY2*ZB(JX-1,JY) + WY3*ZB(JX-1,JY+1)
                ZXY(JX,JY) = WY2*(WX2*ZAB(JX-1,JY-1)+WX3*ZAB(JX,JY-1)) + &
                    WY3*(WX2*ZAB(JX-1,JY)+WX3*ZAB(JX,JY))

  470       CONTINUE
  480   CONTINUE

! WHEN (U(K).LT.X(1)).OR.(U(K).GT.X(LX))
        IF (IX.EQ.LXP1) GOTO 530
        IF (IX.NE.1) GOTO 590
        W2 = A4*(3.0*A3+A4)
        W1 = 2.0*A3*(A3-A4) + W2

        DO 500 JY=2,3

            ZX(1,JY) = (W1*ZA(1,JY-1)+W2*ZA(2,JY-1))/(W1+W2)
            ZY(1,JY) = ZY(2,JY) + ZY(2,JY) - ZY(3,JY)
            ZXY(1,JY) = ZXY(2,JY) + ZXY(2,JY) - ZXY(3,JY)

            DO 490 JX1=2,3

                JX = 5 - JX1
                ZX(JX,JY) = ZX(JX-1,JY)
                ZY(JX,JY) = ZY(JX-1,JY)
                ZXY(JX,JY) = ZXY(JX-1,JY)

  490       CONTINUE
  500   CONTINUE

        X3 = X3 - 1.0/A4
        Z33 = Z33 - Z3A2/A4

        DO 510 JY=1,5
            ZB(2,JY) = ZB(1,JY)
  510   CONTINUE

        DO 520 JY=2,4
            ZB(1,JY) = ZB(1,JY) - ZAB(1,JY-1)/A4
  520   CONTINUE

        A3 = A4
        JX = 1
        GOTO 570

  530   W4 = A2*(3.0*A3+A2)
        W5 = 2.0*A3*(A3-A2) + W4

        DO 550 JY=2,3

            ZX(4,JY) = (W4*ZA(4,JY-1)+W5*ZA(5,JY-1))/(W4+W5)
            ZY(4,JY) = ZY(3,JY) + ZY(3,JY) - ZY(2,JY)
            ZXY(4,JY) = ZXY(3,JY) + ZXY(3,JY) - ZXY(2,JY)
```

```
            DO 540 JX=2,3

                ZX(JX,JY) = ZX(JX+1,JY)
                ZY(JX,JY) = ZY(JX+1,JY)
                ZXY(JX,JY) = ZXY(JX+1,JY)

  540       CONTINUE
  550   CONTINUE

        X3 = X4
        Z33 = Z43

        DO 560 JY=1,5
            ZB(1,JY) = ZB(2,JY)
  560   CONTINUE

        A3 = A2
        JX = 3
  570   ZA(3,1) = ZA(JX+1,1)

        DO 580 JY=1,3
            ZAB(2,JY) = ZAB(JX,JY)
  580   CONTINUE

! WHEN (V(K).LT.Y(1)).OR.(V(K).GT.Y(LY))
  590   IF (IY.EQ.LYP1) GOTO 630
        IF (IY.NE.1) GOTO 680
        W2 = B4*(3.0*B3+B4)
        W1 = 2.0*B3*(B3-B4) + W2

        DO 620 JX=2,3

            IF (JX.EQ.3 .AND. IX.EQ.LXP1) GOTO 600
            IF (JX.EQ.2 .AND. IX.EQ.1) GOTO 600
            ZY(JX,1) = (W1*ZB(JX-1,1)+W2*ZB(JX-1,2))/(W1+W2)
            ZX(JX,1) = ZX(JX,2) + ZX(JX,2) - ZX(JX,3)
            ZXY(JX,1) = ZXY(JX,2) + ZXY(JX,2) - ZXY(JX,3)

  600       DO 610 JY1=2,3

                JY = 5 - JY1
                ZY(JX,JY) = ZY(JX,JY-1)
                ZX(JX,JY) = ZX(JX,JY-1)
                ZXY(JX,JY) = ZXY(JX,JY-1)

  610       CONTINUE
  620   CONTINUE

        Y3 = Y3 - 1.0/B4
        Z33 = Z33 - Z3B2/B4
        Z3A3 = Z3A3 - ZA3B2/B4
        Z3B3 = Z3B2
        ZA3B3 = ZA3B2
        B3 = B4
        GOTO 670

  630   W4 = B2*(3.0*B3+B2)
        W5 = 2.0*B3*(B3-B2) + W4

        DO 660 JX=2,3
            IF (JX.EQ.3 .AND. IX.EQ.LXP1) GOTO 640
            IF (JX.EQ.2 .AND. IX.EQ.1) GOTO 640
            ZY(JX,4) = (W4*ZB(JX-1,4)+W5*ZB(JX-1,5))/(W4+W5)
            ZX(JX,4) = ZX(JX,3) + ZX(JX,3) - ZX(JX,2)
            ZXY(JX,4) = ZXY(JX,3) + ZXY(JX,3) - ZXY(JX,2)
  640       DO 650 JY=2,3
                ZY(JX,JY) = ZY(JX,JY+1)
                ZX(JX,JY) = ZX(JX,JY+1)
                ZXY(JX,JY) = ZXY(JX,JY+1)

  650       CONTINUE
```

```
  660   CONTINUE

        Y3 = Y4
        Z33 = Z33 + Z3B3/B3
        Z3A3 = Z3A3 + ZA3B3/B3
        Z3B3 = Z3B4
        ZA3B3 = ZA3B4
        B3 = B2
  670   IF (IX.NE.1 .AND. IX.NE.LXP1) GOTO 680
        JX = IX/LXP1 + 2
        JX1 = 5 - JX
        JY = IY/LYP1 + 2
        JY1 = 5 - JY
        ZX(JX,JY) = ZX(JX1,JY) + ZX(JX,JY1) - ZX(JX1,JY1)
        ZY(JX,JY) = ZY(JX1,JY) + ZY(JX,JY1) - ZY(JX1,JY1)
        ZXY(JX,JY) = ZXY(JX1,JY) + ZXY(JX,JY1) - ZXY(JX1,JY1)

! DETERMINATION OF THE COEFFICIENTS OF THE POLYNOMIAL
  680   ZX3B3 = (ZX34-ZX33)*B3
        ZX4B3 = (ZX44-ZX43)*B3
        ZY3A3 = (ZY43-ZY33)*A3
        ZY4A3 = (ZY44-ZY34)*A3
        A = ZA3B3 - ZX3B3 - ZY3A3 + ZXY33
        B = ZX4B3 - ZX3B3 - ZXY43 + ZXY33
        C = ZY4A3 - ZY3A3 - ZXY34 + ZXY33
        D = ZXY44 - ZXY43 - ZXY34 + ZXY33
        E = A + A - B - C
        A3SQ = A3*A3
        B3SQ = B3*B3
        P02 = (2.0*(Z3B3-ZY33)+Z3B3-ZY34)*B3
        P03 = (-2.0*Z3B3+ZY34+ZY33)*B3SQ
        P12 = (2.0*(ZX3B3-ZXY33)+ZX3B3-ZXY34)*B3
        P13 = (-2.0*ZX3B3+ZXY34+ZXY33)*B3SQ
        P20 = (2.0*(Z3A3-ZX33)+Z3A3-ZX43)*A3
        P21 = (2.0*(ZY3A3-ZXY33)+ZY3A3-ZXY43)*A3
        P22 = (3.0*(A+E)+D)*A3*B3
        P23 = (-3.0*E-B-D)*A3*B3SQ
        P30 = (-2.0*Z3A3+ZX43+ZX33)*A3SQ
        P31 = (-2.0*ZY3A3+ZXY43+ZXY33)*A3SQ
        P32 = (-3.0*E-C-D)*B3*A3SQ
        P33 = (D+E+E)*A3SQ*B3SQ

! COMPUTATION OF THE POLYNOMIAL
  690   DY = VK - Y3
        Q0 = P00 + DY*(P01+DY*(P02+DY*P03))
        Q1 = P10 + DY*(P11+DY*(P12+DY*P13))
        Q2 = P20 + DY*(P21+DY*(P22+DY*P23))
        Q3 = P30 + DY*(P31+DY*(P32+DY*P33))
        DX = UK - X3
        W(K) = Q0 + DX*(Q1+DX*(Q2+DX*Q3))
  700 CONTINUE

! NORMAL EXIT
      RETURN

! ERROR EXIT
  710 WRITE (IU0,99999)
      GOTO 800
  720 WRITE (IU0,99998)
      GOTO 800
  730 WRITE (IU0,99997)
      GOTO 800
  740 WRITE (IU0,99996)
      GOTO 760
  750 WRITE (IU0,99995)
  760 WRITE (IU0,99994) IX, X(IX)
      GOTO 800
  770 WRITE (IU0,99993)
      GOTO 790
  780 WRITE (IU0,99992)
  790 WRITE (IU0,99991) IY, Y(IY)
```

```
  800 WRITE (IU0,99990) LX0, LY0, N0
RETURN

! FORMAT STATEMENTS
99999 FORMAT(1X/23H  ***    LX = 1 OR LESS./)
99998 FORMAT(1X/23H  ***    LY = 1 OR LESS./)
99997 FORMAT(1X/22H  ***    N = 0 OR LESS./)
99996 FORMAT(1X/27H  ***    IDENTICAL X VALUES./)
99995 FORMAT(1X/33H  ***    X VALUES OUT OF SEQUENCE./)
99994 FORMAT(7H    IX =, I6, 10X, 7HX(IX) =, E12.3)
99993 FORMAT(1X/27H  ***    IDENTICAL Y VALUES./)
99992 FORMAT(1X/33H  ***    Y VALUES OUT OF SEQUENCE./)
99991 FORMAT(7H    IY =, I6, 10X, 7HY(IY) =, E12.3)
99990 FORMAT(7H    LX =, I6, 10X, 4HLY =, I6, 10X, 3HN =, I7/ &
       36H ERROR DETECTED IN ROUTINE    ITPLBV)

END SUBROUTINE ITPLBV

FUNCTION MAX(x,y)
      IMPLICIT NONE

      DOUBLE PRECISION x, y, MAX

      IF (x > y) THEN

              MAX = x

      ELSE

              MAX = y

      END IF

END FUNCTION MAX
```

```
PROGRAM THERMO
    IMPLICIT NONE

    ! This program uses the Chance Vought thermodynamic
    ! data to solve thermodynamic values given certain
    ! inputs. The current code will solve the following
    ! problems:
    !
    !   Temperature(pressure, enthalpy)
    !   Temperature(density, enthalpy)
    !   Temperature(pressure, density)
    !   Density(pressure, enthalpy)
    !   Density(pressure, temperature)
    !   Density(enthalpy, temperature)
    !   Enthalpy(pressure, temperature)
    !   Enthalpy(pressure, density)
    !   Pressure(enthalpy, temperature)
    !   Pressure(density, enthalpy)
    !
    ! The code is bound by the limits of the Vought data,
    ! which is roughly speaking 3,000 K to 100,000 K,
    ! 10^-5 atm to 100,000 atm, 1.2251E-6 kg/m^3 to
    ! 122.51 kg/m^3, and 3.7229 MJ/kg to 2895.59 MJ/kg.
    ! See the Chance Vought document or my thesis for
    ! a more detailed range.

    ! Constants
    INTEGER, PARAMETER :: LX = 35   ! TEMP_FIT
    INTEGER, PARAMETER :: LY = 43
    INTEGER, PARAMETER :: NN = 10
    INTEGER, PARAMETER :: DX = 27   ! DENS_FIT
    INTEGER, PARAMETER :: DY = 12
    INTEGER, PARAMETER :: DN = 10

    ! Declare Variables
    INTEGER J, choice
    DOUBLE PRECISION p_atm, p_Pa, enth, h_Jkg, rho, T
    DOUBLE PRECISION var1, var2
    DOUBLE PRECISION XX, YY, ZZ, XXX, YYY, ZZZ

    ! Declare Arrays
    DIMENSION XX(LX), YY(LY), ZZ(LX,LY)
    DIMENSION XXX(DX), YYY(DY), ZZZ(DX,DY)

    ! Vought Data - pressure, enthalpy, temperature
    OPEN(UNIT=10, FILE="thermo1.dat", STATUS='OLD')
    READ(10,*) XX(1:LX)
J1: DO J=1, LY
        READ(10,*) YY(J), ZZ(1:LX,J)
    END DO J1
    CLOSE(10)

    ! Vought Data - enthalpy, pressure, density
    OPEN(UNIT=20, FILE="thermo2.dat", STATUS='OLD')
    READ(20,*) XXX(1:DX)
J2: DO J=1, DY
        READ(20,*) YYY(J), ZZZ(1:DX,J)
    END DO J2
    CLOSE(20)

    ! Reading Input
    OPEN(UNIT=30, FILE='thermo.in', STATUS='OLD')
    READ(30,*) choice
    READ(30,*) var1
    READ(30,*) var2
    CLOSE(30)

! Determine which case has the proper inputs
SWITCH: SELECT CASE(choice)

    ! Inputs: pressure, enthalpy
```

```
CASE(1)

    ! Unit Conversions
    p_atm = var1
    p_Pa = p_atm * 101325.0
    enth = var2
    h_Jkg = enth * 82731.168

    ! Executing curve fit routines
    CALL DENS_FIT(p_Pa, h_Jkg, rho, XXX, YYY, ZZZ, DX, DY, DN)
    CALL TEMP_FIT(h_Jkg, p_Pa, T, XX, YY, ZZ, LX, LY, NN)

! Inputs: pressure, temperature
CASE(2)

    ! Unit Conversions
    p_atm = var1
    p_Pa = p_atm * 101325.0
    T = var2

    ! Executing curve fit routines
    CALL ENTH_FIT(T, p_Pa, h_Jkg, XX, YY, ZZ, LX, LY, NN)
    CALL DENS_FIT(p_Pa, h_Jkg, rho, XXX, YYY, ZZZ, DX, DY, DN)

! Inputs: enthalpy, temperature
CASE(3)

    ! Unit Conversions
    enth = var1
    h_Jkg = enth * 82731.168
    T = var2

    ! Executing curve fit routines
    CALL PRES_FIT2(h_Jkg, T, p_Pa, XX, YY, ZZ, LX, LY, NN)
    CALL DENS_FIT(p_Pa, h_Jkg, rho, XXX, YYY, ZZZ, DX, DY, DN)

! Inputs: enthalpy, density
CASE(4)

    ! Unit Conversions
    enth = var1
    h_Jkg = enth * 82731.168
    rho = var2

    ! Executing curve fit routines
    CALL PRES_FIT(rho, h_Jkg, p_Pa, XXX, YYY, ZZZ, DX, DY, DN)
    CALL TEMP_FIT(h_Jkg, p_Pa, T, XX, YY, ZZ, LX, LY, NN)

! Inputs: pressure, density
CASE(5)

    ! Unit Conversions
    p_atm = var1
    p_Pa = p_atm * 101325.0
    rho = var2

    ! Executing curve fit routines
    CALL ENTH_FIT2(p_Pa, rho, h_Jkg, XXX, YYY, ZZZ, DX, DY, DN)
    CALL TEMP_FIT(h_Jkg, p_Pa, T, XX, YY, ZZ, LX, LY, NN)

! Uknown inputs
CASE DEFAULT

    WRITE(1,*) "Invalid choice"

END SELECT SWITCH

! Output Solution in SI units
WRITE(1,90) "Pressure:    ", p_Pa / 1000000.0, " MPa"
WRITE(1,90) "Enthalpy:    ", h_Jkg / 1000000.0, " MJ/kg"
WRITE(1,90) "Density:     ", rho, " kg/m^3"
```

```
    WRITE(1,90) "Temperature: ", T, " K"
90  FORMAT(A, EN13.3, A)
END PROGRAM THERMO
```

```
!*************************************************************************
! ENTH_FIT2
!    Routine to solve enthalpy from pressure and density using a
!    curve fit of the Vought data. The routine requires an iterative
!    solution of solving pressure from enthalpy and density until
!    the desired pressure is found.
!*************************************************************************

SUBROUTINE ENTH_FIT2(p_Pa, rho, h_Jkg, X, Y, Z, LX, LY, N)
    IMPLICIT NONE

    DOUBLE PRECISION p_Pa, rho, h_Jkg, X, Y, Z
    INTEGER LX, LY, N
    ! p_Pa in Pa
    ! rho in kg/m^3
    ! h_Jkg in J/kg

    ! Constants
    INTEGER, PARAMETER :: IU = 0

    ! Declare Variables
    LOGICAL DONE, FIND
    INTEGER I, LX1, LX2, LY1, LY2
    DOUBLE PRECISION U, V, W, p, h
    DOUBLE PRECISION USTEP, h_min, h_max, temp

    ! Declare Arrays
    DIMENSION X(LX), Y(LY), Z(LX,LY), U(N), V(N), W(N)

    ! Convert Inputs
    p = p_Pa / 101325.0    ! atm

    ! Determine valid range of data from data arrays
    if ((rho >= 1.2251E-6) .AND. (rho < 4.8772E-5)) then

        h_min = 90.0
        h_max = 35000.0
        LX1 = 2
        LY1 = 1
        LX2 = 27
        LY2 = 4

    else if ((rho >= 4.8872E-5) .AND. (rho <= 0.12251)) then

        h_min = 90.0
        h_max = 20000.0
        LX1 = 2
        LY1 = 1
        LX2 = 26
        LY2 = 8

    else if ((rho > 0.12251) .AND. (rho <= 12.251)) then

        h_min = 45.0
        h_max = 10000.0
        LX1 = 1
        LY1 = 7
        LX2 = 25
        LY2 = 11

    else if ((rho > 12.251) .AND. (rho <= 122.51)) then

        h_min = 45.0
        h_max = 5500.0
        LX1 = 1
        LY1 = 8
        LX2 = 24
        LY2 = 12

    else
```

```
        ! Out of range, but try to solve anyway
        write(1,*) "Density out of range at rho = ", rho, " kg/m^3"
        h_min = 90.0
        h_max = 5500.0
        LX1 = 2
        LY1 = 1
        LX2 = 24
        LY2 = 12

    endif

    ! Set range of possible enthalpy
    U(1) = h_min
    U(N) = h_max

    ! Loop until solution found
    DONE = .FALSE.
    do 100 while (DONE .EQV. .FALSE.)

        ! Calculate range steps
        USTEP = (U(N) - U(1))/(N-1.0)
        V(1) = rho
        V(N) = rho
        do 10 I=2,N-1
            U(I) = U(1) + (I-1)*USTEP
            V(I) = rho
10      continue

        ! Get pressure data using curve fit code
        call ITPLBV(IU, LX2-LX1+1, LY2-LY1+1, X(LX1:LX2), Y(LY1:LY2), &
                    Z(LX1:LX2,LY1:LY2), N, U, V, W)

        FIND = .FALSE.
        ! Account for out of range cases
        if (p > W(N)) then
            write(1,*) "OUT OF RANGE AT rho = ", rho, " kg/m^3 AND P = ",p*0.101325, " MP
a"
            EXIT
            U(1) = U(N)
            U(N) = 2 * U(N)
            FIND = .TRUE.
        end if

        if (p < W(1)) then
            write(1,*) "OUT OF RANGE AT rho = ", rho, " kg/m^3 AND P = ",p*0.101325, " MP
a"
            EXIT
            U(N) = U(1)
            U(1) = h_min
            FIND = .TRUE.
        end if

        ! Step through new pressure values, W()
        do 30 I=2,N

            ! Determine if correct value
            temp = abs((P - W(I))/P)
            if (temp < 0.001) then
                h = U(I)
                FIND = .TRUE.
                DONE = .TRUE.
            end if

            ! Determine new range
            if (P <= W(I)) then
                if (FIND .EQV. .FALSE.) then
                    U(N) = U(I)
                    U(1) = U(I-1)
                    FIND = .TRUE.
                end if
            end if
```

```
30      end do

100 end do

    ! Convert output
    h_Jkg = h * 82731.168  ! J/kg

RETURN
END SUBROUTINE ENTH_FIT2
```

```
!*****************************************************************************
! PRES_FIT2
!    Routine to solve pressure from temperature and enthalpy using a
!    curve fit of the Vought data. The routine requires an iterative
!    solution of solving enthalpy from temperature and pressure until
!    the desired enthalpy is found.
!*****************************************************************************

SUBROUTINE PRES_FIT2(h_Jkg, t, p_Pa, X, Y, Z, LX, LY, N)
    IMPLICIT NONE

    DOUBLE PRECISION h_Jkg, t, p_Pa, X, Y, Z
    INTEGER LX, LY, N
    ! h_Jkg in J/kg
    ! t in K
    ! p_Pa in Pa

    ! Constants
    INTEGER, PARAMETER :: IU = 0

    ! Declare Variables
    LOGICAL DONE, FIND
    INTEGER I, LX1, LX2, LY1, LY2
    DOUBLE PRECISION U, V, W, h, p
    DOUBLE PRECISION USTEP, p_min, p_max, temp

    ! Declare Arrays
    DIMENSION X(LX), Y(LY), Z(LX,LY), U(N), V(N), W(N)

    ! Convert Inputs
    h = h_Jkg / 82731.168

    ! Determine valid range of data from data arrays
    if ((T >= 3000.0) .AND. (T < 3600.0)) then

        p_min = 0.00002
        p_max = 1000.0
        LX1 = 1
        LY1 = 1
        LX2 = 24
        LY2 = 6

    else if ((T >= 3600.0) .AND. (T < 4000.0)) then

        p_min = 0.00002
        p_max = 1250.0
        LX1 = 1
        LY1 = 4
        LX2 = 25
        LY2 = 6

    else if ((T >= 4000.0) .AND. (T < 5500.0)) then

        p_min = 0.00005
        p_max = 1250.0
        LX1 = 2
        LY1 = 4
        LX2 = 25
        LY2 = 15

    else if ((T >= 5500.0) .AND. (T < 6500.0)) then

        p_min = 0.00005
        p_max = 2000.0
        LX1 = 2
        LY1 = 12
        LX2 = 26
        LY2 = 15

    else if ((T >= 6500.0) .AND. (T < 7000.0)) then
```

```
    p_min = 0.00005
    p_max = 2500.0
    LX1 = 2
    LY1 = 14
    LX2 = 27
    LY2 = 15

else if ((T >= 7000.0) .AND. (T < 7500.0)) then

    p_min = 0.0001
    p_max = 2500.0
    LX1 = 3
    LY1 = 14
    LX2 = 27
    LY2 = 19

else if ((T >= 7500.0) .AND. (T < 9000.0)) then

    p_min = 0.0001
    p_max = 3000.0
    LX1 = 3
    LY1 = 16
    LX2 = 28
    LY2 = 19

else if ((T >= 9000.0) .AND. (T < 11000.0)) then

    p_min = 0.0002
    p_max = 3000.0
    LX1 = 4
    LY1 = 16
    LX2 = 28
    LY2 = 24

else if ((T >= 11000.0) .AND. (T < 15000.0)) then

    p_min = 0.0002
    p_max = 5000.0
    LX1 = 4
    LY1 = 22
    LX2 = 29
    LY2 = 24

else if ((T >= 15000.0) .AND. (T < 16000.0)) then

    p_min = 0.0005
    p_max = 5000.0
    LX1 = 5
    LY1 = 22
    LX2 = 29
    LY2 = 30

else if ((T >= 16000.0) .AND. (T < 25000.0)) then

    p_min = 0.0005
    p_max = 10000.0
    LX1 = 5
    LY1 = 25
    LX2 = 30
    LY2 = 30

else if ((T >= 25000.0) .AND. (T <= 40000.0)) then

    p_min = 0.001
    p_max = 20000.0
    LX1 = 6
    LY1 = 30
    LX2 = 31
    LY2 = 35

else if ((T > 40000.0) .AND. (T <= 45000.0)) then
```

```
    p_min = 0.002
    p_max = 20000.0
    LX1 = 7
    LY1 = 30
    LX2 = 31
    LY2 = 38

else if ((T > 45000.0) .AND. (T <= 55000.0)) then

    p_min = 0.002
    p_max = 40000.0
    LX1 = 7
    LY1 = 36
    LX2 = 32
    LY2 = 38

else if ((T > 55000.0) .AND. (T <= 60000.0)) then

    p_min = 0.005
    p_max = 40000.0
    LX1 = 8
    LY1 = 36
    LX2 = 32
    LY2 = 43

else if ((T > 60000.0) .AND. (T <= 70000.0)) then

    p_min = 0.005
    p_max = 60000.0
    LX1 = 8
    LY1 = 39
    LX2 = 33
    LY2 = 43

else if ((T > 70000.0) .AND. (T <= 80000.0)) then

    p_min = 0.005
    p_max = 80000.0
    LX1 = 8
    LY1 = 40
    LX2 = 34
    LY2 = 43

else if ((T > 80000.0) .AND. (T <= 100000.0)) then

    p_min = 0.005
    p_max = 100000.0
    LX1 = 8
    LY1 = 41
    LX2 = 35
    LY2 = 43

else

    ! Out of range, but try to solve anyway
    write(1,*) "Temperature out of range at T = ", T, " K"
    p_min = 0.005
    p_max = 1000.0
    LX1 = 8
    LY1 = 1
    LX2 = 24
    LY2 = 43

endif

! Set range of possible pressures
U(1) = p_max
U(N) = p_min

! Loop until solution found
```

```
    DONE = .FALSE.
    do 100 while (DONE .EQV. .FALSE.)

        ! Calculate range steps
        USTEP = (U(N) - U(1))/(N-1.0)
        V(1) = T
        V(N) = T
        do 10 I=2,N-1
            U(I) = U(1) + (I-1)*USTEP
            V(I) = T
10      continue

        ! Get enthalpy data using curve fit code
        call ITPLBV(IU, LX2-LX1+1, LY2-LY1+1, X(LX1:LX2), Y(LY1:LY2), &
                    Z(LX1:LX2,LY1:LY2), N, U, V, W)

        FIND = .FALSE.
        ! Account for out of range cases
        if (h .GT. W(N)) then
            write(1,*) "OUT OF RANGE AT T = ", T, " K AND h = ", h*19.76/238.845896627, "
MJ/kg"
            EXIT
            U(1) = U(N)
            U(N) = 2 * U(N)
            FIND = .TRUE.
        end if

        if (h .LT. W(1)) then
            write(1,*) "OUT OF RANGE AT T = ", T, " K AND h = ", h*19.76/238.845896627, "
 MJ/kg"
            EXIT
            U(N) = U(1)
            U(1) = p_max
            FIND = .TRUE.
        end if

        ! Step through new enthalpy values, W()
        do 30 I=2, N

            ! Determine if correct value
            temp = abs((h - W(I))/h)
            if (temp < 0.001) then
                P = U(I)
                write(*,*) U(I-1), U(I), U(I+1)
                FIND = .TRUE.
                DONE = .TRUE.
            end if

            ! Determine new range
            if (h .LE. W(I)) then
                if (FIND .EQV. .FALSE.) then
                    U(N) = U(I)
                    U(1) = U(I-1)
                    FIND = .TRUE.
                end if
            end if
30      end do

100 end do

    ! Convert output
    p_Pa = p * 101325.0

RETURN
END SUBROUTINE PRES_FIT2
```

```
!***********************************************************************
! ENTH_FIT
!    Routine to solve enthalpy from pressure and temperature using a
!    curve fit of the Vought data. This routine does not require an
!    iterative solution because of the nature of Vought data.
!***********************************************************************

SUBROUTINE ENTH_FIT(t, p_Pa, h_Jkg, X, Y, Z, LX, LY, N)
    IMPLICIT NONE

    DOUBLE PRECISION p_Pa, h_Jkg, t, X, Y, Z
    INTEGER LX, LY, N
    ! p_Pa in Pa
    ! t in K
    ! h_Jkg in J/kg

    ! Constants
    INTEGER, PARAMETER :: IU = 0

    ! Declare variables
    INTEGER LX1, LX2, LY1, LY2
    DOUBLE PRECISION U, V, W, h, p

    ! Declare arrays
    DIMENSION X(LX), Y(LY), Z(LX,LY), U(1), V(1), W(1)

    ! Convert Inputs
    p = p_Pa / 101325.0

    ! Determine valid range of data from data arrays
    if ((p >= 0.00002) .AND. (p < 0.00005)) then

        LX1 = 1
        LY1 = 1
        LX2 = 24
        LY2 = 6

    else if ((p >= 0.00005) .AND. (p < 0.0001)) then

        LX1 = 2
        LY1 = 1
        LX2 = 24
        LY2 = 15

    else if ((p >= 0.0001) .AND. (p < 0.0002)) then

        LX1 = 3
        LY1 = 1
        LX2 = 24
        LY2 = 19

    else if ((p >= 0.0002) .AND. (p < 0.0005)) then

        LX1 = 4
        LY1 = 1
        LX2 = 24
        LY2 = 24

    else if ((p >= 0.0005) .AND. (p < 0.001)) then

        LX1 = 5
        LY1 = 1
        LX2 = 24
        LY2 = 30

    else if ((p >= 0.001) .AND. (p < 0.002)) then

        LX1 = 6
        LY1 = 1
        LX2 = 24
        LY2 = 35
```

```
else if ((p >= 0.002) .AND. (p < 0.005)) then

    LX1 = 7
    LY1 = 1
    LX2 = 24
    LY2 = 38

else if ((p >= 0.005) .AND. (p <= 1000.0)) then

    LX1 = 8
    LY1 = 1
    LX2 = 24
    LY2 = 43

else if ((p > 1000.0) .AND. (p <= 1250.0)) then

    LX1 = 8
    LY1 = 4
    LX2 = 25
    LY2 = 43

else if ((p > 1250.0) .AND. (p <= 2000.0)) then

    LX1 = 8
    LY1 = 12
    LX2 = 26
    LY2 = 43

else if ((p > 2000.0) .AND. (p <= 2500.0)) then

    LX1 = 8
    LY1 = 14
    LX2 = 27
    LY2 = 43

else if ((p > 2500.0) .AND. (p <= 3000.0)) then

    LX1 = 8
    LY1 = 16
    LX2 = 28
    LY2 = 43

else if ((p > 3000.0) .AND. (p <= 5000.0)) then

    LX1 = 8
    LY1 = 22
    LX2 = 29
    LY2 = 43

else if ((p > 5000.0) .AND. (p <= 10000.0)) then

    LX1 = 8
    LY1 = 25
    LX2 = 30
    LY2 = 43

else if ((p > 10000.0) .AND. (p <= 20000.0)) then

    LX1 = 8
    LY1 = 30
    LX2 = 31
    LY2 = 43

else if ((p > 20000.0) .AND. (p <= 40000.0)) then

    LX1 = 8
    LY1 = 36
    LX2 = 32
    LY2 = 43
```

```
    else if ((p > 40000.0) .AND. (p <= 60000.0)) then

        LX1 = 8
        LY1 = 39
        LX2 = 33
        LY2 = 43

    else if ((p > 60000.0) .AND. (p <= 80000.0)) then

        LX1 = 8
        LY1 = 40
        LX2 = 34
        LY2 = 43

    else if ((p > 80000.0) .AND. (p <= 100000.0)) then

        LX1 = 8
        LY1 = 41
        LX2 = 35
        LY2 = 43

    else

        ! Out of pressure range, but try to solve anyway
        write(1,*) "Pressure out of range at P = ", p*0.101325, " MPa"
        LX1 = 8
        LY1 = 1
        LX2 = 24
        LY2 = 43

    endif

    ! Account for out of temperature range cases
    if ((t < Y(LY1)) .OR. (t > Y(LY2))) then

        write(1,*) "Temperature out of range at T = ", t, " K"

    endif

    ! Set pressure and temperature
    U(1) = p
    V(1) = t

    ! Get enthalpy data using curve fit code
    call ITPLBV(IU, LX2-LX1+1, LY2-LY1+1, X(LX1:LX2), Y(LY1:LY2), &
                Z(LX1:LX2,LY1:LY2), 1, U, V, W)

    ! Set enthalpy
    h = W(1)
    N = N

    ! Convert Output
    h_Jkg = h * 82731.168

RETURN
END SUBROUTINE ENTH_FIT
```

```
!*********************************************************************
! PRES_FIT
!    Routine to solve pressure from enthalpy and density using a
!    curve fit of the Vought data. This routine does not require an
!    iterative solution because of the nature of Vought data.
!*********************************************************************

SUBROUTINE PRES_FIT(rho, h_Jkg, p_Pa, X, Y, Z, LX, LY, N)
    IMPLICIT NONE

    DOUBLE PRECISION p_Pa, h_Jkg, rho, X, Y, Z
    INTEGER LX, LY, N
    ! p_Pa in Pa
    ! h_Jkg in J/kg
    ! rho in kg/m^3

    ! Constants
    INTEGER, PARAMETER :: IU = 0

    ! Declare variables
    INTEGER LX1, LX2, LY1, LY2
    DOUBLE PRECISION U, V, W, h, p

    ! Declare arrays
    DIMENSION X(LX), Y(LY), Z(LX,LY), U(1), V(1), W(1)

    ! Convert Inputs
    h = h_Jkg / 82731.168

    ! Determine valid range of data from data arrays
    if ((h >= 45.0) .AND. (h < 90.0)) then

        LX1 = 1
        LY1 = 8
        LX2 = 24
        LY2 = 12

    else if ((h >= 90.0) .AND. (h <= 5500.0)) then

        LX1 = 2
        LY1 = 1
        LX2 = 24
        LY2 = 12

    else if ((h > 5500.0) .AND. (h <= 10000.0)) then

        LX1 = 2
        LY1 = 1
        LX2 = 25
        LY2 = 11

    else if ((h > 10000.0) .AND. (h <= 20000.0)) then

        LX1 = 2
        LY1 = 1
        LX2 = 26
        LY2 = 8

    else if ((h > 20000.0) .AND. (h <= 35000.0)) then

        LX1 = 2
        LY1 = 1
        LX2 = 27
        LY2 = 4

    else

        ! Out of enthalpy range, but try to solve anyway
        write(1,*) "Enthalpy out of range at h = ", h*19.76/238.845896627, " MJ/kg"
        LX1 = 2
        LY1 = 1
```

```
        LX2 = 24
        LY2 = 12

    endif

    ! Account for out of density range case
    if ((rho < Y(LY1)) .OR. (rho > Y(LY2))) then

        write(1,*) "Density out of range at rho = ", rho, " kg/m^3"

    endif

    ! Set enthalpy and density
    U(1) = h
    V(1) = rho

    ! Get pressure data
    call ITPLBV(IU, LX2-LX1+1, LY2-LY1+1, X(LX1:LX2), Y(LY1:LY2), &
                Z(LX1:LX2,LY1:LY2), 1, U, V, W)

    ! Set pressure
    p = W(1)
    N = N

    ! Convert Output
    p_Pa = p * 101325.0

RETURN
END SUBROUTINE PRES_FIT
```

```
PROGRAM ERRCHK
    IMPLICIT NONE

    ! This program chooses a random pressure and a random
    ! density for several points and then uses the Tannehill
    ! curve fits and the new Vought curve fits to solve
    ! the temperature and enthalpy. This is done to allow a
    ! comparison between the two methods. The sampling is
    ! limited by the ranges of both methods' curves.

    ! Constants
    INTEGER, PARAMETER :: LX = 35   ! TEMP_FIT
    INTEGER, PARAMETER :: LY = 43
    INTEGER, PARAMETER :: NN = 10
    INTEGER, PARAMETER :: DX = 27   ! DENS_FIT
    INTEGER, PARAMETER :: DY = 12
    INTEGER, PARAMETER :: DN = 10

    ! Declare Variables
    DOUBLE PRECISION p_atm, rho, T, T_tann, p_Pa, p_MPa
    DOUBLE PRECISION err1, err2, h1_MJkg, h2_MJkg, h_Jkg
    DOUBLE PRECISION XX, YY, ZZ, XXX, YYY, ZZZ
    DOUBLE PRECISION max_err1, max_err2, err1_sum, err2_sum
    DOUBLE PRECISION val
    INTEGER I, J
    INTEGER stats, stats2

    ! Declare Arrays
    DIMENSION stats(6), stats2(6)
    DIMENSION XX(LX), YY(LY), ZZ(LX,LY)
    DIMENSION XXX(DX), YYY(DY), ZZZ(DX,DY)

    ! Loading Vought Data - pressure, enthalpy, temperature
    OPEN(UNIT=10, FILE="thermo1.dat", STATUS='OLD')
    READ(10,*) XX(1:LX)
J1: DO J=1, LY
        READ(10,*) YY(J), ZZ(1:LX,J)
    END DO J1
    CLOSE(10)

    ! Loading Vought Data - enthalpy, pressure, density
    OPEN(UNIT=20, FILE="thermo2.dat", STATUS='OLD')
    READ(20,*) XXX(1:DX)
J2: DO J=1, DY
        READ(20,*) YYY(J), ZZZ(1:DX,J)
    END DO J2
    CLOSE(20)

    ! Prepare random seed
    CALL RANDOM_SEED()

    ! Open output files
    OPEN(UNIT=1, FILE="error.out", STATUS='NEW')
    OPEN(UNIT=2, FILE="h_error0.out", STATUS='NEW')
    OPEN(UNIT=3, FILE="h_error1.out", STATUS='NEW')
    OPEN(UNIT=4, FILE="h_error2.out", STATUS='NEW')
    OPEN(UNIT=7, FILE="h_error3.out", STATUS='NEW')
    OPEN(UNIT=8, FILE="h_error4.out", STATUS='NEW')
    OPEN(UNIT=9, FILE="p_data.out", STATUS='NEW')
    OPEN(UNIT=10, FILE="t_error0.out", STATUS='NEW')
    OPEN(UNIT=11, FILE="t_error1.out", STATUS='NEW')
    OPEN(UNIT=12, FILE="t_error2.out", STATUS='NEW')
    OPEN(UNIT=13, FILE="t_error3.out", STATUS='NEW')
    OPEN(UNIT=14, FILE="t_error4.out", STATUS='NEW')
    OPEN(UNIT=15, FILE="t_data.out", STATUS='NEW')


    ! Initialize variables
    err1_sum = 0.0
    max_err1 = 0.0
    err2_sum = 0.0
```

```
    max_err2 = 0.0
    stats = 0
    stats2 = 0

I = 0
! Loop through 10,000 random points
LOOP:DO 100 WHILE (I .LE. 10000)

        ! 1.2251E-6 < rho < 68.892 kg/m^3
        CALL RANDOM_NUMBER(val)
        rho = 1.2251 * 10.0**(-6.0 + 7.75*val)

        ! 7.5423 < h_MJkg < 454.47 MJ/kg
        CALL RANDOM_NUMBER(val)
        h1_MJkg = 0.0827 * 10**(1.96 + 1.78*val)

        ! Enthalpy Conversion
        h_Jkg = h1_MJkg * 1000000.0

        ! Vought pressure fit
        CALL PRES_FIT(rho, h_Jkg, p_Pa, XXX, YYY, ZZZ, DX, DY, DN)

        ! Pressure Conversion
        p_atm = p_Pa / 101325.0

        ! Insure pressure is within current Vought temperature fit range
        IF ((p_atm >= 0.0001) .AND. (p_atm <= 20000.0)) THEN

            IF ((p_atm < 0.0002) .AND. (h1_MJkg > 98.0)) THEN

                T = 0.0

            ELSE IF ((p_atm < 0.0005) .AND. (h1_MJkg > 167.5)) THEN

                T = 0.0

            ELSE IF ((p_atm < 0.001) .AND. (h1_MJkg > 434.3)) THEN

                T = 0.0

            ELSE

                ! Vought temperature fit
                CALL TEMP_FIT(h_Jkg, p_Pa, T, XX, YY, ZZ, LX, LY, NN)

            END IF

        ELSE

            ! If temperature fit is out of range
            T = 0.0

        END IF

        ! Insure within Tannehill temperature range
        IF ((T >= 3000.0) .AND. (T <= 23000.0)) THEN

            ! Vought enthalpy fit
            CALL ENTH_FIT2(p_Pa, rho, h_Jkg, XXX, YYY, ZZZ, DX, DY, DN)
            h1_MJkg = h_Jkg / 1000000.0

            ! Vought temperature fit
            CALL TEMP_FIT(h_Jkg, p_Pa, T, XX, YY, ZZ, LX, LY, NN)

            ! Tannehill enthalpy and temperature fit
            CALL Tannehill(p_pa, rho, h_Jkg, T_tann)
            h2_MJkg = h_Jkg/1000000.0

            ! Pressure Conversion
            p_MPa = p_Pa / 1000000.0
```

```
! Difference in Vougth and Tannehill enthalpy
err1 = dabs((h2_MJkg - h1_MJkg)/(h2_MJkg))

! Sum for average enthalpy error
err1_sum = err1_sum + err1

! Determine if maximum enthalpy error
IF (err1 > max_err1) max_err1 = err1

! Store enthalpy error data in files
! (pressure in atm, enthalpy in Vought)
IF (err1 <= 0.01) THEN

    stats(1) = stats(1) + 1
    WRITE(2,85) p_MPa, h1_MJkg, h1_MJkg

ELSE IF ((err1 > 0.01) .AND. (err1 <= 0.02)) THEN

    stats(2) = stats(2) + 1
    WRITE(3,85) p_MPa, h1_MJkg, h2_MJkg

ELSE IF ((err1 > 0.02) .AND. (err1 <= 0.03)) THEN

    stats(3) = stats(3) + 1
    WRITE(4,85) p_MPa, h1_MJkg, h2_MJkg

ELSE IF ((err1 > 0.03) .AND. (err1 <= 0.04)) THEN

    stats(4) = stats(4) + 1
    WRITE(7,85) p_MPa, h1_MJkg, h2_MJkg

ELSE IF ((err1 > 0.04) .AND. (err1 <= 0.05)) THEN

    stats(5) = stats(5) + 1
    WRITE(8,85) p_MPa, h1_MJkg, h2_MJkg

ELSE

    stats(6) = stats(6) + 1
    WRITE(8,85) p_MPa, h1_MJkg, h2_MJkg

END IF

! Special output for errors greater than 5%
IF (err1 > 0.05) THEN

    WRITE(1,80) "Error: ", err1*100.0, "%"

    WRITE(1,90) "Pressure:    ", p_MPa, " MPa"
    WRITE(1,90) "Enthalpy:    ", h1_MJkg, " MJ/kg"
    WRITE(1,90) "Tann Enth:   ", h2_MJkg, " MJ/kg"
    WRITE(1,90) "Density:     ", rho, " kg/m^3"
    WRITE(1,90) "Temperature: ", T, " K"
    WRITE(1,*)

END IF

80      FORMAT(A, F6.2, A)
85      FORMAT(F14.7, " ", F9.3, " ", F9.3)

! Difference in Vougth and Tannehill temperature
err2 = dabs((T_tann - T)/(T_tann))

! Sum for average temperature error
err2_sum = err2_sum + err2

! Determine if maximum temperature error
IF (err2 > max_err2) max_err2 = err2

! Store temperature error in files
! (pressure in atm, temperature in K)
```

```
            IF (err2 <= 0.01) THEN

                stats2(1) = stats2(1) + 1
                WRITE(10,85) p_MPa, T, T_tann

            ELSE IF ((err2 > 0.01) .AND. (err2 <= 0.02)) THEN

                stats2(2) = stats2(2) + 1
                WRITE(11,85) p_MPa, T, T_tann

            ELSE IF ((err2 > 0.02) .AND. (err2 <= 0.03)) THEN

                stats2(3) = stats2(3) + 1
                WRITE(12,85) p_MPa, T, T_tann

            ELSE IF ((err2 > 0.03) .AND. (err2 <= 0.04)) THEN

                stats2(4) = stats2(4) + 1
                WRITE(13,85) p_MPa, T, T_tann

            ELSE IF ((err2 > 0.04) .AND. (err2 <= 0.05)) THEN

                stats2(5) = stats2(5) + 1
                WRITE(14,85) p_MPa, T, T_tann

            ELSE

                stats2(6) = stats2(6) + 1
                WRITE(14,85) p_MPa, T, T_tann

            END IF

            ! Special output for errors greater than 5%
            IF (err2 > 0.05) THEN

                WRITE(1,80) "Error: ", err2*100.0, "%"

                WRITE(1,90) "Pressure:    ", p_MPa, " MPa"
                WRITE(1,90) "Enthalpy:    ", h1_MJkg, " MJ/kg"
                WRITE(1,90) "Density:     ", rho, " kg/m^3"
                WRITE(1,90) "Temperature: ", T, " K"
                WRITE(1,90) "Tann Temp:   ", T_tann, " K"
                WRITE(1,*)

            END IF

                ! Output enthalpy and temperature data
                WRITE(9,85) p_MPa, h1_MJkg, h2_MJkg
                WRITE(15,85) p_MPa, T, T_tann

            ! Next random point
            I = I+1

90          FORMAT(A, EN13.3, A)

        END IF

! Next random point, but not counted if not used
100 END DO LOOP

    ! Output enthalpy error statistics
    WRITE(1,*)
    WRITE(1,80) "Avg. Enthalpy Error: ", err1_sum/I*100.0, "%"
    WRITE(1,80) "Max. Enthalpy Error: ", max_err1*100.0, "%"
    WRITE(1,*)
    WRITE(1,*) "Error Breakdown"
    WRITE(1,80) "   0% - 1%: ", DBLE(stats(1))/I*100.0, "%"
    WRITE(1,80) "   1% - 2%: ", DBLE(stats(2))/I*100.0, "%"
    WRITE(1,80) "   2% - 3%: ", DBLE(stats(3))/I*100.0, "%"
    WRITE(1,80) "   3% - 4%: ", DBLE(stats(4))/I*100.0, "%"
    WRITE(1,80) "   4% - 5%: ", DBLE(stats(5))/I*100.0, "%"
```

```
    WRITE(1,80) "   5% +   : ", DBLE(stats(6))/I*100.0, "%"

    ! Output temperature error statistics
    WRITE(1,*)
    WRITE(1,80) "Avg. Temperature Error: ", err2_sum/I*100.0, "%"
    WRITE(1,80) "Max. Temperature Error: ", max_err2*100.0, "%"
    WRITE(1,*)
    WRITE(1,*) "Error Breakdown"
    WRITE(1,80) "   0% - 1%: ", DBLE(stats2(1))/I*100.0, "%"
    WRITE(1,80) "   1% - 2%: ", DBLE(stats2(2))/I*100.0, "%"
    WRITE(1,80) "   2% - 3%: ", DBLE(stats2(3))/I*100.0, "%"
    WRITE(1,80) "   3% - 4%: ", DBLE(stats2(4))/I*100.0, "%"
    WRITE(1,80) "   4% - 5%: ", DBLE(stats2(5))/I*100.0, "%"
    WRITE(1,80) "   5% +   : ", DBLE(stats2(6))/I*100.0, "%"

    CLOSE(1);
    CLOSE(2);
    CLOSE(3);
    CLOSE(4);
    CLOSE(7);
    CLOSE(8);
    CLOSE(9);
    CLOSE(10);
    CLOSE(11);
    CLOSE(12);
    CLOSE(13);
    CLOSE(14);
    CLOSE(15);

END PROGRAM ERRCHK


!**********************************************************************
!   Tannehill Curve Fits
!      Performs Tannehill's enthalpy and temperature curve fits
!      given pressure and density. From "Simplified Curve Fits
!      for the Thermodynamic Properties of Equilibrium Air", 1987.
!**********************************************************************

SUBROUTINE Tannehill(P, rho, Enthalpy, Temp)
    IMPLICIT NONE

    DOUBLE PRECISION P, rho, Enthalpy, Temp
    ! P in pascals
    ! rho in kg/m^3
    ! Enthalpy in J/kg
    ! Temp in K

    ! Declare variables
    INTEGER sign
    DOUBLE PRECISION X, Y, Z, gamma
    DOUBLE PRECISION rho_o, P_o, T_o, R
    DOUBLE PRECISION c1,c2,c3,c4,c5,c6,c7,c8,c9,c10,c11,c12,c13,c14,c15
    DOUBLE PRECISION c16,c17,c18,c19,c20,c21,c22,c23,c24
    DOUBLE PRECISION d1,d2,d3,d4,d5,d6,d7,d8,d9,d10,d11,d12,d13,d14,d15
    DOUBLE PRECISION d16,d17,d18,d19,d20,d21,d22,d23,d24

!   Density:    1.2925E-7 < rho < 1292.5   kg/m^3
!   Pressure:   127560    <  P  < 10132500 Pa

    PARAMETER(rho_o = 1.292)
    PARAMETER(P_o = 101330.0)
    PARAMETER(T_o = 273.15)
    PARAMETER(R = 287.06)

    X = LOG10(P/P_o)
    Y = LOG10(rho/rho_o)
    Z = X - Y

    IF ((Y >= -7.0) .AND. (Y <= -4.5)) THEN
```

```
! Tannehill Enthalpy Curve Coefficients
IF (Z <= 0.1) THEN

    c1  = 1.3986
    c2  = 0.0
    c3  = 0.0
    c4  = 0.0
    c5  = 0.0
    c6  = 0.0
    c7  = 0.0
    c8  = 0.0
    c9  = 0.0
    c10 = 0.0
    c11 = 0.0
    c12 = 0.0
    c13 = 0.0
    c14 = 0.0
    c15 = 0.0
    c16 = 0.0
    c17 = 0.0
    c18 = 0.0
    c19 = 0.0
    c20 = 0.0
    c21 = 0.0
    c22 = 0.0
    c23 = 0.0
    c24 = 0.0
    sign = 1.0

ELSE IF ((Z > 0.10) .AND. (Z <= 0.85)) THEN

    c1  =  253.908
    c2  =  101.491
    c3  = -387.199
    c4  = -154.304
    c5  =  7.28532
    c6  =  98.6233
    c7  = -8.04378
    c8  =  46.3763
    c9  = -1.82577E-3
    c10 =  21.8994
    c11 = -252.423
    c12 = -101.445
    c13 =  387.210
    c14 =  154.298
    c15 = -7.2773
    c16 = -98.7576
    c17 =  8.04277
    c18 = -46.3883
    c19 =  2.28399E-3
    c20 = -21.9438
    c21 = -11.0
    c22 =  2.0
    c23 =  11.0
    c24 = -2.0
    sign = -1.0

ELSE IF ((Z > 0.85) .AND. (Z <= 1.3)) THEN

    c1  = -10.5745
    c2  = -1.93693
    c3  =  30.7202
    c4  =  3.35578
    c5  = -7.79965E-2
    c6  = -26.0637
    c7  =  6.68790E-2
    c8  = -1.42391
    c9  = -9.86882E-4
    c10 =  7.23223
    c11 = -18.6342
    c12 =  2.41997E-2
```

```
      c13 =  32.0880
      c14 = -0.746914
      c15 =  3.75161E-2
      c16 = -16.9985
      c17 = -4.10125E-2
      c18 =  0.539041
      c19 =  5.74637E-4
      c20 =  2.56253
      c21 =  276.8567
      c22 =  21.52383
      c23 = -216.4837
      c24 = -13.94837
      sign = 1.0

ELSE IF ((Z > 1.3) .AND. (Z <= 1.95)) THEN

      c1  =  0.617584
      c2  = -0.240690
      c3  =  1.95904
      c4  =  0.341644
      c5  = -1.01073E-2
      c6  = -1.68951
      c7  =  6.77631E-3
      c8  = -0.110932
      c9  = -1.15922E-4
      c10 =  0.426058
      c11 = -13.4222
      c12 = -0.543713
      c13 =  18.1528
      c14 =  0.395928
      c15 = -7.41105E-3
      c16 = -7.97425
      c17 =  1.67768E-3
      c18 = -5.80593E-2
      c19 = -3.32714E-6
      c20 =  1.12448
      c21 =  86.77803
      c22 = -8.370349
      c23 = -40.74084
      c24 =  7.407405
      sign = 1.0

ELSE IF (Z > 1.95) THEN

      c1  = -8.32595
      c2  = -0.350219
      c3  =  13.6455
      c4  =  0.359350
      c5  = -3.70109E-3
      c6  = -6.49007
      c7  =  3.30836E-3
      c8  = -8.38594E-2
      c9  =  1.10018E-4
      c10 =  1.02443
      c11 = -30.8441
      c12 = -1.49510
      c13 =  30.0585
      c14 =  0.919650
      c15 = -3.60024E-2
      c16 = -9.33522
      c17 =  1.02522E-2
      c18 = -0.135228
      c19 = -4.68760E-4
      c20 =  0.892634
      c21 =  88.00047
      c22 = -16.79356
      c23 = -33.33353
      c24 =  8.465574
      sign = 1.0

END IF
```

```
! Tannehill Temperature Curve Coefficients
IF ((Z > 0.25) .AND. (Z <= 0.95)) THEN

    d1  =  0.123718
    d2  =  1.08623E-2
    d3  =  0.224239
    d4  = -8.24608E-2
    d5  = -1.17615E-3
    d6  =  1.18397
    d7  = -1.87566E-3
    d8  =  6.4852E-2
    d9  = -1.19155E-4
    d10 = -0.552634
    d11 =  0.0
    d12 =  0.0
    d13 =  0.0
    d14 =  0.0
    d15 =  0.0
    d16 =  0.0
    d17 =  0.0
    d18 =  0.0
    d19 =  0.0
    d20 =  0.0
    d21 =  0.0
    d22 =  0.0
    d23 =  0.0
    d24 =  0.0

ELSE IF ((Z > 0.95) .AND. (Z <= 1.40)) THEN

    d1  = -8.12952
    d2  = -0.828637
    d3  =  22.6904
    d4  =  1.41132
    d5  = -2.98633E-2
    d6  = -19.1806
    d7  =  2.70066E-2
    d8  = -0.578875
    d9  = -2.28103E-4
    d10 =  5.62580
    d11 = -3.99845
    d12 =  0.226369
    d13 =  2.52870
    d14 = -0.728448
    d15 =  1.09769E-2
    d16 =  2.99238
    d17 = -1.83819E-2
    d18 =  0.391440
    d19 = -1.51380E-4
    d20 = -2.04463
    d21 = -38.87015
    d22 = -29.08228
    d23 =  40.70557
    d24 =  26.82347

ELSE IF ((Z > 1.40) .AND. (Z <= 1.95)) THEN

    d1  = -19.8573
    d2  = -1.67225
    d3  =  37.6159
    d4  =  2.10964
    d5  = -3.40174E-2
    d6  = -22.2215
    d7  =  2.31712E-2
    d8  = -0.644596
    d9  = -9.80275E-5
    d10 =  4.40486
    d11 = -5.36809
    d12 =  0.241201
    d13 = -1.25881
```

```
        d14 = -0.862744
        d15 = -3.79774E-3
        d16 =  5.58609
        d17 = -7.81335E-3
        d18 =  0.378963
        d19 = -3.80005E-4
        d20 = -1.81566
        d21 =  20.8
        d22 = -25.6
        d23 =  1.0
        d24 =  18.0

    ELSE IF (Z > 1.95) THEN

        d1  = -23.3271
        d2  = -1.89958
        d3  =  32.1440
        d4  =  1.68622
        d5  = -4.42123E-2
        d6  = -13.8645
        d7  =  2.82629E-2
        d8  = -0.340976
        d9  =  6.63272E-4
        d10 =  2.04466
        d11 =  8.35474
        d12 =  1.71347
        d13 = -16.0715
        d14 = -1.63139
        d15 =  4.14641E-2
        d16 =  8.70275
        d17 = -2.30068E-2
        d18 =  0.360966
        d19 =  1.53246E-5
        d20 = -1.46166
        d21 =  111.5884
        d22 = -6.452606
        d23 = -53.37863
        d24 =  2.026986

    END IF

ELSE IF ((Y > -4.5) .AND. (Y <= -0.5)) THEN

    ! Tannehill Enthalpy Curve Coefficients
    IF (Z <= 0.1) THEN

        c1  = 1.399
        c2  = 0.0
        c3  = 0.0
        c4  = 0.0
        c5  = 0.0
        c6  = 0.0
        c7  = 0.0
        c8  = 0.0
        c9  = 0.0
        c10 = 0.0
        c11 = 0.0
        c12 = 0.0
        c13 = 0.0
        c14 = 0.0
        c15 = 0.0
        c16 = 0.0
        c17 = 0.0
        c18 = 0.0
        c19 = 0.0
        c20 = 0.0
        c21 = 0.0
        c22 = 0.0
        c23 = 0.0
        c24 = 0.0
        sign = 1.0
```

```
ELSE IF ((Z > 0.10) .AND. (Z <= 0.95)) THEN

    c1  = -133.083
    c2  = -9.98707
    c3  =  394.734
    c4  =  23.5810
    c5  =  1.43957
    c6  = -384.712
    c7  = -1.43175
    c8  = -13.6367
    c9  =  1.77068E-5
    c10 =  124.325
    c11 =  134.486
    c12 =  9.99122
    c13 = -394.719
    c14 = -23.5853
    c15 = -1.43799
    c16 =  384.616
    c17 =  1.43039
    c18 =  13.6318
    c19 =  1.44367E-4
    c20 = -124.348
    c21 = -21.41444
    c22 =  1.381584
    c23 =  20.39473
    c24 = -1.315789
    sign = -1.0

ELSE IF ((Z > 0.95) .AND. (Z <= 1.5)) THEN

    c1  = -7.36684
    c2  = -1.13247
    c3  =  24.7879
    c4  =  1.99625
    c5  = -4.91630E-2
    c6  = -23.299
    c7  =  4.16673E-2
    c8  = -0.859418
    c9  = -6.58149E-4
    c10 =  7.19016
    c11 = -2.42647
    c12 =  0.557912
    c13 = -2.03055
    c14 = -1.22031
    c15 =  3.74866E-2
    c16 =  7.75414
    c17 = -3.39278E-2
    c18 =  0.608488
    c19 =  5.21042E-4
    c20 = -3.68326
    c21 =  80.77385
    c22 = -12.73807
    c23 = -65.47623
    c24 =  11.90475
    sign = 1.0

ELSE IF ((Z > 1.5) .AND. (Z <= 2.0)) THEN

    c1  =  0.431520
    c2  = -0.283857
    c3  =  2.27791
    c4  =  0.399159
    c5  = -1.29444E-2
    c6  = -1.84314
    c7  =  8.78724E-3
    c8  = -0.128136
    c9  = -1.60583E-4
    c10 =  0.445362
    c11 = -10.3883
    c12 = -0.358718
```

```
    c13 =  13.5068
    c14 =  0.187268
    c15 = -4.28184E-3
    c16 = -5.63894
    c17 = -9.52016E-4
    c18 = -1.45625E-3
    c19 = -4.10506E-5
    c20 =  0.739915
    c21 =  294.9221
    c22 =  13.6866
    c23 = -155.9335
    c24 = -3.78776
    sign = 1.0

ELSE IF (Z > 2.0) THEN

    c1  = -3.77766
    c2  = -0.553738
    c3  =  6.60834
    c4  =  0.487181
    c5  = -2.11045E-2
    c6  = -2.94754
    c7  =  9.67277E-3
    c8  = -0.102365
    c9  = -2.1942E-4
    c10 =  0.43962
    c11 =  40.5813
    c12 =  3.25692
    c13 = -47.9583
    c14 = -2.53660
    c15 =  9.06436E-2
    c16 =  18.904
    c17 = -3.47578E-2
    c18 =  0.494114
    c19 =  1.00077E-3
    c20 = -2.48554
    c21 =  534.718
    c22 =  74.95657
    c23 = -221.9822
    c24 = -30.17229
    sign = 1.0

END IF

! Tannehill Temperature Curve Fit Coefficients
IF ((Z > 0.25) .AND. (Z <= 0.95)) THEN

    d1  =  2.03910E-2
    d2  =  7.67310E-3
    d3  =  0.848581
    d4  = -2.93086E-2
    d5  =  8.40269E-4
    d6  =  0.267251
    d7  = -1.47701E-3
    d8  =  2.37262E-2
    d9  =  3.13687E-5
    d10 = -0.141973
    d11 =  0.0
    d12 =  0.0
    d13 =  0.0
    d14 =  0.0
    d15 =  0.0
    d16 =  0.0
    d17 =  0.0
    d18 =  0.0
    d19 =  0.0
    d20 =  0.0
    d21 =  0.0
    d22 =  0.0
    d23 =  0.0
    d24 =  0.0
```

```
ELSE IF ((Z > 0.95) .AND. (Z <= 1.45)) THEN

    d1  = -5.12404
    d2  = -0.28474
    d3  =  15.4532
    d4  =  0.452475
    d5  = -1.22881E-2
    d6  = -13.5181
    d7  =  8.56845E-3
    d8  = -0.168725
    d9  = -3.25256E-4
    d10 =  4.18451
    d11 =  7.52564
    d12 =  0.835238
    d13 = -19.5558
    d14 = -1.23393
    d15 =  3.34510E-2
    d16 =  17.1779
    d17 = -2.34269E-2
    d18 =  0.454628
    d19 =  4.81788E-4
    d20 = -5.09936
    d21 =  61.48442
    d22 = -18.28123
    d23 = -54.68755
    d24 =  15.62500

ELSE IF ((Z > 1.45) .AND. (Z <= 2.05)) THEN

    d1  = -12.3779
    d2  = -1.14728
    d3  =  24.1382
    d4  =  1.38957
    d5  = -3.63693E-2
    d6  = -14.2844
    d7  =  2.24265E-2
    d8  = -0.406553
    d9  = -3.23888E-4
    d10 =  2.8762
    d11 =  4.40782
    d12 =  1.33046
    d13 = -11.5405
    d14 = -1.59892
    d15 =  5.30580E-2
    d16 =  8.57309
    d17 = -3.10376E-2
    d18 =  0.471274
    d19 =  4.77650E-4
    d20 = -1.96233
    d21 =  140.75
    d22 = -6.499992
    d23 = -77.5
    d24 =  5.0

ELSE IF (Z > 2.05) THEN

    d1  = -12.7244
    d2  = -1.66684
    d3  =  17.2708
    d4  =  1.45307
    d5  = -3.64515E-2
    d6  = -6.97208
    d7  =  1.90463E-2
    d8  = -0.304323
    d9  =  4.80787E-4
    d10 =  0.967524
    d11 =  7.71330
    d12 =  0.50834
    d13 = -9.8211
    d14 = -0.449138
```

```
        d15 = -9.41787E-4
        d16 =  4.1653
        d17 = -2.40293E-3
        d18 =  9.63923E-2
        d19 = -8.28450E-4
        d20 = -0.588807
        d21 = -1092.654
        d22 = -305.312
        d23 =  465.6243
        d24 =  131.2498

    END IF

ELSE IF ((Y > -0.5) .AND. (Y <= 3.0)) THEN

    ! Tannehill Enthalpy Curve Coefficients
    IF (Z <= 0.1) THEN

        c1  = 1.4017
        c2  = 0.0
        c3  = 0.0
        c4  = 0.0
        c5  = 0.0
        c6  = 0.0
        c7  = 0.0
        c8  = 0.0
        c9  = 0.0
        c10 = 0.0
        c11 = 0.0
        c12 = 0.0
        c13 = 0.0
        c14 = 0.0
        c15 = 0.0
        c16 = 0.0
        c17 = 0.0
        c18 = 0.0
        c19 = 0.0
        c20 = 0.0
        c21 = 0.0
        c22 = 0.0
        c23 = 0.0
        c24 = 0.0
        sign = 1.0

    ELSE IF ((Z > 0.10) .AND. (Z <= 1.05)) THEN

        c1  = -96.7488
        c2  =  0.205296
        c3  =  269.927
        c4  = -1.92887
        c5  =  0.378392
        c6  = -246.711
        c7  = -0.324965
        c8  =  1.54416
        c9  = -3.61036E-3
        c10 =  74.8760
        c11 =  98.1502
        c12 = -0.205448
        c13 = -269.913
        c14 =  1.93052
        c15 = -0.378527
        c16 =  246.630
        c17 =  0.324832
        c18 = -1.54646
        c19 =  3.66182E-3
        c20 = -74.898
        c21 = -26.59865
        c22 =  1.564631
        c23 =  23.12926
        c24 = -1.360543
        sign = -1.0
```

```
ELSE IF ((Z > 1.05) .AND. (Z <= 1.6)) THEN

    c1  = -0.267593
    c2  = -0.187457
    c3  =  5.07693
    c4  =  0.272286
    c5  =  1.04541E-2
    c6  = -5.0852
    c7  = -1.42211E-2
    c8  = -7.81935E-2
    c9  =  6.38962E-4
    c10 =  1.58711
    c11 =  2.87969
    c12 =  0.39009
    c13 = -8.06179
    c14 = -0.55125
    c15 = -1.01903E-2
    c16 =  7.29592
    c17 =  1.35906E-2
    c18 =  0.183861
    c19 = -8.97772E-4
    c20 = -2.15153
    c21 =  182.8573
    c22 = -34.28596
    c23 = -151.786
    c24 =  29.76212
    sign = 1.0

ELSE IF (Z > 1.6) THEN

    c1  =  0.921537
    c2  = -0.239670
    c3  =  1.30714
    c4  =  0.34299
    c5  = -2.18847E-2
    c6  = -1.20916
    c7  =  1.36691E-2
    c8  = -0.110206
    c9  = -4.90274E-4
    c10 =  0.3087920
    c11 = -6.77089
    c12 = -6.90476E-2
    c13 =  8.18168
    c14 = -9.52708E-2
    c15 =  2.98487E-2
    c16 = -3.07662
    c17 = -1.78706E-2
    c18 =  6.60408E-2
    c19 =  6.28419E-4
    c20 =  0.338590
    c21 =  159.16669
    c22 =  39.76192
    c23 = -79.66199
    c24 = -16.6667
    sign = 1.0

END IF

! Tannehill Temperature Curve Fit Coefficients
IF ((Z > 0.25) .AND. (Z <= 1.00)) THEN

    d1  = -1.54141E-3
    d2  =  6.58337E-4
    d3  =  0.982201
    d4  = -3.85028E-3
    d5  =  1.23111E-4
    d6  =  3.77441E-2
    d7  = -4.08210E-4
    d8  =  4.56963E-3
    d9  =  2.13592E-5
```

```
      d10 = -2.35172E-2
      d11 =  0.0
      d12 =  0.0
      d13 =  0.0
      d14 =  0.0
      d15 =  0.0
      d16 =  0.0
      d17 =  0.0
      d18 =  0.0
      d19 =  0.0
      d20 =  0.0
      d21 =  0.0
      d22 =  0.0
      d23 =  0.0
      d24 =  0.0

   ELSE IF ((Z > 1.00) .AND. (Z <= 1.45)) THEN

      d1  =  0.806492
      d2  =  9.91293E-2
      d3  = -1.70742
      d4  = -0.228264
      d5  =  5.03500E-3
      d6  =  3.02351
      d7  = -6.13927E-3
      d8  =  0.131574
      d9  =  1.69824E-4
      d10 = -1.12755
      d11 = -0.117930
      d12 = -0.212207
      d13 =  1.36524
      d14 =  0.405886
      d15 = -1.88260E-2
      d16 = -2.10926
      d17 =  1.65486E-2
      d18 = -0.189881
      d19 = -5.1140E-4
      d20 =  0.879806
      d21 =  195.9604
      d22 = -42.69391
      d23 = -173.4931
      d24 =  37.62898

   ELSE IF (Z > 1.45) THEN

      d1  = -1.66249
      d2  = -8.91113E-2
      d3  =  4.11648
      d4  =  8.78093E-2
      d5  = -3.09742E-3
      d6  = -1.84445
      d7  =  1.99879E-3
      d8  = -7.50324E-3
      d9  =  6.85472E-5
      d10 =  0.305784
      d11 =  11.1555
      d12 =  1.3210
      d13 = -17.1236
      d14 = -1.2919
      d15 =  6.28124E-2
      d16 =  8.63804
      d17 = -3.07949E-2
      d18 =  0.307809
      d19 =  1.57743E-3
      d20 = -1.42634
      d21 =  133.0611
      d22 =  8.979635
      d23 = -72.65298
      d24 = -2.449009

   END IF
```

```
    END IF

    ! Tannehill Enthalpy Equation
    gamma = c1 + c2*Y + c3*Z + c4*Y*Z + c5*Y*Y + c6*Z*Z + c7*Y*Y*Z + &
          & c8*Y*Z*Z + c9*Y*Y*Y + c10*Z*Z*Z + (c11 + c12*Y + c13*Z + &
          & c14*Y*Z + c15*Y*Y + c16*Z*Z + c17*Y*Y*Z + c18*Y*Z*Z +    &
          & c19*Y*Y*Y + c20*Z*Z*Z) / (1.0 + sign*DEXP(c21 + c22*Y +  &
          & c23*Z + c24*Y*Z))

    Enthalpy = (P/rho) * (gamma/(gamma-1.0))

    IF (Z <= 0.25) THEN

        Temp = P/(rho * R)

    ELSE

        ! Tannehill Temperature Equation
        gamma = d1 + d2*Y + d3*Z + d4*Y*Z + d5*Y*Y + d6*Z*Z + d7*Y*Y*Z + &
              & d8*Y*Z*Z + d9*Y*Y*Y + d10*Z*Z*Z + (d11 + d12*Y + d13*Z + &
              & d14*Y*Z + d15*Y*Y + d16*Z*Z + d17*Y*Y*Z + d18*Y*Z*Z +    &
              & d19*Y*Y*Y + d20*Z*Z*Z) / (1.0 + DEXP(d21 + d22*Y +  &
              & d23*Z + d24*Y*Z))

        Temp = (10.0**gamma) * T_o

    END IF


END SUBROUTINE Tannehill
```

```
PROGRAM ERRCHK2
    IMPLICIT NONE

    ! This program chooses a random pressure and a random
    ! enthalpy for several points and then uses the Fought
    ! curve fits and the new Vought curve fits to solve
    ! the temperature. This is done to allow a comparison
    ! between the two methods. The sampling is limited by
    ! the ranges of both methods' curves.

    ! Constants
    INTEGER, PARAMETER :: LX = 35   ! TEMP_FIT
    INTEGER, PARAMETER :: LY = 43
    INTEGER, PARAMETER :: NN = 10
    INTEGER, PARAMETER :: DX = 27   ! DENS_FIT
    INTEGER, PARAMETER :: DY = 12
    INTEGER, PARAMETER :: DN = 10

    ! Declare Variables
    DOUBLE PRECISION p_atm, rho, T, T_fought, p_Pa, p_MPa
    DOUBLE PRECISION err2, h_MJkg, h_Jkg
    DOUBLE PRECISION XX, YY, ZZ, XXX, YYY, ZZZ
    DOUBLE PRECISION max_err2, err2_sum
    DOUBLE PRECISION val
    INTEGER I, J
    INTEGER stats2

    ! Declare Arrays
    DIMENSION stats2(6)
    DIMENSION XX(LX), YY(LY), ZZ(LX,LY)
    DIMENSION XXX(DX), YYY(DY), ZZZ(DX,DY)

    ! Loading Vought Data - pressure, enthalpy, temperature
    OPEN(UNIT=10, FILE="thermo1.dat", STATUS='OLD')
    READ(10,*) XX(1:LX)
J1: DO J=1, LY
        READ(10,*) YY(J), ZZ(1:LX,J)
    END DO J1
    CLOSE(10)

    ! Loading Vought Data - enthalpy, pressure, density
    OPEN(UNIT=20, FILE="thermo2.dat", STATUS='OLD')
    READ(20,*) XXX(1:DX)
J2: DO J=1, DY
        READ(20,*) YYY(J), ZZZ(1:DX,J)
    END DO J2
    CLOSE(20)

    ! Prepare random seed
    CALL RANDOM_SEED()

    ! Open output files
    OPEN(UNIT=1, FILE="error2.out", STATUS='NEW')
    OPEN(UNIT=10, FILE="t2_error0.out", STATUS='NEW')
    OPEN(UNIT=11, FILE="t2_error1.out", STATUS='NEW')
    OPEN(UNIT=12, FILE="t2_error2.out", STATUS='NEW')
    OPEN(UNIT=13, FILE="t2_error3.out", STATUS='NEW')
    OPEN(UNIT=14, FILE="t2_error4.out", STATUS='NEW')
    OPEN(UNIT=15, FILE="t2_data.out", STATUS='NEW')


    ! Initialize variables
    err2_sum = 0.0
    max_err2 = 0.0
    stats2 = 0

I = 0
! Loop through 10,000 random points
LOOP:DO 100 WHILE (I .LE. 10000)

        ! 1.2251E-6 < rho < 68.892 kg/m^3
```

```
          CALL RANDOM_NUMBER(val)
          rho = 1.2251 * 10.0**(-6.0 + 7.75*val)

          ! 7.5423 < h_MJkg < 454.47 MJ/kg
          CALL RANDOM_NUMBER(val)
          h_MJkg = 0.0827 * 10**(1.96 + 1.78*val)

          ! enthlpy conversions
          h_Jkg = h_MJkg * 1000000.0

          ! Vought pressure fit
          CALL PRES_FIT(rho, h_Jkg, p_Pa, XXX, YYY, ZZZ, DX, DY, DN)

          ! Pressure conversion
          p_atm = p_Pa / 101325.0

          ! Insure pressure is within current Vought and Fought pressure ranges
          IF ((p_atm >= 0.0001) .AND. (p_atm <= 1000.0)) THEN

              IF ((p_atm < 0.0002) .AND. (h_MJkg > 98.0)) THEN

                  T = 0.0

              ELSE IF ((p_atm < 0.0005) .AND. (h_MJkg > 167.5)) THEN

                  T = 0.0

              ELSE IF ((p_atm < 0.001) .AND. (h_MJkg > 434.3)) THEN

                  T = 0.0

              ELSE

                  ! Vought temperature fit
                  CALL TEMP_FIT(h_Jkg, p_Pa, T, XX, YY, ZZ, LX, LY, NN)

              END IF

          ELSE

              ! If temperature fit is out of range
              T = 0.0

          END IF

          ! Insure within Fought temperature range
          IF ((T >= 3000.0) .AND. (T <= 29000.0)) THEN

              ! Fought temperature fit
              CALL TEMPNEW(h_Jkg, p_Pa, T_fought)

              ! Pressure conversion
              p_MPa = p_Pa / 1000000.0

80            FORMAT(A, F6.2, A)
85            FORMAT(F14.7, " ", F7.0, " ", F7.0)

              ! Difference in Vougth and Fought temperature
              err2 = dabs((T_fought - T)/(T_fought))

              ! Sum for average temperature error
              err2_sum = err2_sum + err2

              ! Determine if maximum temperature error
              IF (err2 > max_err2) max_err2 = err2

              ! Store temperature error in files
              ! (pressure in atm, temperature in K)
              IF (err2 <= 0.01) THEN

                  stats2(1) = stats2(1) + 1
```

```
            WRITE(10,85) p_MPa, T, T_fought

        ELSE IF ((err2 > 0.01) .AND. (err2 <= 0.02)) THEN

            stats2(2) = stats2(2) + 1
            WRITE(11,85) p_MPa, T, T_fought

        ELSE IF ((err2 > 0.02) .AND. (err2 <= 0.03)) THEN

            stats2(3) = stats2(3) + 1
            WRITE(12,85) p_MPa, T, T_fought

        ELSE IF ((err2 > 0.03) .AND. (err2 <= 0.04)) THEN

            stats2(4) = stats2(4) + 1
            WRITE(13,85) p_MPa, T, T_fought

        ELSE IF ((err2 > 0.04) .AND. (err2 <= 0.05)) THEN

            stats2(5) = stats2(5) + 1
            WRITE(14,85) p_MPa, T, T_fought

        ELSE

            stats2(6) = stats2(6) + 1
            WRITE(14,85) p_MPa, T, T_fought

        END IF

        ! Special output for errors greater than 5%
        IF (err2 > 0.05) THEN

            WRITE(1,80) "Error: ", err2*100.0, "%"

            WRITE(1,90) "Pressure:    ", p_MPa, " MPa"
            WRITE(1,90) "Enthalpy:    ", h_MJkg, " MJ/kg"
            WRITE(1,90) "Density:     ", rho, " kg/m^3"
            WRITE(1,90) "Temperature: ", T, " K"
            WRITE(1,90) "Fought Temp: ", T_fought, " K"
            WRITE(1,*)

        END IF

            ! Output temperature data
            WRITE(15,85) p_MPa, T, T_fought

        ! Next random point
        I = I+1

90      FORMAT(A, EN13.3, A)

    END IF

! Next random point, but not counted if not used
100 END DO LOOP

    ! Output temperature error statistics
    WRITE(1,*)
    WRITE(1,80) "Avg. Temperature Error: ", err2_sum/I*100.0, "%"
    WRITE(1,80) "Max. Temperature Error: ", max_err2*100.0, "%"
    WRITE(1,*)
    WRITE(1,*) "Error Breakdown"
    WRITE(1,80) "   0% - 1%: ", DBLE(stats2(1))/I*100.0, "%"
    WRITE(1,80) "   1% - 2%: ", DBLE(stats2(2))/I*100.0, "%"
    WRITE(1,80) "   2% - 3%: ", DBLE(stats2(3))/I*100.0, "%"
    WRITE(1,80) "   3% - 4%: ", DBLE(stats2(4))/I*100.0, "%"
    WRITE(1,80) "   4% - 5%: ", DBLE(stats2(5))/I*100.0, "%"
    WRITE(1,80) "   5% +   : ", DBLE(stats2(6))/I*100.0, "%"

    CLOSE(1);
    CLOSE(10);
```

```
    CLOSE(11);
    CLOSE(12);
    CLOSE(13);
    CLOSE(14);
    CLOSE(15);

END PROGRAM ERRCHK2
```

VITA

Chris Thames was born in Raleigh, North Carolina on April 1, 1969. He attended public school in Raleigh and graduated from Millbrook Senior High School in June of 1987. He had a strong interest in space from early childhood. Since his strengths were in math and science, he quickly decided on pursuing a career in Aerospace Engineering when looking at possible careers in high school. In May of 1994, he received his Bachelor of Science in Aerospace Engineering and a Minor in Computer Programming from North Carolina State University. After spending a year looking at graduate schools and working at Data General Corporation, he entered University of Tennessee, Knoxville to pursue a Master of Science in Aerospace Engineering. With completion of this thesis, he will graduate with this degree and a Master's Minor in Computer Science in August of 1997. Chris will be beginning a programming job with Lockheed Martin in the Shuttle Flight Software Group immediately after graduating.