8-1-2022

Enhancements to Nuclear Thermal Propulsion Rockets

Kimberly Gonzalez

ENHANCEMENTS TO NUCLEAR THERMAL PROPULSION ROCKETS

By

Kimberly Gonzalez

Bachelor of Science – Mechanical Engineering
University of Nevada, Las Vegas
2018

A dissertation submitted in partial fulfillment
of the requirements for the

Doctor of Philosophy – Mechanical Engineering

Department of Mechanical Engineering
Howard R. Hughes College of Engineering
Graduate College

University of Nevada, Las Vegas
August 2022

Dissertation Approval

The Graduate College
The University of Nevada, Las Vegas

June 24, 2022

This dissertation prepared by

Kimberly Gonzalez

entitled

Enhancements to Nuclear Thermal Propulsion Rockets

is approved in partial fulfillment of the requirements for the degree of

Doctor of Philosophy – Mechanical Engineering
Department of Mechanical Engineering

William Culbreth, Ph.D.
Examination Committee Co-Chair

Alexander Barzilov, Ph.D.
Examination Committee Co-Chair

Yi-Tung Chen, Ph.D.
Examination Committee Member

Georg Mauer, Ph.D.
Examination Committee Member

Monika Neda, Ph.D.
Graduate College Faculty Representative

Alyssa Crittenden, Ph.D.
Vice Provost for Graduate Education &
Dean of the Graduate College

ABSTRACT

Nuclear thermal rocket propulsion has been proposed as a highly efficient technology for space vehicles traveling from earth orbit to the moon, Mars, and other locations in the solar system. With twice the performance of a chemical rocket, nuclear thermal propulsion (NTP) uses the thrust produced by heating hydrogen gas within a thermal nuclear reactor where the exhaust is then passed through a de Laval nozzle to produce supersonic flow. NTP engines were the subject of the NERVA experiments at the Nevada Test Site in the 1970's, and they produced a specific impulse of up to 900 seconds which is almost twice the performance of the Saturn V rocket at 451 seconds.

There are areas where nuclear thermal propulsion technology can be improved. To minimize corrosion within the reactor, different axial and radial neutron reflector geometries are recommended to minimize gradients in the neutron flux within the core resulting in smaller temperature gradients within the fuel. NTP also involves the use of hydrogen propellant at a lower mass flow rate to remove fission decay heat from the core to prevent melting of the fuel. Variable area rocket nozzles were studied to remove decay heat while providing significant thrust. This additional thrust was included in the calculation of the total impulse required during a burn of the rocket engine. Additionally, increases in the temperature of the hydrogen propellant in the reactor chamber translates to higher specific impulse. This could be achieved by using molten uranium carbide fuel and employing high temperature ceramics, including Ta_4HfC_5 and HfC. The temperature of the hydrogen propellant would increase from 2700 K for the NERVA engine to over 4000 K resulting in increased engine efficiency. Finally, an optimization program was written to determine whether the NERVA rocket design could be improved. The same program determined the optimal molten uranium carbide rocket design for space propulsion.

ACKNOWLEDGMENTS

I’d like to thank

- Jesus Christ
- My mom for all her support
- Dr. Culbreth
- John Vargas for helping me graduate from my undergrad (haha I graduated first). You the real one even though you forget I exist sometimes.
- My sister
- Billie Jean
- Charlie, I love you even though you’re ugly and like to bite my face for no reason.
- Vanilla, I love you more. If Lil Wayne knew you, he would’ve known how to love.
- Everyone else
- And Covid… who says you need to test negative to graduate.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS

Symbols Used in Compressible Flow Theory

a	speed of sound, (m/s).
A	cross-sectional area, (m^2).
c_p	specific heat at constant pressure, (J/kg-K).
c_v	specific heat at constant volume, (J/kg-K).
e	specific internal energy, (J/kg).
g_{earth}	gravitational acceleration on the surface of the earth, (9.81 m/s^2).
h	specific enthalpy, (J/kg).
k	ratio of specific heats, c_p / c_v.
$\dot{m}$	mass flowrate, (kg/s).
M	Mach number.
$\widehat{M}$	molecular weight of the gas, (kg/mol).
P	pressure.
R_{gas}	gas constant, (J/kg-K).
$R_{universal}$	universal gas constant, (J/mol-K).
T	temperature.
v	specific volume, (m^3/kg).
ρ	density, (kg/m^3).

Subscripts

d	pertaining to the design or optimal de Laval nozzle.
e	value at the exit of the de Laval nozzle.
max	maximum value corresponding to choked flow at the throat.
o	stagnation value.
*	choked value at the throat at M = 1.
∞	ambient fluid property.

Symbols Used in Nuclear Reactor Theory

B	reactor or material buckling, (1/m).
D	neutron diffusion coefficient, (m).
f	thermal utilization factor.
E	neutron energy, (MeV).
$\vec{J}$	neutron current density, (neutrons/m^2-s).
k_{eff}	effective neutron multiplication factor.
k_∞	infinite neutron multiplication factor.
L	thermal diffusion length, (m).
n	number density, (atoms or molecules / m^3).
p	resonance escape probability.
P_{NL}	non-leakage probability

R_e	extrapolated radius, $R_e = R + \delta_e$

Δ	extrapolation distance, (m).
ε	fast fission factor.
η	thermal fission factor.
λ	neutron mean free path, (m).
ν	number of neutrons produced per thermal fission.
ξ	logarithmic energy decrement or lethargy.
σ	microscopic neutron cross-section, (b).
Σ	macroscopic neutron cross-section, (1/m).
φ	neutron flux, (neutrons/m^2-s).

Subscripts

a	absorption
c	core
e	value at extrapolated boundary
f	fission
r	reflector
s	scattering
t	total
tr	transport

Bessel Functions

$J_o(x)$	Bessel function of the first kind of order zero
$Y_o(x)$	Bessel function of the second kind of order zero
$I_o(x)$	Modified Bessel function of the first kind of order zero
$K_o(x)$	Modified Bessel function of the second kind of order zero

CHAPTER 1

INTRODUCTION

The current state of space propulsion is primarily based on chemical rockets where combusted gases expand through a de Laval nozzle to supersonic velocities and are ejected from the rocket engine to produce thrust. Ion engines are also used to provide the small amounts of thrust necessary to maintain satellites in a stable orbit. More exotic propulsion methods have been proposed for the future, including nuclear fusion, antimatter, and photonic drives.

The journey of humans to Mars is a tantalizing goal and shows the limitations of chemical rockets. Unmanned missions to Mars using chemical rockets can take up to a year or more, depending upon the relative position of earth and Mars. Radiation levels in space make manned flights of this duration dangerous to humans and more efficient space propulsion technology is needed. Nuclear thermal propulsion (NTP) was originally envisioned in the 1950’s and then tested by the U.S. government at the Nevada Test Site in the 1960’s and 70’s under Project Rover and NERVA, Nuclear Engine for Rocket Vehicle Application. With twice the performance of the Saturn V rocket, NTP rocket engines were proposed by NASA in 1968 for the first mission to Mars in 1981. Unfortunately, those plans were cancelled during the Nixon presidency. There is renewed interest in manned missions to Mars and nuclear thermal propulsion has been proposed to decrease the travel time between low earth orbit and Mars orbit. Due to the limitations of chemical rocket technology, manned flights to Mars, the asteroid belt, and to the icy moons of Jupiter and Saturn may rely on nuclear fission propulsion.

1.1 Overview of Current Design

The current state of space propulsion is primarily based on chemical rockets where combusted gases expand through a de Laval nozzle to supersonic velocities and are ejected from the rocket engine to produce thrust. Ion engines are also used to provide the small amounts of thrust necessary to maintain satellites in a stable orbit. More exotic propulsion methods have been proposed for the future, including nuclear fusion, antimatter, and photonic drives. The journey of humans to Mars is a tantalizing goal and shows the limitations of chemical rockets since unmanned missions to Mars can take a year or more depending upon the relative position of earth and Mars. Radiation levels in space make manned flights of this duration dangerous to humans and more efficient space propulsion technology is needed. Nuclear thermal propulsion (NTP) was envisioned in the 1950's and NTP rockets were tested by the U.S. government at the Nevada Test Site in the 1960's and 70's under Project Rover and NERVA. With twice the performance of the Saturn V rocket, NTP rocket engines were proposed by NASA for the first mission to Mars in scheduled for 1981.

1.2 NTP Technology and Space Exploration

The NERVA tests carried out at the Nevada Test Site were successful and NTP technology is currently under investigation by the U.S. Air Force under Project DRACO to develop a nuclear rocket for cis-lunar travel. NTP can still be improved. Problems with the technology include rapid fuel corrosion and the use of hydrogen propellant to remove decay heat. Both issues were addressed in this work. Higher performance requires higher reactor temperatures and a novel reactor design based on molten uranium carbide fuel in hafnium carbide elements was developed. Optimal rocket design is based on many factors, including mass flowrate of propellant, reactor geometry, and neutron reflector geometry. A genetic optimization algorithm was used to study the NERVA reactor and rocket design, as well as the molten uranium carbide reactor, to arrive at optimal designs.

1.3 Brief History of Modern Rockets

The history of modern liquid-fueled rockets began with Pedro Paulet in 1927 and his claim to have experimented with these devices in Paris thirty years earlier. The rockets were considered incredibly powerful at the time when compared to solid fuel rockets [Ordway]. The first documented flight of a liquid-propellant rocket was conducted by Dr. Robert H. Goddard in 1926 in Massachusetts using liquid oxygen and gasoline. This rocket only flew 41 feet from the launch pad, but the technology was successfully demonstrated.

In Germany, Friedrich Sander developed liquid-fuel rockets for Opel RAK and had two successful launches in 1929. Wernher von Braun developed liquid-fuel rockets for the Nazis in the 1930's which culminated in the V-2 rocket starting with liquid oxygen and a mixture of ethanol and water for fuel. Von Braun invented a technique to cool the de Laval nozzle by circulating liquid oxygen through a jacket surrounding the throat of the nozzle before the oxygen was sent to the combustion chamber. This prevented the nozzle from melting. Over 5,000 V-2 rockets were constructed during World War II. The V-2 design is still used in large conventional rockets. A liquid oxidizer and fuel are pumped by a turbopump into the combustion chamber and the hot exhaust is directed through a de Laval nozzle where the subsonic gases are increased to supersonic velocities. The thrust of a rocket engine is based on the product of the mass flowrate of the exhaust and the exit velocity. Rocket performance is typically expressed in terms of specific impulse, I_{sp}, which is the ratio of thrust to the weight of propellant consumed per unit time.

Wernher von Braun used his expertise as director of NASA to design new rocket systems, including the Saturn V, that resulted in the first humans on the moon [Bilstein]. The last two stages of the Saturn V rocket achieved a specific impulse of 421 seconds, and this may be the highest value ever achieved by a liquid-fuel rocket engine.

1.4 Project Rover

Scientists working on the Manhattan Project proposed using atomic bombs to provide propulsion for space vehicles and Stan Ulam and Joseph Evertt presented their ideas in a paper in 1947. Robert Bussard, in 1953, proposed using hydrogen as a propellant for nuclear propulsion in space and concluded that graphite could serve as an adequate neutron moderator due to its high melting temperature. The Department of Defense was conducting work on the design of intercontinental ballistic missiles under the Atlas program in the 1950's and nuclear propulsion was proposed for the last stage of the rocket.

This project was transferred to NASA in 1957 [Watson] and a design was formed for a nuclear thermal propulsion rocket where hydrogen propellant would pass through a turbopump into the reactor inlet where the hydrogen would travel through small channels in the reactor to be heated by fission. The reactor would use graphite as a moderator and beryllium reflectors would help moderate neutrons and reduce the required size of the critical reactor core. In 1960, NASA created the Space Nuclear Propulsion Office (SNPO) to manage the nuclear rocket project. By 1961, the new program for the development of space nuclear rocket propulsion, Project Rover was initiated.

Project Rover explored the development of nuclear thermal rocket engines using a turbopump to force hydrogen through channels in a nuclear reactor and then pass through a de

Laval nozzle to produce thrust. The hydrogen was to be stored in liquid form below 20 K. Hydrogen temperatures at the exit of the nozzle would exceed 2500 K and the Rover program explored materials that could withstand these temperatures.

The first testing conducted under this program involved Kiwi A and B with reactor power levels of up to 450 MW. These engines tests were conducted on the ground at the Nevada Test Site and operated for several minutes. The reactors employed beryllium reflectors to reduce the critical mass of UO_2 required for criticality and graphite plates served as a moderator. The Kiwi tests demonstrated that nuclear thermal propulsion was viable and led to the Phoebus 1A, 1B, 1C and 2A tests starting in 1965. Phoebus 1A exploded resulting in a cleanup of radioactive material. Phoebus 1B testing began in 1967 and ran with 588 MW of power for 150 seconds resulting in a maximum fuel temperature of 2444 K. These tests led to the development of Pewee in 1968, the third phase of Project Rover.

The Pewee rocket [Benensky, N-Division] was another ground-based test of a small nuclear engine producing 25,000 lbf of thrust with a power level of 503 MW. The reactor was small with a diameter of 0.6 m and a length of 1.32 m. The tests produced a specific impulse as high as 900 seconds, roughly double that of the Saturn V. Pewee was operated for 2400 seconds at full power indicating that NTP engines could provide propulsion for space missions. The mass flowrate of hydrogen was 18.8 kg/s, and the exit temperature of the hydrogen exceeded 2500 K. The reactor was started and stopped 3 times and successfully completed each burn. Hexagonal fuel elements made from 93% enriched uranium carbide in graphite with zirconium served as the fuel, and each fuel element with a total length of 1.32 m had 19 coolant channels for the flow of hydrogen propellant. The coolant channel diameter was 2.54 mm, and a thin coating of niobium carbide on the inside of each channel helped decrease corrosion of the carbon fuel with hot hydrogen propellant. The coating was later changed to zirconium carbide which performed better.

The Pewee tests were followed by the Nuclear Furnace for the testing of improved composite fuels.

1.5 NERVA

Under NASA's NERVA program, a SNRE (small nuclear reactor engine) was designed based on the success of Pewee. By 1967, the program lost Congressional funding, and the nuclear thermal propulsion program in the U.S. reached an end.

Dr. Stanley Borowski [Borowski] and his team at NASA Lewis in Ohio continued designs of NTP systems for travel to Mars and the asteroid belt. Their designs include hybrid systems that used the main engine to provide thrust and to provide electrical power for long space missions [Finseth]. They placed large hydrogen storage tanks between the reactor and the crew quarters to decrease radiation exposure and called for a rotation of the reactor and crew habitation about the center of mass of the rocket to produce artificial gravity during the transit to Mars. Dr. Mike Houts and his team at NASA Marshall have developed designs for nuclear rockets that utilize low enriched uranium (LEU) to comply with current U.S. policy to not employ high enriched uranium (HEU) in space.

Other engineering teams have been working on improvements to the NERVA SNRE design over the years since the NERVA experiments. KANUTER, the Korea Advanced Nuclear Thermal Engine Rocket [Nam], was designed with a higher melting temperature fuel to achieve an estimated 945 seconds of specific impulse. They also noted that their thrust to weight ratio of 5:1 showed a distinct improvement over the value of 2.9:1 for the SNRE design. They noted that the thermal conductivity of UC_2 is only 18 W/m-K and melts at 2,710 K, while a carbide formed from a mixture of (U, Zr, Ta)C could achieve melting temperatures of 3,900 K with a thermal conductivity of 50 W/m-K. To decrease the mass of the reactor, zirconium hydride was used as a moderator in the SNRE and this was replaced by lighter Li_7H decreasing the mass of moderator by 33% for a 100 MW reactor.

1.6 Current Nuclear Thermal Propulsion Programs

In 2020, the U.S. agency DARPA (Defense Advanced Research Projects Agency) issued an invitation to aerospace companies to develop designs for a cislunar rocket based on nuclear thermal propulsion technology. This system is to be designed for missions between low earth orbit to lunar orbit [Howell]. In 2021, NASA chose 3 companies: General Atomics, Blue Origin, and Lockheed Martin, to work on the first phase of this research which is to test NTP technology in a mission above low earth orbit by 2025.

The Russians have also developed nuclear thermal rocket engines using hydrogen propellant. Their RD-0410 rocket began development in 1965 and used uranium carbide fuel in a tungsten carbide matrix with a zirconium hydride moderator. To prevent fuel corrosion, one percent heptane was added to the hydrogen. This rocket was intended for missions to Mars in the 1980’s.

Russia is also developing TEM (Transport and Energy Module) for missions to the moon, Venus, and Jupiter in the 2030’s [RIA Novosti]. This is a nuclear electric rocket design generating one megawatt of electricity that can propel spacecraft with 18 newtons (4 lbf) of thrust and a specific impulse of 7000 seconds. The reactor is a GCFR design (gas cooled fast reactor).

1.7 Thermal Nuclear Reactor Design

In NTP rocket design, the heat added to hydrogen propellant is produced by a nuclear fission reactor. The reactor has small coolant channels that run along the length of the reactor to allow the hydrogen propellant to be heated from an initial temperature of about 120 K to a final temperature than can exceed 2700 K. The hotter that the exhaust gas is, the greater the specific impulse and thrust of the rocket. This is shown by the equation for specific impulse where T_o represents the stagnation temperature at the exit of the reactor before the converging chamber of the de Laval nozzle.

$$I_{sp} = \frac{1}{g_e}\sqrt{\frac{2\,k\,T_o}{k\,-\,1}\,\frac{R_{universal}}{\widehat{M}}\left[1\,-\,\left(\frac{p_e}{p_o}\right)^{\frac{k\,-\,1}{k}}\right]} \tag{1}$$

For an optimal, or design nozzle, the exit pressure, p_e, will be close to the ambient pressure. In space, the ambient pressure is essentially zero, so $p_e = 0$ and the temperature, *To*, is limited by the melting temperature of the fuel or its protective coating. The fission reactors used for NTP are unique and are designed with fuel that have a high melting temperature. Lightweight material is used as a neutron reflector on the sides of the reactor to maintain a high neutron flux within the reactor core. The hydrogen plays two rolls in NTP design. Hydrogen gas gains heat from the reactor and acts as a propellant for the rocket. Hydrogen also acts as a neutron moderator in the core where multiple collisions between atoms of hydrogen and neutrons results in heating of the hydrogen and slowing down of the fast neutrons produced by fission. The role of a moderator is essential in thermal fission reactor design. These concepts are described in this chapter along with a summary of the equations that affect NTP reactor design. This chapter also includes a derivation of the neutron flux in a reflected reactor core and the techniques used to determine the neutron flux and power generated within an NTP reactor.

1.8 Thrust Generated by a Rocket

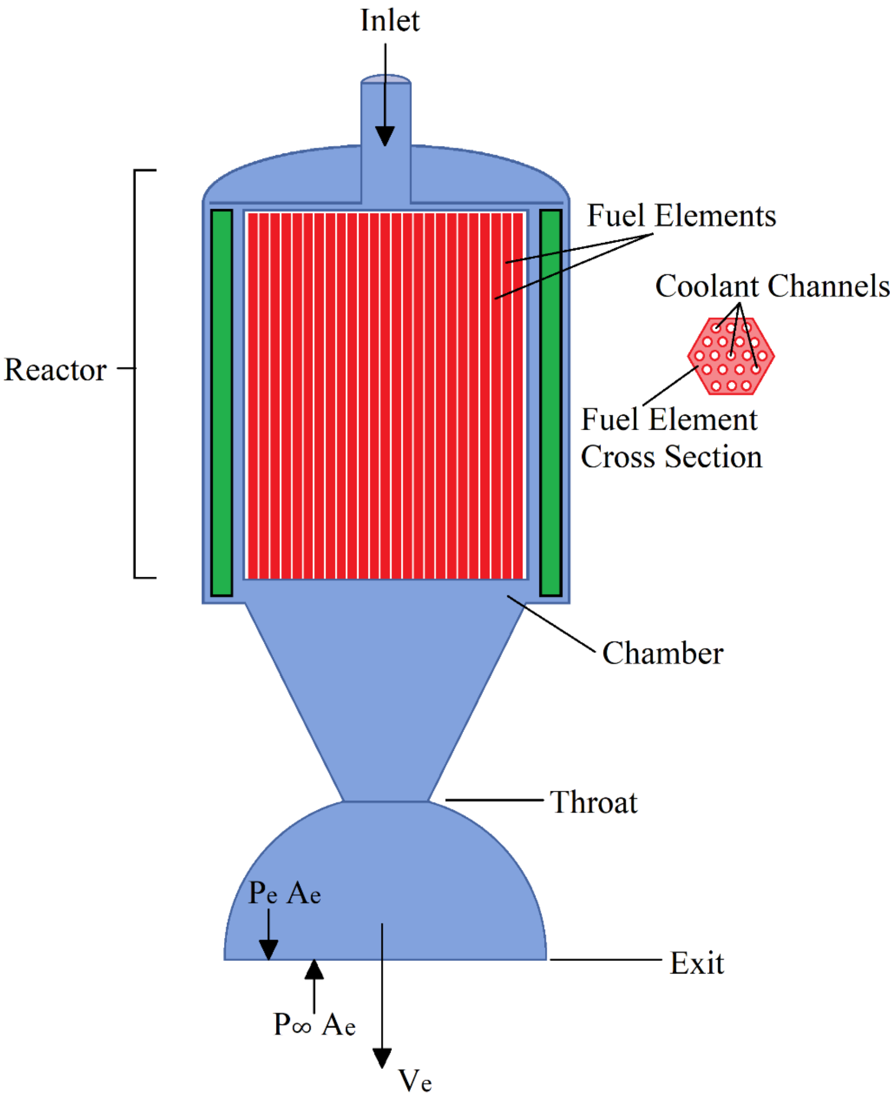


Fig. 1. Forces and Momentum Transfer on a Rocket

Conservation of momentum can be applied to a rocket engine to determine the thrust, which is the effective force on the rocket in the axial direction. Based on the geometry shown in Fig. 1 and assuming steady flow through the rocket, the thrust can be computed.

$$\sum \vec{F} = \oiint_{\substack{control \\ surface}} \rho \vec{V}\, \vec{V} \cdot d\vec{A} \tag{2}$$

$$thrust + p_\infty A_e - p_e A_e = \dot{m} V_e \tag{3}$$

$$thrust = \dot{m} V_e + (p_e - p_\infty) A_e \tag{4}$$

For an optimally designed nozzle, the exit pressure would equal the ambient pressure and the last term would be zero.

1.9 Derivation of the Specific Impulse Equation

Fig. 1 also shows the chamber below the reactor where the stagnation temperature and pressure are given by T_o and p_o. Specific impulse can be related to these values to provide significant insight into how they affect the performance of the rocket engine.

$$I_{sp} = \frac{thrust}{weight\ of\ propellant\ used\ per\ unit\ time} = \frac{\dot{m} V_e}{\dot{m}\, g_e} = \frac{V_e}{g_e} \tag{5}$$

The flow from the exit of the reactor and within the chamber moves isentropically out through the exit of the nozzle.

$$V_e^2 = M_e^2\, a_e^2 = M_e^2\, k\, R\, T_e \tag{6}$$

The temperature and pressure at any point within the nozzle can be related back to the stagnation values.

$$\frac{T_o}{T_e} = \left(1 + \frac{k-1}{2} M_e^2\right) \tag{7}$$

$$\frac{p_o}{p_e} = \left(1 + \frac{k-1}{2} M_e^2\right)^{\frac{k}{k-1}} \tag{8}$$

This second equation can be used to solve for the exit Mach number in terms of the pressure ratio.

$$M_e^2 = \frac{2}{k-1}\left(\left(\frac{p_o}{p_e}\right)^{\frac{k-1}{k}} - 1\right) \tag{9}$$

By replacing the Mach number and exit temperature with values from equations (7) and (9), equation (6) can be expressed in terms of stagnation pressure and temperature.

$$V_e^2 = \left\{\frac{2}{k-1}\left[\left(\frac{p_o}{p_e}\right)^{\frac{k-1}{k}} - 1\right]\right\} k\ R_{gas}\left(\frac{T_o}{1 + \frac{k-1}{2} M_e^2}\right) \tag{10}$$

$$V_e^2 = \frac{2\ k\ R_{gas} T_o}{k-1}\left[\frac{\left(\frac{p_o}{p_e}\right)^{\frac{k-1}{k}} - 1}{1 + \frac{k-1}{2} M_e^2}\right] \tag{11}$$

$$V_e^2 = \frac{2\ k\ R_{gas} T_o}{k-1}\left[\frac{\left(\frac{p_o}{p_e}\right)^{\frac{k-1}{k}} - 1}{1 + \frac{k-1}{2}\,\frac{2}{k-1}\left(\left(\frac{p_o}{p_e}\right)^{\frac{k-1}{k}} - 1\right)}\right] \tag{12}$$

$$V_e^2 = \frac{2\ k\ R_{gas} T_o}{k-1}\left[\frac{\left(\frac{p_o}{p_e}\right)^{\frac{k-1}{k}} - 1}{\left(\frac{p_o}{p_e}\right)^{\frac{k-1}{k}}}\right] \tag{13}$$

$$V_e^2 = \frac{2\,k\,R_{gas}T_o}{k-1}\left[1-\left(\frac{p_e}{p_o}\right)^{\frac{k-1}{k}}\right] \tag{14}$$

$$I_{sp} = \frac{v_e}{g_e} = \frac{1}{g_e}\sqrt{\frac{2\,k\,R_{gas}T_o}{k-1}\left[1-\left(\frac{p_e}{p_o}\right)^{\frac{k-1}{k}}\right]} \tag{15}$$

1.10 Research Plan

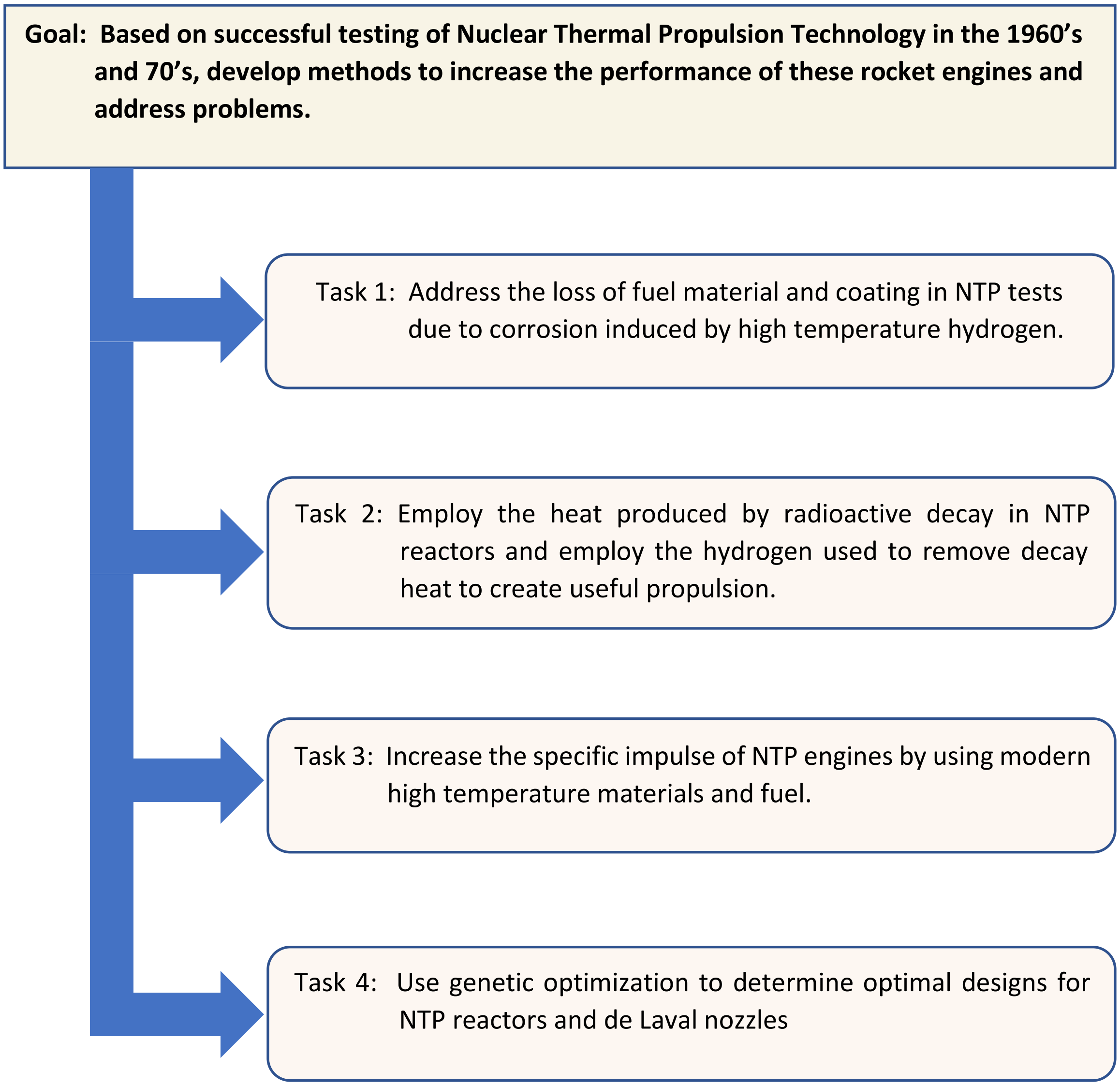
The NERVA tests carried out at the Nevada Test Site were successful and NTP technology is currently under investigation by the U.S. Air Force under Project DRACO to develop a nuclear rocket for cis-lunar travel. NTP can still be improved. Problems with the technology include rapid fuel corrosion and the use of hydrogen propellant to remove decay heat. Both issues were addressed in this work. Higher performance requires higher reactor temperatures, and a novel reactor design based on molten uranium carbide fuel in hafnium carbide elements was developed. Optimal rocket design is based on many factors, including mass flowrate of propellant, reactor geometry, and neutron reflector geometry. The Russian NTP design will be adding 1% heptane to the hydrogen propellant to reduce corrosion. Heptane will, unfortunately, also decrease the specific impulse of the rocket.

To address the issue of fuel corrosion, a genetic optimization algorithm was used to study the Pewee reactor and rocket design to see if simple reactor modifications could be done to minimize corrosion. The results of this study are reported in Chapter 3.

Fission reactors generate fission products. These radioactive isotopes decay and emit particles, gamma rays, and decay heat. The decay heat can be significant, and it decreases rapidly after the reactor is shut down. If the reactor has no coolant flow, decay heat can cause the temperatures within the reactor core to rise above the melting temperature of the fuel. To avoid the fuel from melting, current NTP technology requires hydrogen propellant to flow through the reactor core to remove decay heat, and this hydrogen is subsequently vented into space. The de Laval nozzle used to generate thrust in an NTP rocket is designed for one mass flowrate. If the flow of propellant is below this design flowrate, the hydrogen will exit the nozzle at subsonic velocities producing little thrust. To capture the decay heat and to effectively use this hydrogen coolant to generate additional thrust, the use of variable area nozzles was explored, and the hydrogen savings proved to be significant. This method of capturing decay heat is discussed in Chapter 4.

To further increase the performance of an NTP engine so that the specific impulse will exceed 900 seconds, higher temperatures are required at the entrance of the de Laval nozzle. Pewee, the nuclear rocket tested at the Nevada Test Site in 1968 to 1972, contained fuel that melted at roughly 2700 K. This limited the maximum hydrogen propellant temperature. Today, new materials are available with much higher melting temperatures. In this report, carbides of refractory elements concentrating on hafnium and tantalum carbides are considered as possible reactor materials. Ampules of hafnium carbide enclosing molten uranium carbide fuel can raise the hydrogen temperature to nearly 4200 K resulting in a significant increase in specific impulse. This design is proposed in Chapter 5 with a recommended design for a NTP rocket engine with a specific impulse of about 1100 seconds.

The final method that is employed in this dissertation to enhance NTP design involve the use of a genetic optimization program to determine the best combination of reactor diameter, length, coolant channel diameter and pitch, reflector thickness, and propellant mass flowrate to produce a desired thrust with a defined reactor power with the greatest possible performance. The objective function for this optimization is the ratio of reactor weight to specific impulse. The analysis technique and results are presented in Chapter 6. Fig. 2 outlines the different goals presented in this report.



Goal: Based on successful testing of Nuclear Thermal Propulsion Technology in the 1960's and 70's, develop methods to increase the performance of these rocket engines and address problems.

Task 1: Address the loss of fuel material and coating in NTP tests due to corrosion induced by high temperature hydrogen.

Task 2: Employ the heat produced by radioactive decay in NTP reactors and employ the hydrogen used to remove decay heat to create useful propulsion.

Task 3: Increase the specific impulse of NTP engines by using modern high temperature materials and fuel.

Task 4: Use genetic optimization to determine optimal designs for NTP reactors and de Laval nozzles

Fig. 2. Goals and Tasks for this Research

Next, Chapter 2 presents theory that is applied to NTP analysis. This includes compressible flow, neutron diffusion theory and nuclear criticality, and optimization theory. The reader may want to read ahead to Chapter 3 if they are familiar with the theory in each of these areas.

CHAPTER 2

THEORY

2.1 Thermodynamic Model

In NTP rocket design, the heat added to hydrogen propellant is produced by a nuclear fission reactor. The reactor has small coolant channels that run along the length of the reactor to allow the hydrogen propellant to be heated from an initial temperature of about 120 K to a final temperature than can exceed 2700 K. The hotter the exhaust gas, the greater the specific impulse and the thrust of the rocket. This is shown by the equation for specific impulse where T_o represents the stagnation temperature at the exit of the reactor before the converging chamber of the de Laval nozzle.

$$I_{sp} = \frac{1}{g_e} \sqrt{\frac{2\,k\,T_o}{k-1} \frac{R_{universal}}{\widehat{M}} \left[1 - \left(\frac{p_e}{p_o}\right)^{\frac{k-1}{k}}\right]} \tag{16}$$

For an optimal, or design nozzle, the exit pressure, p_e, will be close to the ambient pressure. In space, the ambient pressure, p_e, is essentially zero, and the temperature, T_o, is limited by the melting temperature of the fuel or its protective coating. The fission reactors used for NTP are unique and are designed with fuel that have a high melting temperature. Lightweight material is used as a neutron reflector on the sides of the reactor to maintain a high neutron flux within the reactor core. The hydrogen plays two rolls in NTP design. Hydrogen gas gains heat from the reactor and acts as a propellant for the rocket. Hydrogen also acts as a neutron moderator in the core where multiple collisions between atoms of hydrogen and neutrons result in heating of the hydrogen and slowing down of the fast neutrons produced by fission. The role of a moderator is essential in thermal fission reactor design. These concepts are described in this chapter along with a summary of the equations that affect NTP reactor design. This chapter also includes a derivation of the neutron flux in a reflected reactor core and the techniques used to determine the neutron flux and power generated within an NTP reactor.

The neutron flux within the reactor governs the rate of nuclear fission and heat generation, so an understanding of the neutron flux and how it changes within the reactor is discussed in the following sections. The theoretical derivations in this chapter are limited to one energy group theory representing only thermal neutrons since NTP reactors rely upon thermal neutrons to produce the majority of fission collisions between neutrons and uranium nuclei. Thermal neutrons have a Maxwell-Boltzmann energy distribution with a peak energy that is related to the reactor core temperature. Room temperature neutrons have an average energy of 0.0253 eV (electron volts) with an average velocity of 2,200 m/s. For the analysis of the neutron multiplication factor, $k_{eff,}$ in Chapters 5 and 6, MCNP (Monte Carlo Neutron Production) was used. This is a comprehensive Monte Carlo simulation computer program developed by the Los Alamos National Laboratory that uses extensive neutron cross-section libraries to model neutrons across all energies.

NTP reactor cores also have neutron reflectors that help to reflect neutrons back into the core. The presence of these reflectors, usually fabricated out of beryllium, decrease the required size and mass a critical reactor. Diffusion theory will be used to describe the benefits of neutron reflectors and how they impact the neutron flux within the core.

2.1.1 Fermi Diffusion Theory

Neutron diffusion theory was developed in the 1940's to model the movement of neutrons through material. This theory is dependent upon the use of Fick's law of diffusion to estimate neutron current density, $\vec{J}$, as a function of neutron flux, $\varphi(\vec{r})$, and the diffusion coefficient for neutrons through the material, *D*. Heat transfer is also modeled using diffusion theory as expressed in equation (18) and neutron current density is analogous to heat flux while neutron flux is analogous to temperature.

$$\vec{J} = -D\,\vec{\nabla}\varphi \tag{17}$$

$$\vec{q} = -k\,\vec{\nabla}T \tag{18}$$

In diffusion theory, the production, loss, and transport of neutrons in a nuclear reactor governs the rate of fission and the reactor behavior. The rate of fission can be given in terms of the macroscopic fission cross-section, Σ_f, the number density of neutrons, *n* (neutrons/m^3), the average neutron velocity, v, the neutron flux, φ (neutrons/m^2-s), the neutron diffusion coefficient, *D*, the neutron absorption cross-section Σ_a, and ν, the number of neutrons produced per fission. These are combined into the neutron transport equation shown below. In a control volume within a reactor core, the time rate of change of neutron flux equals the net outflow of neutrons by diffusion, the loss by absorption within the fuel matrix, and the production of neutrons by fission.

$$\frac{1}{v}\,\frac{\partial\varphi(\vec{r})}{\partial t} = -D\,\nabla^2\varphi(\overrightarrow{r}) - \Sigma_a\,\varphi(\vec{r}) + \nu\,\Sigma_f\,\varphi(\vec{r}) \tag{19}$$

For a steady-state reactor, the time derivative is zero and this equation reduces to a simpler form.

$$\nabla^2 \varphi(\vec{r}) + \; B^2 \; \varphi(\vec{r}) = 0 \tag{20}$$

In this equation, ***B*** is referred to as the material buckling due to the similarity of this differential equation to the formula for the buckling of a column. Since this is a steady-state equation, the solutions are in the form of eigenfunctions based on an eigen condition and eigenvalues.

In the subsequent sections of this chapter, equation (20) will be solved for different reactor geometries to obtain the required size or buckling of the reactor and to obtain the neutron flux distribution within the reactor core. The neutron flux governs the production of heat due to fission based on *G*, the energy emitted per fission.

$$\dot{q}'''(\vec{r}) = G \; \Sigma_f \; \varphi(\vec{r}) \tag{21}$$

2.1.2 Infinitely Long Bare Cylindrical Reactor Core

A simple example of a reactor that approximates an NTP reactor is an infinitely long cylindrical core. Without a neutron reflector at the outer wall, the reactor is considered bare and neutrons will leak from the outside wall of the core. For a steady state reactor core the neutron flux in the radial direction, $\varphi(r)$, the effective neutron multiplication factor, k_{eff}, and the neutron current density can be computed.

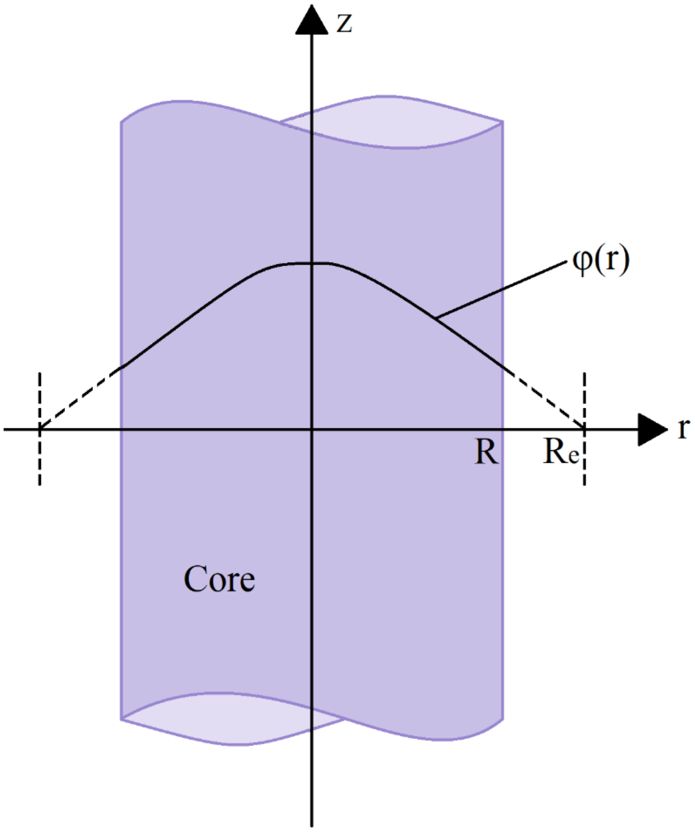


Fig. 3. Infinitely Long Bare Cylindrical Reactor Core

The steady-state neutron diffusion equation is a second order ordinary differential equation that requires two boundary conditions to ensure that the flux at the center of the reactor is finite and the neutron flux is zero at the extrapolated boundary, $r = R_e$. The extrapolated radius can be found from $R_e = R + \delta$ where the extrapolation distance, $\delta = 0.71 / \Sigma_{tr}$, and the geometric radius of the reactor is R.

$$\nabla^2\varphi + B^2\varphi = 0 \tag{22}$$

$$\frac{1}{r}\frac{\partial}{\partial r}\left(r\frac{\partial\varphi}{\partial r}\right) + B^2\varphi = 0 \tag{23}$$

Boundary conditions:

$$\varphi(0) = finite \tag{24}$$

$$\varphi(R_e) = 0 \tag{25}$$

Equation (23) is an example of Bessel's Equation. The general solution is composed of Bessel functions of the first and second kind of order zero. Their behavior is shown in Fig. 4.

$$\varphi(r) = A_1 J_o(r) + A_2 Y_o(r) \tag{26}$$

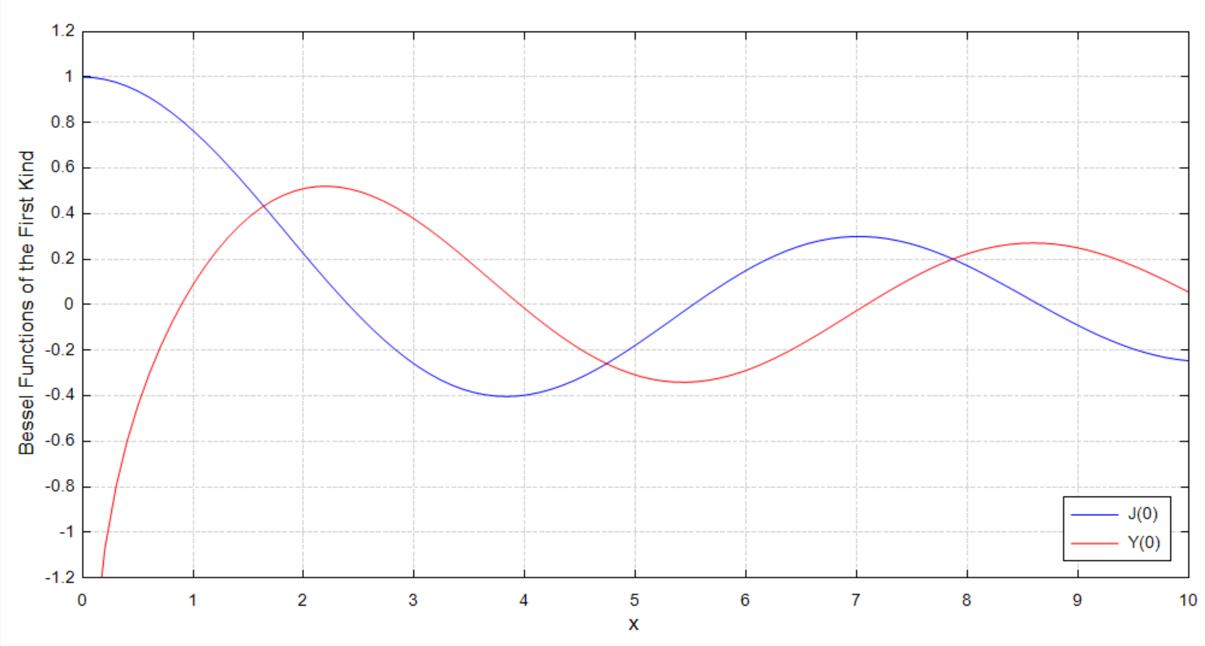


Fig. 4. Bessel Function of the First (Blue) and Second (Red) Kind of Order Zero

The application of the first boundary condition to the general solution demonstrates that A_2 must be zero since $Y_o(0) = -\infty$. Application of the second boundary condition results in an eigen condition.

$$\varphi(R_e) = A_1 J_o(R_e) + A_2 Y_o(R_e) = 0 \tag{27}$$

$$A_1\ J_o(B\ R_e) = 0 \tag{28}$$

The eigen condition can be met by requiring A_1 to be zero, resulting in a trivial solution, or by setting $B\ R_e$ equal to eigenvalues that result in $J_o(B\ R_e)$ equal to zero.

$$B\ R_e = 2.405, 5.5201, 8.6537\ \ldots \tag{29}$$

These eigenvalues represent all possible solutions where $B\ R_e = 2.405$ is the fundamental solution and the remaining terms are harmonics. By incorporating time as an independent variable in this solution, it can be shown that the harmonics quickly converge to zero over time and the fundamental solution is the only one that persists. Equation (29) can be used to find the critical radius from the extrapolated radius, R_e based on the material buckling.

Based on the fundamental solution, only, and by noting that the neutron flux at the center of the reactor is φ_o, the general solution is an eigenfunction and can be written as:

$$\varphi(r) = \ \varphi_o\, J_o\left(\frac{2.405\ R}{R_e}\right) \tag{30}$$

This equation does a good job of predicting the changes in flux in the radial direction as shown in Fig. 5., but an analysis of flux in both radial and axial directions will be covered in section 3.4. Fig. 5. is based on the dimensions, in meters, of the Pewee nuclear rocket engine.

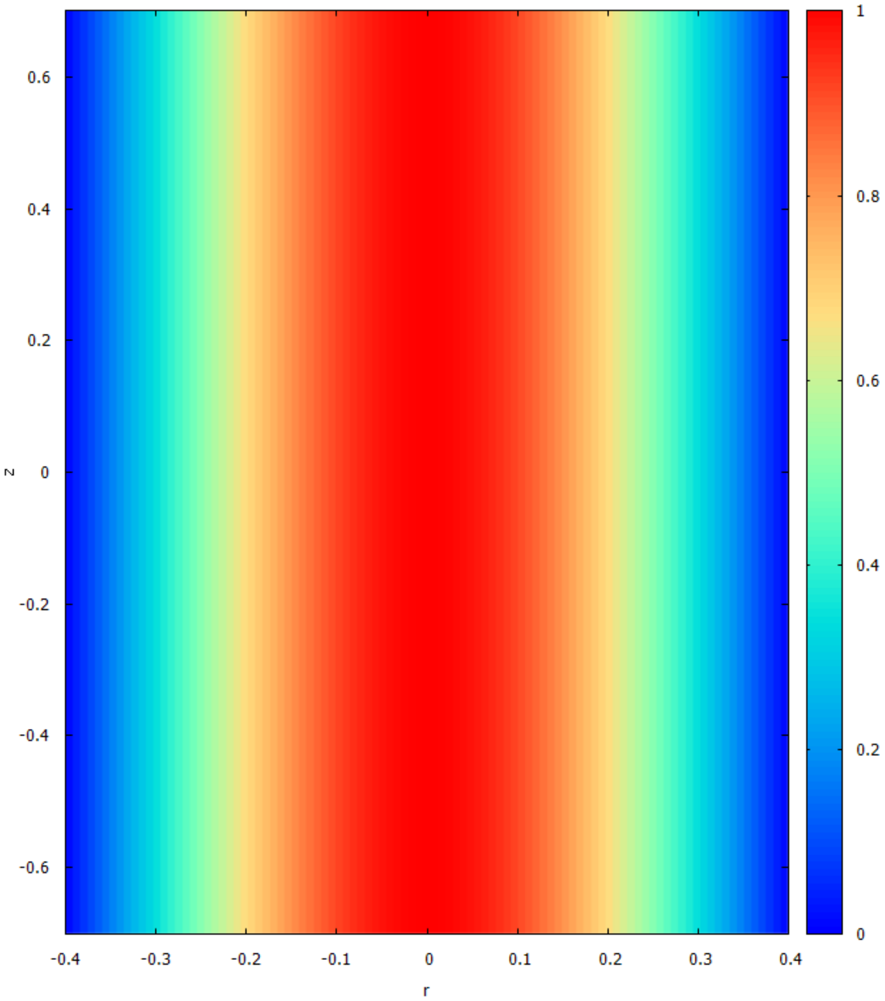


Fig. 5. Neutron Flux in an Infinitely Long Cylindrical Reactor

The power generated by nuclear fission in the reactor core can be used to compute the peak neutron flux, φ_o. The differential power can be written in terms of G, the amount of energy converted to heat per fission.

$$dP = G\ \Sigma_f\ \varphi(r)\ d\tau \tag{31}$$

The differential volume, $d\tau$, can be written as:

$$d\tau = 2\,\pi\,r\,dr\,dz \tag{32}$$

By looking at the power per unit height, *dP' = dP/dz*:

$$P' = G\,\Sigma_f \int_{r=0}^{R} \varphi_o\,J_o\left(\frac{2.405\,r}{R_e}\right)\,(2\,\pi\,r\,dr) \tag{33}$$

By making use of the identity:

$$\int r\,J_o(a\,r)\,dr = \frac{r\,J_1(a\,r)}{a} \tag{34}$$

The power generated per unit height, *P'*, is computed next.

$$P' = 2\,\pi\,G\,\Sigma_f\,\varphi_o \left[\frac{R_e\,R\,J_1\left(\frac{2.405\,R}{R_e}\right)}{2.405}\right] \tag{35}$$

The Bessel function of the first kind of order 1 is shown in Fig. 6.

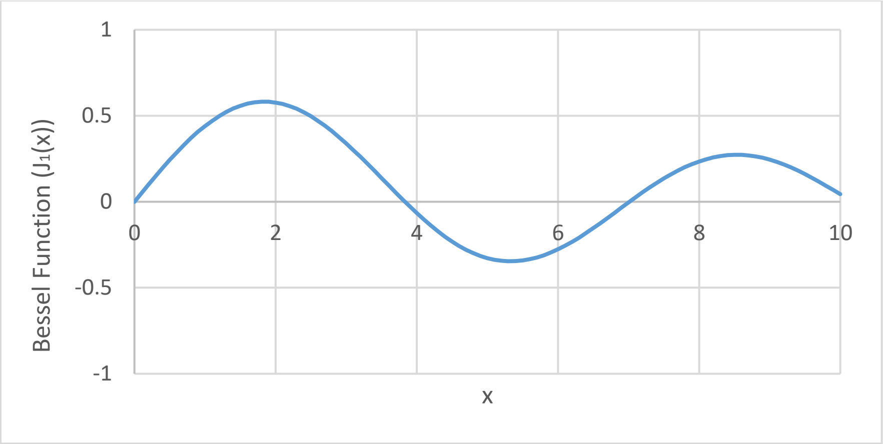


Fig. 6. Bessel Function of the First Kind of Order 1

The power per unit height can be computed. The neutron flux at the center of the core can also be written in terms of the power.

$$\varphi_o = \frac{2.405\, P'}{2\, \pi\, G\, \Sigma_f\, R_e\, R\, J_1\left(\frac{2.405\, R}{R_e}\right)} \tag{36}$$

One last concept for this core geometry involves the leakage of neutrons from the core. The boundary condition assumes that the reactor core is surrounded by a vacuum. Neutrons can leak out, but there are no atoms in the vacuum, so there is nothing capable of reflecting neutrons back into the core if they escape through the wall.

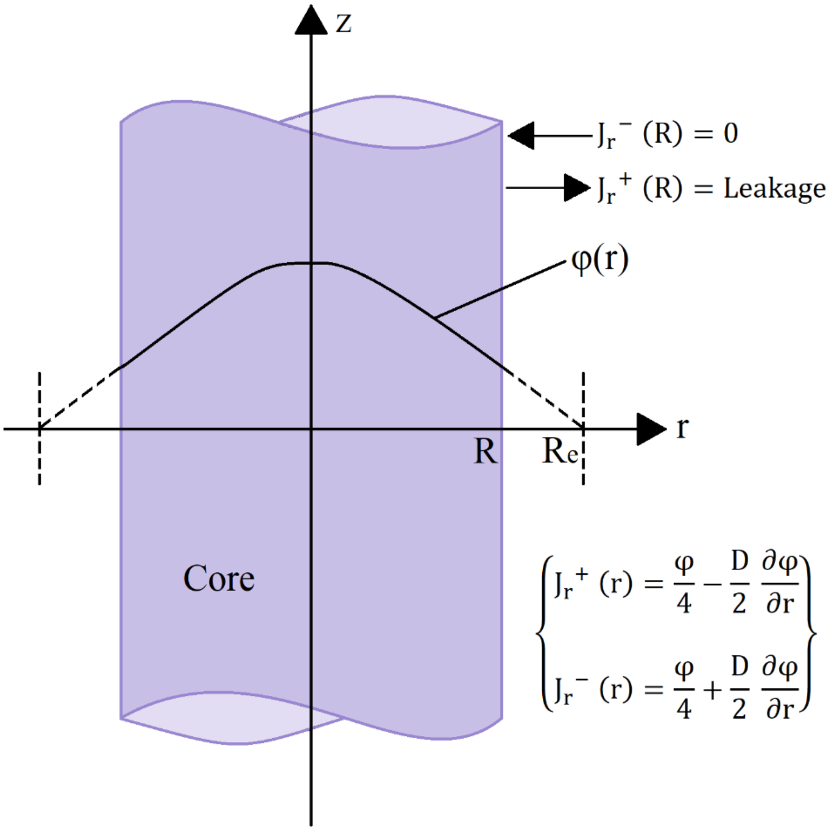

z
J_r^- (R) = 0
J_r^+ (R) = Leakage
φ(r)
r
R
Re
Core
$\left\{ \begin{aligned} J_r^+ (r) &= \frac{\varphi}{4} - \frac{D}{2}\frac{\partial \varphi}{\partial r} \\ J_r^- (r) &= \frac{\varphi}{4} + \frac{D}{2}\frac{\partial \varphi}{\partial r} \end{aligned} \right\}$

Fig. 7. Neutron Current Density Throughout the Outer Wall of the Reactor

Since the neutron flux is known, the neutron current density passing out through the reactor wall can be computed.

$$J_r^+(r) = \frac{\varphi}{4} - \frac{D}{2}\frac{\partial \varphi}{\partial r} \tag{37}$$

$$J_r^+(R) = \frac{\varphi_o}{4}\left[J_o\left(\frac{2.405\,R}{R_e}\right) + \frac{2\,D\,(2.405)}{R_e}\,J_1\left(\frac{2.405\,R}{R_e}\right)\right] \tag{38}$$

The outflowing neutron current density in equation (38) represents a loss in neutrons within the core that could otherwise induce fission within the core. To capture these lost neutrons, a neutron reflector could be added to the outside wall of the reactor. Reflector material typically has a low Z value (low atomic number), a low neutron absorption cross-section, and a good scattering cross-section. For space applications, the mass of the reflector must be minimal and beryllium is usually chosen. The effect of a reflector is discussed in the next section since reflectors play a very important role in space reactor design.

2.1.3 Radially Reflected Cylindrical Core

The geometry of the reactor core in an NTP rocket is typically a cylinder, and to reduce the mass of uranium required to make the reactor critical, a neutron reflector is added to the outside radius of the reactor core. This is shown in Fig. 8. The reflector must be made of a material with a low neutron absorption cross-section, with a relatively high melting temperature, and it must be as light as possible to be launched with the reactor into space. In the ROVER and NERVA programs, beryllium served as a neutron reflector. The average thermal neutron cross-section for beryllium is 0.0076 barns, a very low value. The density is only 1,850 kg/m^3, over 30% less dense than aluminum metal. The melting temperature of pure beryllium metal is 1,560 K, much lower than the 2,620 K melting temperature of the uranium carbide fuel. The thermal conductivity of beryllium is 200 (W/m-K) at room temperature, which is about 15% lower than aluminum metal.

In the design of Pewee, a combination of a radial reflector and an axial reflector at the hydrogen inlet of the reactor were used. To analyze the benefit of a radial reflector, a diffusion model of an infinitely long cylindrical reactor with a radial reflector is completed in this section.

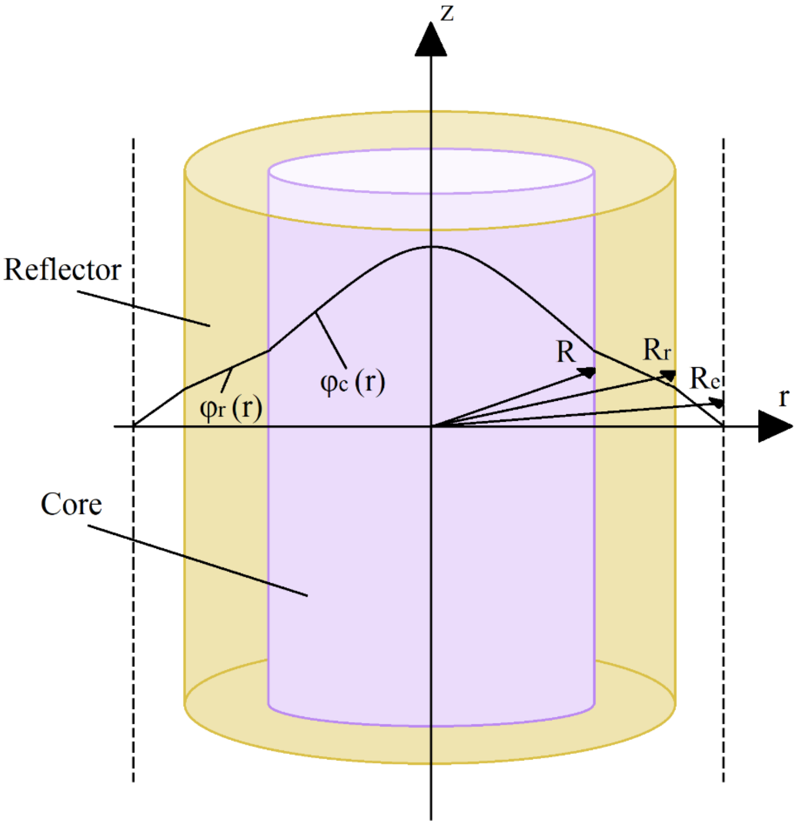


Fig. 8. Infinite Cylindrical Reactor Core with a Radial Reflector

Beginning with the geometry shown in Fig. 8, the steady-state neutron diffusion equation is different in the reflector than it is in the reactor core.

$$\nabla^2 \varphi_c + B_c^2 \, \varphi_c = 0 \tag{39}$$

$$\nabla^2 \varphi_r - \frac{\varphi_r}{L_r^2} = 0 \tag{40}$$

Both the neutron flux and the neutron current density in the radial direction must agree at the boundary between the core and the reflector, corresponding to $r = R$. The Laplacian operator in equations (39) and (40) is limited to variations in the radial direction.

$$\nabla^2 \varphi = \frac{1}{r} \frac{\partial}{\partial r} \left(r \frac{\partial \varphi}{\partial r} \right) \tag{41}$$

For the reactor core and in the reflector, the neutron transport equations can now be defined.

$$\frac{1}{r} \frac{\partial}{\partial r} \left(r \frac{\partial \varphi_c}{\partial r} \right) + B_c^2 \, \varphi_c = 0 \tag{42}$$

$$\frac{1}{r} \frac{\partial}{\partial r} \left(r \frac{\partial \varphi_r}{\partial r} \right) - \frac{\varphi_r}{L_r^2} = 0 \tag{43}$$

Equation (42) is Bessel's equation with solutions in terms of Bessel functions of the first and second kind of order zero. The subscripts "r" and "c" are used to indicate a property of the core or the reflector. The general form of Bessel's equation is:

$$x^2 y'' + x y' + (x^2 - n^2) y = 0 \tag{44}$$

The general solution contains Bessel functions of the first and second kind of order zero.

$$\varphi_c(r) = A_1 \, J_o(B_c \, r) + A_2 \, Y_o(B_c \, r) \tag{45}$$

Equation (43) is a modified Bessel's equation, and the general solution can be written in terms of modified Bessel functions of the first and second kind of order zero.

$$x^2 y'' + x y' - (x^2 - n^2) y = 0 \tag{46}$$

The behavior of these functions is shown in Fig. 9.

$$\varphi_r(r) = A_3 \, I_o\left(\frac{r}{L_r}\right) + A_4 \, K_o\left(\frac{r}{L_r}\right) \tag{47}$$

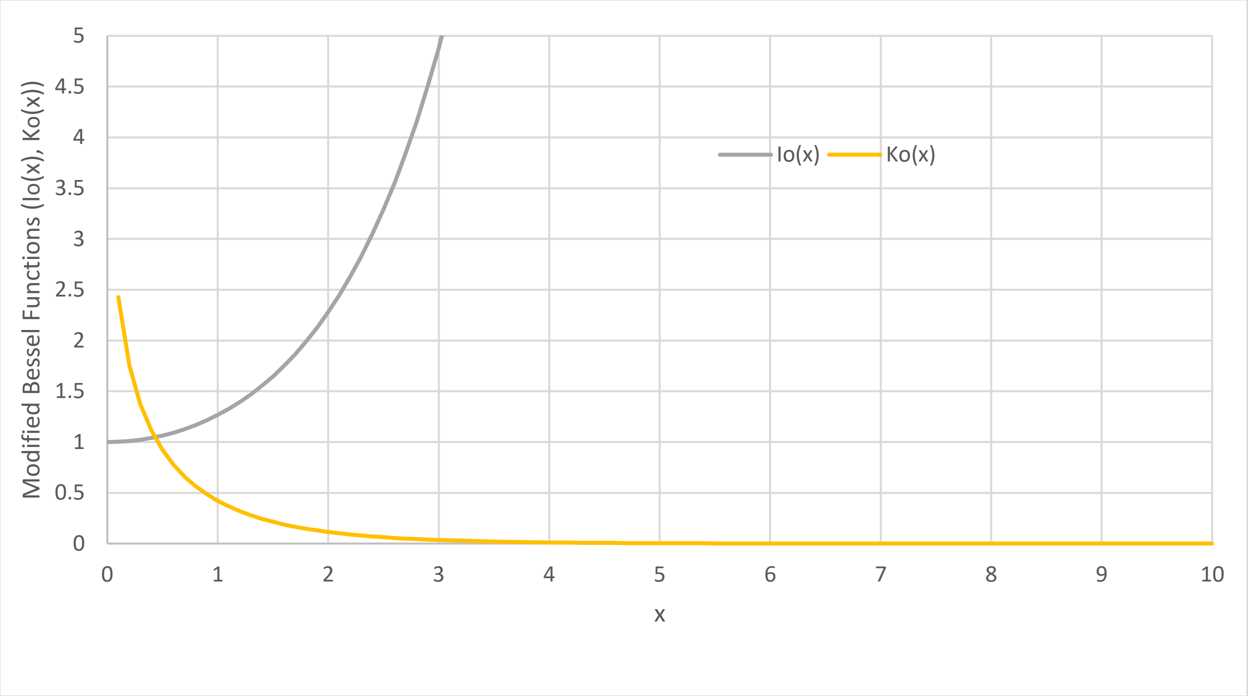


Fig. 9. Modified Bessel Functions of the First and Second Kind of Order Zero

Boundary conditions are based on the reflector neutron flux equaling zero at the extrapolated boundary. The core flux must still be finite along the centerline. Finally, the neutron flux and neutron current density based on the reflector and core flux equations must match at the boundary between the two. This is analogous to heat transfer problems where the temperatures and heat fluxes must match at the boundary.

$$\varphi_c(0,z) = \varphi_o, a\ finite\ value \tag{48}$$

$$\varphi_r(R_e,z) = 0 \tag{49}$$

$$\varphi_c(R,z) = \varphi_r(R,z) \tag{50}$$

$$J_c(R,z) = J_r(R,z) \tag{51}$$

The last boundary condition may be expressed in terms of derivatives of the neutron flux. The first boundary condition requires that $A_2 = 0$ since $Y_o(0)$ approaches negative infinity and the solution must be finite.

$$\varphi_c(0) = \varphi_o \tag{52}$$

$$\varphi_o = A_1\, \cancel{J_o(0)}^{1} + A_2\, \cancel{Y_o(0)}^{\infty} \tag{53}$$

$$\therefore A_2 = 0 \tag{54}$$

$$\varphi_c(r) = \varphi_o\, J_o(Br) \tag{55}$$

The second boundary condition is applied to the equation for the flux in the reflector and requires the flux to be zero at the extrapolated radius.

$$\varphi_r(R_e) = 0 \tag{56}$$

$$0 = A_3\, I_o\left(\frac{R_e}{L_r}\right) + A_4\, K_o\left(\frac{R_e}{L_r}\right) \tag{57}$$

$$\therefore\ \ A_4 = -A_3 \frac{I_o\left(\frac{R_o}{L_r}\right)}{K_o\left(\frac{R_o}{L_r}\right)} \tag{58}$$

$$\varphi_r(r) = A_3\left[I_o\left(\frac{r}{L_r}\right) - \frac{I_o\left(\frac{R_e}{L_r}\right)}{K_o\left(\frac{R_e}{L_r}\right)} K_o\left(\frac{r}{L_r}\right)\right] \tag{59}$$

The third boundary condition requires that the flux be the same at the boundary between the core and the reflector.

$$\varphi_c(R) = \varphi_r(R) \tag{60}$$

$$\varphi_o\, J_o(BR) = A_3\left[I_o\left(\frac{R}{L_r}\right) - \frac{I_o\left(\frac{R_e}{L_r}\right)}{K_o\left(\frac{R_e}{L_r}\right)} K_o\left(\frac{R}{L_r}\right)\right] \tag{61}$$

This fixes the value of the constant A_3.

$$A_3 = \frac{\varphi_o\, J_o(BR)}{\left[I_o\left(\frac{R}{L_r}\right) - \frac{I_o\left(\frac{R_e}{L_r}\right)}{K_o\left(\frac{R_e}{L_r}\right)} K_o\left(\frac{R}{L_r}\right) \right]} \tag{62}$$

Next, the fourth boundary condition can be applied. This leads to an eigen condition.

$$J_c(R) = J_r(R) \tag{63}$$

$$\varphi_c{}'(r) = -\varphi_o\, B\, J_o(Br) \tag{64}$$

$$\beta = \frac{I_o\left(\frac{R_e}{L_r}\right)}{K_o\left(\frac{R_e}{L_r}\right)} \tag{65}$$

$$\varphi_r{}'(r) = -\frac{A_3\, I_1\left(\frac{r}{L_r}\right)}{L_r} + \frac{\beta\, A_3}{L_r} K_1\left(\frac{r}{L_r}\right) \tag{66}$$

Using the definition of the neutron current density, the fourth boundary condition can be solved using the derivatives of the neutron flux in the core and in the reflector at the boundary where $r = R$.

$$-D_c\, \frac{\partial \varphi_c(R)}{\partial r} = -D_r\, \frac{\partial \varphi_r(R)}{\partial r} \tag{67}$$

$$-D_c\big(-\varphi_o B\, J_1(BR)\big) = -D_r\left[-\frac{A_3}{L_r} I_1\left(\frac{R}{L_r}\right) + \frac{\beta A_3}{L_r} K_1\left(\frac{R}{L_r}\right)\right] \tag{68}$$

This results in the eigen condition.

$$\frac{D_c L_r B}{D_r}\, \frac{J_1(BR)}{J_o(BR)} = \frac{I_1\left(\frac{R}{L_r}\right) - \frac{I_o\left(\frac{R}{L_r}\right)}{K_o\left(\frac{R}{L_r}\right)} K_1\left(\frac{R}{L_r}\right)}{I_o\left(\frac{R}{L_r}\right) - \frac{I_o\left(\frac{R}{L_r}\right)}{K_o\left(\frac{R}{L_r}\right)} K_o\left(\frac{R}{L_r}\right)} \tag{69}$$

The buckling can be expressed in terms of the diffusion length in the reflector, L_r, the extrapolated outer radius of the reflector, R_e, and the neutron diffusion coefficients in both the reflector and the core. This eigen condition for a radially reflected infinite cylindrical core can be compared to the simple eigen condition that was found for a bare infinite cylindrical core.

$$J_o(\lambda\, r) = \ 0 \tag{70}$$

The eigen condition in equation (70) must be solved numerically where the critical radius of the core, R, can be expressed as function of B, L_r, and R_e.

$$f(B.L_r.R_e) = \frac{D_c}{D_r} L_r\ B\ \frac{J_1(BR)}{J_o(BR)} - \frac{I_1\left(\frac{R}{L_r}\right) - \frac{I_o\left(\frac{R_r}{L_r}\right)}{K_o\left(\frac{R_r}{L_r}\right)} K_1\left(\frac{R}{L_r}\right)}{I_o\left(\frac{R}{L_r}\right) - \frac{I_o\left(\frac{R_r}{L_r}\right)}{K_o\left(\frac{R_r}{L_r}\right)} K_o\left(\frac{R}{L_r}\right)} = 0 \tag{71}$$

To summarize, the neutron flux in the core and the reflector are now known.

$$\varphi_c(r) = \varphi_o\, J_o(Br) \tag{72}$$

$$\varphi_r(r) = \varphi_o \frac{J_o(BR)}{I_o\left(\frac{R}{L_r}\right) - \frac{I_o\left(\frac{R_e}{L_r}\right)}{K_o\left(\frac{R_e}{L_r}\right)} K_o\left(\frac{R}{L_r}\right)} \left[I_o\left(\frac{r}{L_r}\right) - \frac{I_o\left(\frac{R_e}{L_r}\right)}{K_o\left(\frac{R_e}{L_r}\right)} K_o\left(\frac{r}{L_r}\right) \right] \tag{73}$$

The power generated by the core can be expressed in terms of the neutron flux at the center of the core, and the equations is identical to the case of a bare infinite cylindrical core.

$$P = \frac{2\pi\ R\ G\ L \sum_f \varphi_o}{B} J_1(BR) \tag{74}$$

2.1.4 Finite Bare Cylindrical Core

By analyzing the neutron diffusion equation in a finite cylindrical core, the equations that describe the neutron flux in both the radial and axial directions can be derived. The solution also provides an eigen condition that gives the critical height and diameter of the core. This analysis eliminates any neutron reflector to simplify the solution, so this is a bare core. The usual steady-state neutron diffusion equation can be applied.

$$\nabla^2\varphi + B^2\varphi = 0 \tag{75}$$

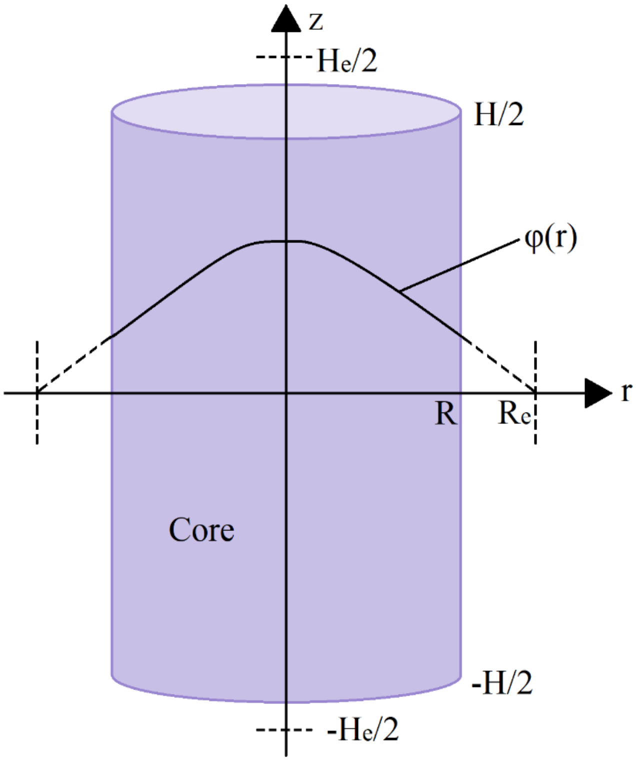


Fig. 10. Finite Cylindrical Core

For a two-dimensional, axisymmetric reactor in polar coordinates,

$$\frac{1}{r}\frac{\partial}{\partial r}\left(r\frac{\partial\varphi}{\partial r}\right) + \frac{\partial^2\varphi}{\partial z^2} + B^2\varphi = 0 \tag{76}$$

The boundary conditions for this problem are listed below.

$$\varphi(0,z) = finite \tag{77}$$

$$\left.\frac{\partial\varphi}{\partial z}\right]_{(r,0)} = 0 \tag{78}$$

$$\varphi(R_e, z) = 0 \tag{79}$$

$$\varphi\left(r, \frac{H_e}{2}\right) = 0 \tag{80}$$

This partial differential equation can be solved using separation of variables. The value of $\varphi(r,z)$ is replaced by two functions that are based on each independent variable as $\varphi(r,z) = R(r)\,\zeta(z)$. This is substituted into equation (76) to find that the partial differential equation can be separated into two separate ordinary differential equations equal to the same constant. The primes indicate derivatives of each function with respect to its independent variable.

$$\left(R'' + \frac{1}{r}\,R'\right)\zeta + R\,\zeta'' + B^2\,R\,\zeta = 0 \tag{81}$$

Divide both sides of this equation by $\zeta = R\,\zeta$.

$$\frac{R''}{R} + \frac{1}{r}\frac{R'}{R} + \frac{\zeta''}{\zeta} + B^2 = 0 \tag{82}$$

This equation is easily separable and since any function of only r that equals a function of only z can only be equal if they also equal zero or a constant. The constant assigned here is λ^2.

$$\frac{R''}{R} + \frac{1}{r}\frac{R'}{R} + +B^2 = -\frac{\zeta''}{\zeta} = +\lambda^2 \tag{83}$$

Separating the two ordinary differential equations:

$$\zeta'' = -\lambda^2\,\zeta \tag{84}$$

$$\frac{R''}{R} + \frac{1}{r}\frac{R'}{R} + B^2 - \lambda^2 = 0 \tag{85}$$

$$B^2 - \lambda^2 = \mu^2 \tag{86}$$

The general solutions for each differential equation are:

$$(z) = A_1 \cos(\lambda\, z) + A_2 \sin(\lambda\, z) \tag{87}$$

$$R(r) = A_3\, J_o(\mu\, r) + A_4\, Y_o(\mu\, r) \tag{88}$$

Rewriting the boundary conditions in terms of the separated variables:

$$R(0) = finite \tag{89}$$

$$\zeta'(0) = 0 \tag{90}$$

$$R(R_e) = 0 \tag{91}$$

$$\zeta\left(\frac{H_e}{2}\right) = 0 \tag{92}$$

By applying the first boundary condition,

$$R(0) = A_1\, J_o(0) + A_2\, Y_o(0) \tag{93}$$

$$\therefore\ A_2 = 0 \tag{94}$$

Next, apply the third boundary condition. This produces an eigen condition and eigenvalues.

$$R(R_e) = 0 = A_3\, J_o(\mu\, R_e) \tag{95}$$

$$\therefore\ A_3 = 0, \qquad trivial\ solution\ or, \tag{96}$$

$$\mu\, R_e = 2.4048, 5.5201, 8.6537, \dots. \tag{97}$$

Assume that only the fundamental solution survives over time.

$$\mu = \frac{2.405}{R_e} \tag{98}$$

The solution for $R(R_e)$ can now be shown.

$$R(r) = A_3\, J_o\left(\frac{2.405\, r}{R_e}\right) \tag{99}$$

Moving on to the second boundary condition.

$$\zeta'(0) = 0 \tag{100}$$

$$\zeta'(z) = -A_1\,\lambda\,sin(\lambda\,z) + A_2\,\lambda\,cos(\lambda\,z) \tag{101}$$

$$R(r) = A_3\,J_o\left(\frac{2.405\,r}{R_e}\right) \tag{102}$$

The fourth boundary condition leads to another eigen condition and eigenvalues.

$$\zeta\left(\frac{H_e}{2}\right) = 0 = A_1 \cos\left(\frac{\lambda\,H_e}{2}\right) \tag{103}$$

$$\therefore\ \frac{\lambda\,H_e}{2} = \frac{\pi}{2}, \frac{3\pi}{2}, \frac{5\pi}{2} \ldots \tag{104}$$

Assume, again, that only the fundamental solution survives over time.

$$\lambda = \frac{\pi}{H_e} \tag{105}$$

At the very center of the reactor core, $\varphi(0,0) = \varphi_{co}$. This leads to the conclusion that:

$$\varphi_{co} = A_1\,A_3 \tag{106}$$

The final steady-state solution can now be shown.

$$\varphi(r) = \varphi_{co}\,J_o\left(\frac{2.405\,r}{R_e}\right) \cos\left(\frac{\pi\,z}{H_e}\right) \tag{107}$$

The physical dimensions of the reactor necessary to make it critical ($k_{eff} = 1$) are given in terms of the buckling, which is a function of the material properties within the fuel and moderator.

$$B^2 = \left(\frac{2.405}{R_e}\right)^2 + \left(\frac{\pi}{H_e}\right)^2 \tag{108}$$

The last task is to obtain the relationship between the maximum neutron flux within the reactor core and the power generated by fission. Equation (109) provides the differential volume for a cylindrical reactor.

$$d\tau = 2\,\pi\,r\,dr\,dz \tag{109}$$

Next, the differential power produced by fission within this differential volume.

$$dP = G\, \Sigma_f\, \varphi(r,z)\, d\tau \tag{110}$$

Integrate over the geometric bounds of the reactor to find the total power produced by fission within the core.

$$P = G\, \Sigma_f \iiint \varphi(r,z)\ d\tau \tag{111}$$

$$P = G\, \Sigma_f \int_{z=-\frac{H_e}{2}}^{\frac{H_e}{2}} \varphi_{co} \cos\left(\frac{\pi\, z}{R_e}\right) dz \int_{r=0}^{R} J_o\left(\frac{2.405\, r}{R_e}\right) 2\, \pi\, r\, dr \tag{112}$$

$$P = \frac{4}{2.405}\, \varphi_{co}\, G\, \Sigma_f\, R_e\, H_e\, R \sin\left(\frac{\pi\, H}{2\, H_e}\right) J_o\left(\frac{2.405\, R}{R_e}\right) \tag{113}$$

This equation has one key use. The maximum neutron flux in the core, φ_{co}, can be represented in terms of the power level, P. For the Pewee reactor, the diameter of the core was 0.80 m and the height was 1.4 m. If the Pewee core did not have beryllium reflectors, the neutron flux based on equation (107) would appear as in Fig. 11.

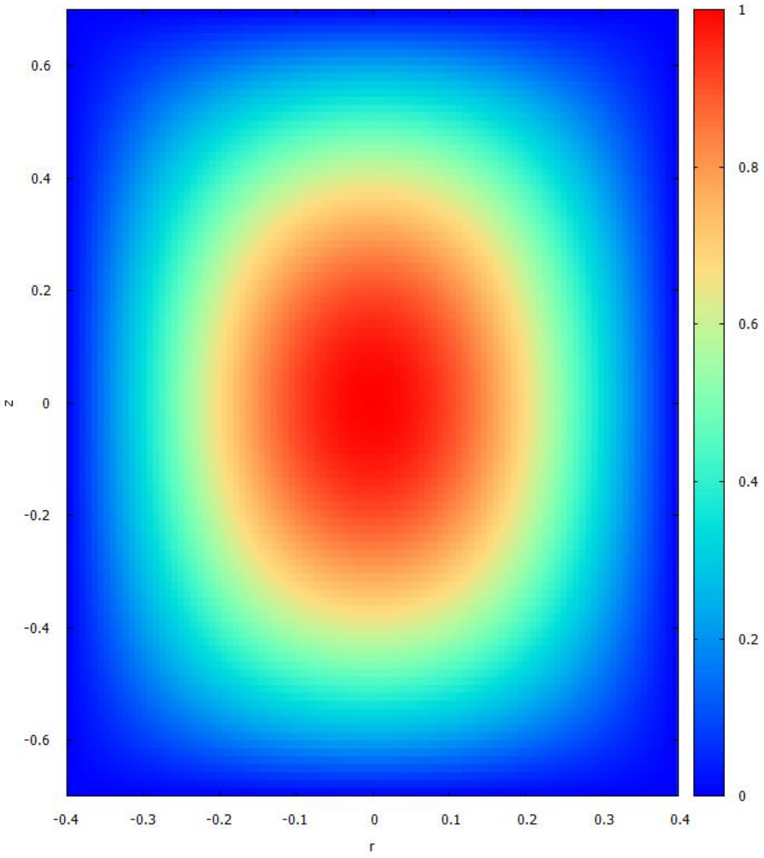


Fig. 11. Neutron Flux Profile in a Bare Finite Cylindrical Reactor

2.1.1.5 Axially Reflected Core

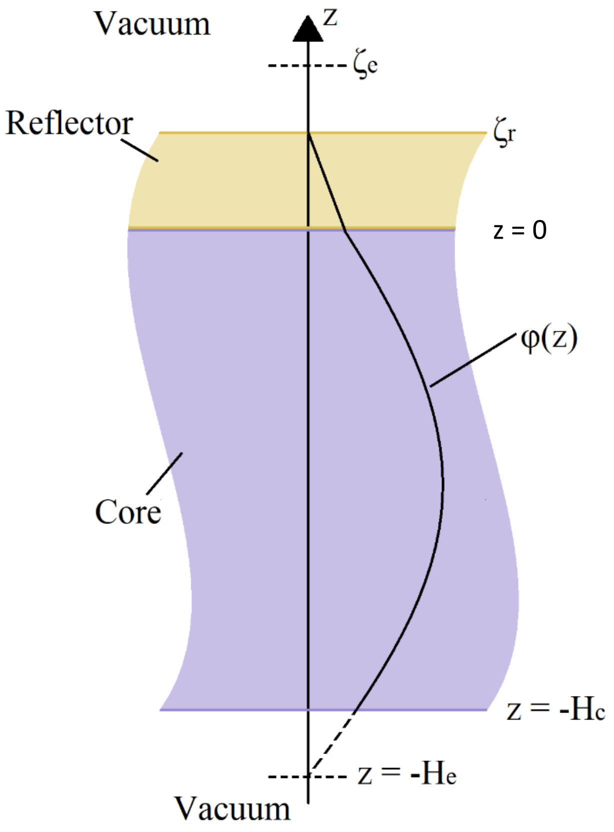


Fig. 12. Infinite Reactor with a Reflector and Core

The Pewee reactor had a beryllium reflector located axially around the core to reflect neutrons produced by fission back into the core to help sustain the chain reaction. The reflector also decreases the mass of fissionable uranium needed to make the reactor critical. It would be desirable to add a second reflector below the reactor, but the de Laval nozzle limits this possibility. The designers did include a beryllium reflector at the inlet (top) of the reactor to help prevent the leakage of neutrons. To model the effect of an axial reflector, the reflector geometry shown in

Fig. 12 with a reflector on top of a reactor that is exposed to a vacuum on the bottom can be used. The reflector (r) and the core (c) are represented by two separate differential equations.

$$\nabla^2 \varphi_c + B^2 \varphi_c = 0 \tag{114}$$

$$\nabla^2 \varphi_r - \frac{\varphi_r}{L_r^2} = 0 \tag{115}$$

The Laplacian term for one-dimensional problems in the z-direction is $\nabla^2 \varphi = \frac{\partial^2 \varphi}{\partial z^2}$. The general solutions are:

$$\varphi_c(z) = A_1 \cos(B\,z) + A_2 \sin(B\,z) \tag{116}$$

$$\varphi_r(z) = A_3\, e^{\frac{z}{L_r}} + A_4\, e^{-\frac{z}{L_r}} \tag{117}$$

The constants of integration are found using the following four boundary conditions.

$$\varphi_c(-H_e) = 0 \tag{118}$$

$$\varphi_r(\zeta_e) = 0 \tag{119}$$

$$\varphi_c(0) = \varphi_r(0) \tag{120}$$

$$J_c(0) = -D_c \frac{\partial \varphi_c}{\partial z}\bigg]_{z=0} = -D_r \frac{\partial \varphi_r}{\partial z}\bigg]_{z=0} = J_r(0) \tag{121}$$

By applying these four boundary conditions, the following eigen condition is found, which is a transcendental equation that must be solved numerically.

$$\frac{D_c}{D_r} L_r\, B \cot(B\,H_e) + coth\left(\frac{\zeta_e}{L_r}\right) = 0 \tag{122}$$

The eigenfunctions that describe the neutron flux in the core and in the reflector are given by:

$$\varphi_c(z) = \varphi_o(\cos(B\,z) + \cot(B\,H_e)\sin(B\,z)) \tag{123}$$

$$\varphi_r(z) = \varphi_o \frac{\sinh\left(\frac{\zeta_e - z}{L_r}\right)}{\sinh\left(\frac{\zeta_e}{L_r}\right)} \tag{124}$$

In this problem the reference neutron flux, φ_o, is defined at the boundary between the core and the reflector where z = 0. This value can be found from the power level of the reactor which must be defined as P'' (W/m2) since the reactor core is infinite in the x and y directions. The differential volume is $d\tau = A\ dz$ and the differential power per unit area is:

$$P'' = G\ \Sigma_f\ \varphi_c(z)\ dz \tag{125}$$

The power per unit area can be found through the following integral.

$$P'' = G\ \Sigma_f \int_{z=-H_e}^{0} \varphi_c(z)\ dz \tag{126}$$

$$P'' = G\ \Sigma_f \varphi_o \int_{z=\ -H_e}^{0} (\cos(B\ z) + \cot(B\ H_e)\sin(B\ z))\ dz \tag{127}$$

The final expression provides us with a relationship between the power per unit area in the core and the reference neutron flux, φ_o.

$$P'' = \frac{G\ \Sigma_f \varphi_o}{B}\left(\frac{1 - \cos(B\ H_e)}{\sin(B\ H_e)}\right) \tag{128}$$

By using the diffusion coefficients, thermal diffusion lengths, and buckling for the Pewee reactor, the neutron flux can be shown for a reactor with a reflector that is 90% efficient in reducing the critical size of the reactor. The neutron flux in the core and the reflector are shown in Fig. 13. The neutron flux peaks at the interface between the reflector and the reactor core. As the reflector is made thicker, the critical height of the reactor decreases until it asymptotically approaches a limit, as shown in Fig. 14.

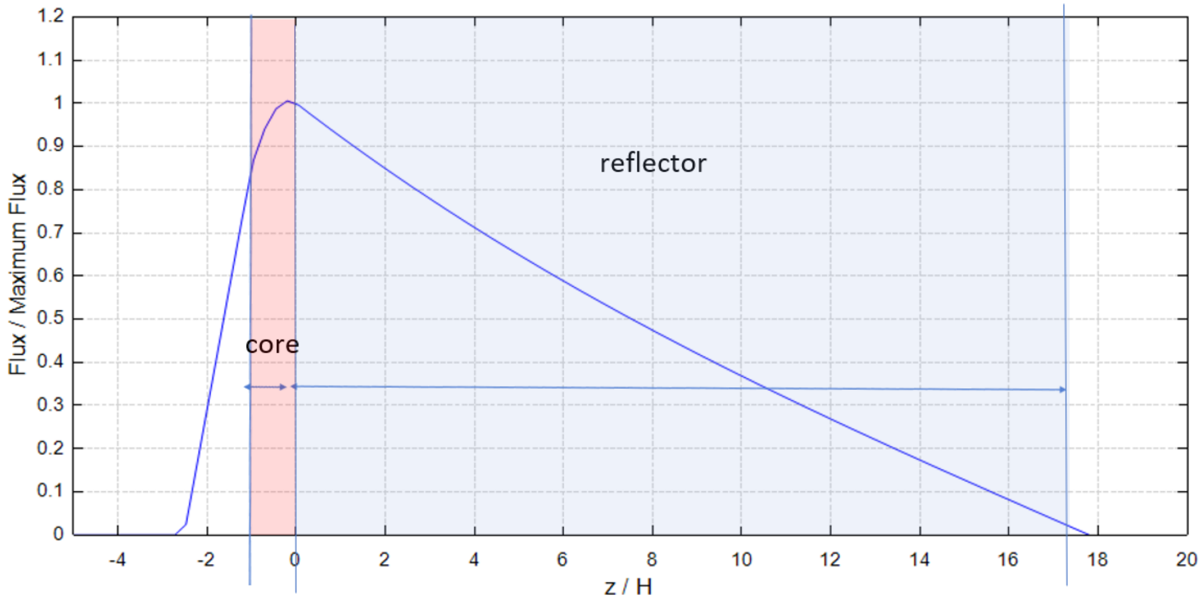


Fig. 13. Neutron Flux Distribution in an Axially Reflected Reactor

In Fig. 14, the normalized critical reactor height asymptotically approaches a value of H * B = 0.662. For comparison, the critical reactor height of a bare slab reactor is H * B = π/2 = 1.5708, so the reactor height can be reduced to 58% of its bare critical height by adding a infinitely thick reflector to the top of the reactor core. This is impractical, but 99% of this savings can be achieved by adding a reflector with a thickness of $\zeta = 1.22 * L_r$ and 90% savings can be reached with a reflector thickness of $\zeta = 0.443 * L_r$. The fuel mixture in the core is much denser than the reflector material, so a reflector can significantly decrease the size of the critical reactor and its mass.

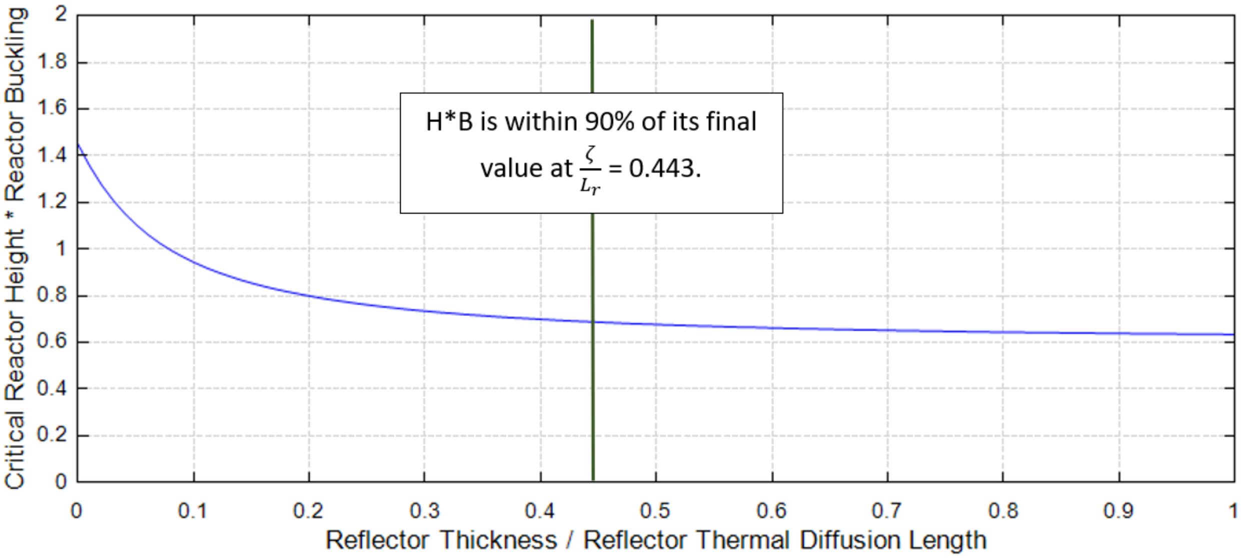


Fig. 14. Critical Reactor Height vs. Reflector Thickness

2.1.5 Reactor Core with Reflectors at Each End

This case is used in the section of this work that describes the mitigation of corrosion within an NTP reactor. The geometry uses two reflectors of different thicknesses on the top and bottom of the reactor core as shown in Fig. 15. The analysis proceeds in the same manner as in the previous sections and only the equations that are relevant to the analysis of corrosion mitigation are provided.

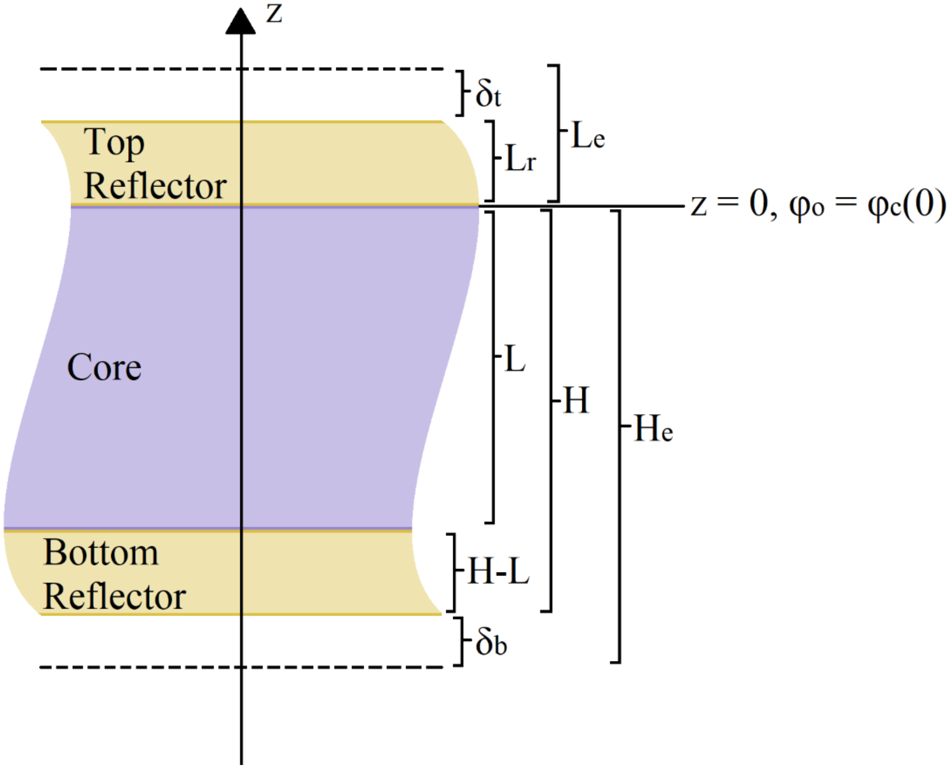


Fig. 15. Infinite One-Dimensional Reactor with Different Neutron Reflectors at Each End

The neutron flux in the top reflector, the core, and in the bottom reflector are given by equations (129), (130), and (131), respectively. The thermal diffusion lengths in the top and bottom reflectors are defined as L_t and L_b. The buckling in the core is defined as B_c and this value can be found from the eigen condition in equation (132) if the length of the core, L, is defined. The reference value for the reactor flux is $\varphi_o = \varphi(0)$.

$$\varphi_t(z) = \varphi_o \left(cosh\left(\frac{z}{L_t}\right) + coth\left(\frac{L_e}{L_t}\right) sinh\left(\frac{z}{L_t}\right)\right) \tag{129}$$

$$\varphi_c(z) = \varphi_o \left(cos(B_c\ z) + \frac{D_t}{D_c\ B_c\ L_t} coth\left(\frac{L_e}{L_t}\right)\ sin(B_c\ z) \right) \tag{130}$$

$$\varphi_b(z) = \varphi_o \left[\frac{cos(B_c\ L) + \frac{D_t}{D_c\ B_c\ L_t} coth\left(\frac{L_e}{L_t}\right)\ sin(B_c\ L)}{cosh\left(\frac{L}{L_b}\right) - coth\left(\frac{H_e}{L_b}\right)\ sinh\left(\frac{L}{L_b}\right)} \right] \left[cosh\left(\frac{z}{L_b}\right) - coth\left(\frac{H_e}{L_b}\right)\ sinh\left(\frac{z}{L_b}\right) \right] \tag{131}$$

$$f(B_c, L) = \left[\frac{tanh\left(\frac{L}{L_b}\right) \tanh\left(\frac{H_e}{L_b}\right) - 1}{tanh\left(\frac{H_e}{L_b}\right) - \tanh\left(\frac{L}{L_b}\right)} \right] \left[\frac{tanh\left(\frac{L_e}{L_t}\right) + \frac{D_t}{D_c\ B_c\ L_t}\ tanh(B_c\ L)}{\frac{D_c\ B_c\ L_b}{D_b}\ tan(B_c\ L) \tanh\left(\frac{L_e}{L_t}\right) - \frac{D_t\ L_b}{D_c\ L_t}} \right] + 1 = 0 \tag{132}$$

An example of the flux distribution is shown in Fig. 16. The dimensions of the reflectors and the core used to generate the graph are L_r = 0.2, L = 1.0, H = 1.3.

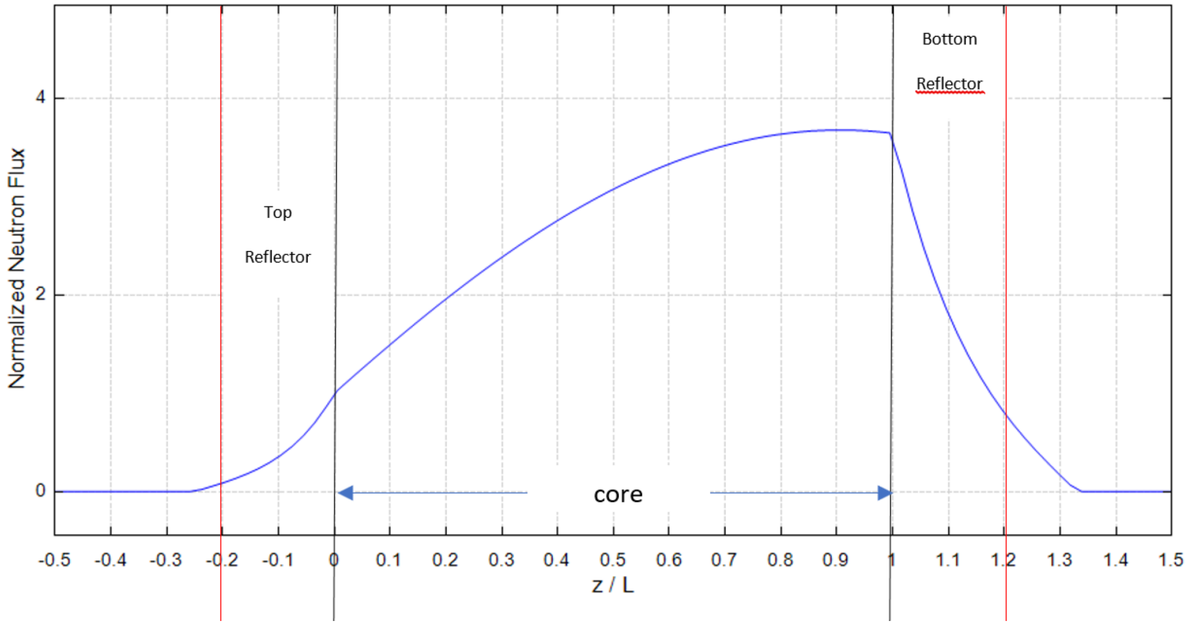


Fig. 16. Example of a Reactor Core with Different Neutron Reflectors at Each End

By differentiating the flux in the reactor core with respect to z, the position where the flux reaches a maximum can be found.

$$z_{max} = \frac{1}{B_c}\, atan\left(\frac{D_t}{D_c\, B_c\, L_t}\, coth\left(\frac{L_e}{L_t}\right)\right) \tag{133}$$

The power generated by the core can also be related to φ_o by integrating the energy produced by fission throughout the volume of the core. Since this reactor is infinite in the x and y directions, *P''* represents the power produced per unit area of the core.

$$P'' = G\,\Sigma_f\,\frac{\varphi_o}{B_c}\left(sin(B_c\,L) + \frac{D_t}{D_c\,B_c\,L_t}\,coth\left(\frac{L_e}{L_t}\right)\left(1 - cos(B_c\,L)\right)\right) \qquad (134)$$

2.2 Thermodynamic Model of the Rocket Engine

The next section summarizes the path that the hydrogen propellant takes as it is moved from its storage tank and out to the exit of the nozzle. The equations used to describe flow through the de Laval nozzle were describe previously in the section on compressible flow.

2.2.1 Heat Transfer and Fluid Flow in a Nuclear Rocket

The propellant in a nuclear rocket follows a complicated path. It starts in the liquid hydrogen storage tank at a temperature of about 20 K and is forced by the compressor section of a turbopump into tubing that surrounds the throat and diverging section of the Laval nozzle. This technique was developed by Wernher Von Braun for the German V-2 rocket and it serves to cool the nozzle and keeps the metal walls from melting. The cold hydrogen gas is then circulated through the outer neutron reflector and control drums surrounding the reactor reaching a temperature of about 120 K. From the top of the reactor assembly, hydrogen is routed through a turbine to drive the compressor and a small amount is used to heat and pressurize the liquid hydrogen tank. The major fraction of the hydrogen gas is forced down through the reactor core to be heated and ejected through the de Laval nozzle to produce thrust.

2.2.2 Flow from the Propellant Tank to the Reactor Inlet

An NTP rocket engine relies upon the nuclear reactor to generate heat that is transferred to the hydrogen propellant to generate thrust. After leaving the reactor, the hydrogen is at a pressure that is sufficient to produce supersonic flow, and a de Laval nozzle is used to ensure that the thrust is maximized.

The path that the hydrogen propellant takes through the rocket system is shown in Fig. 17. Hydrogen is stored as a liquid in large storage tanks [1] below its boiling point of 20.271 K. The liquid hydrogen is pumped [2] into a set of pipes that encircle the upper part of the diverging nozzle and the throat of the de Laval nozzle [5] to remove heat to prevent the metal walls of the nozzle from melting. Next, the heated hydrogen gas/liquid passes through the outer neutron reflector and the control drums [4] that surround the reactor, also to prevent the components from melting.

Some hydrogen passing through the reflector is directed to the turbine that provides power to the pump [9] by a valve. The hydrogen passing through the valve is directed into the reactor by a second valve [13] and a small amount of this hydrogen gas is diverted into the liquid hydrogen tank to provide a positive pressure within the tank [11].

The main hydrogen flow through the turbopump and the majority of the flow passing through the reflector then flows into the reactor inlet where it serves as a neutron moderator, necessary to maintain nuclear criticality within the reactor, and the flow also picks up heat from the reactor by passing through a series of coolant channels. In the next section, the flow of hydrogen through the reactor will be described.

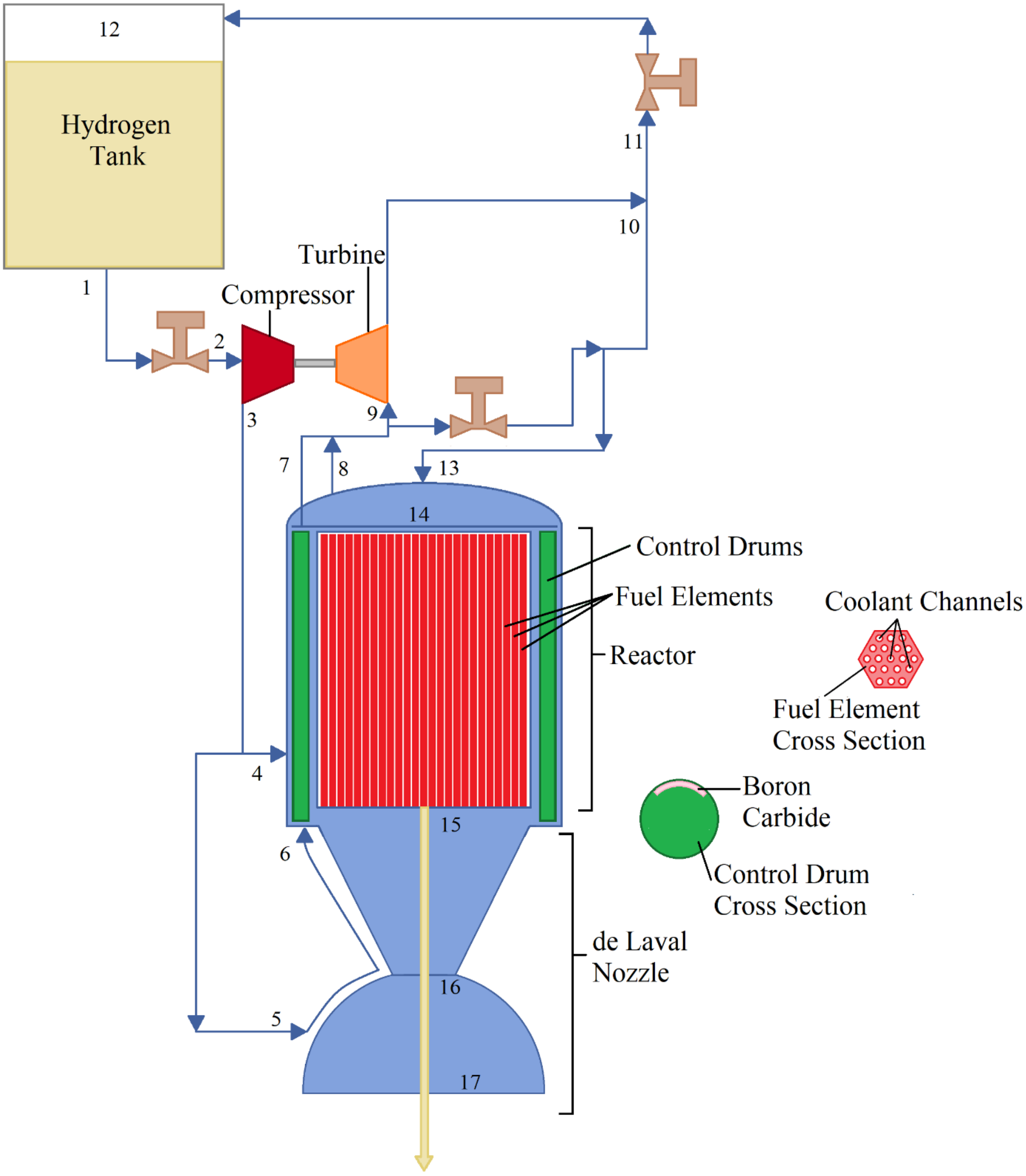


Fig. 17. Reactor Simplified Reactor Geometry and Process

2.2.3 Reactor Geometry and Inlet Conditions

Typical thermodynamic values for the hydrogen flowing into the inlet of the reactor are available for the NERVA/Pewee design:

p_{inlet} = 6.895 MPa = 1000 psia

T_{inlet} = 120 K

P_{outlet} = 4.14 MPa = 600 psia

T_{outlet} = 2700 K

$\dot{m}$ = 18 kg/s

Thrust = $1.11 * 10^5$ N = 25,000 lb.

I_{sp} = 900 s

2.3 Frictionless Duct Flow with Heat Transfer

NTP rocket engines are designed so that hydrogen gas can pass through small channels built into the reactor fuel elements. The channels are typically cylindrical in shape although spiral channels have been used by Russian designers. The hydrogen propellant enters the reactor at subsonic speeds and at a temperature of about 120 K. By convective heat transfer, the hydrogen picks up heat generated by the reactor and exits with a temperature of about 2700 K. The compressibility of the gas affects the Mach number of the flow as it exits the coolant channels in the reactor.

To model the flow of propellant through the reactor, it is helpful to review compressible flow through constant diameter ducts with heat transfer. To obtain closed-form solutions, the heat generated by friction within the duct or channel is ignored. Inlet and outlet properties for a compressible flow through a constant area channel are shown in Fig. 18, and equations can be developed by using the three conservation equations: continuity, conservation of momentum, and conservation of energy.

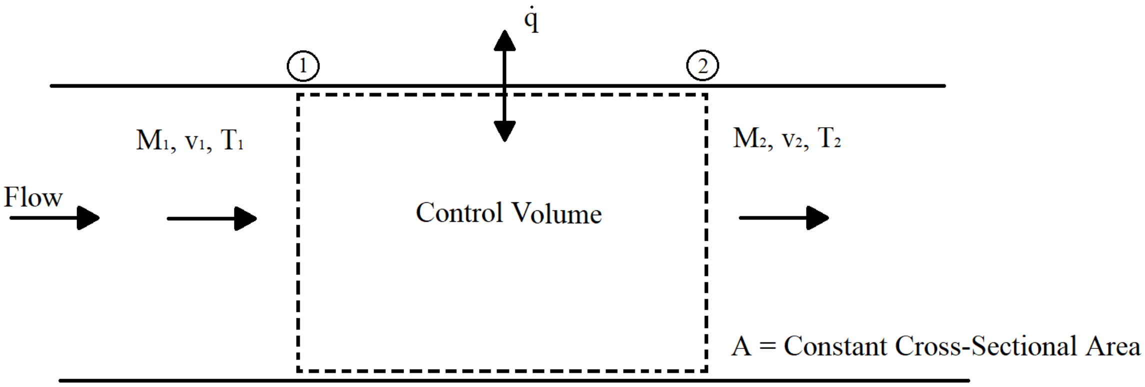


Fig. 18. Control Volume Inlet and Exit Properties

2.3.1 Continuity

Conservation of mass is based on constant mass flowrate through the nozzle.

$$\dot{m} = \rho_1 A V_1 = \rho_2 A V_2 \tag{135}$$

Since cross-sectional area is constant, the mass velocity, G, is constant throughout the control volume.

$$G_1 = \rho_2 V_1 = \rho_2 V_2 = G_2 = G \tag{136}$$

2.3.2 Conservation of Momentum

The inflow and outflow of momentum through the control surface that encloses the control volume is shown in Fig. 19 and, ignoring the effects of friction at the walls, the product of pressure and area lead to the only forces acting on the fluid within the control volume. Conservation of momentum requires that:

$$\sum \vec{F} = \frac{\partial}{\partial\, t} \iiint\limits_{control\ volume} \rho\, \vec{V}\, d\tau + \oiint\limits_{control\ surface} \rho\, \vec{V}\, \vec{V} \cdot d\vec{S} \tag{137}$$

Since all flow moves to the right, this equation reduces to the following equation.

$$p_1\, A - p_2\, A = \dot{m}\, (V_2 - V_1) \tag{138}$$

Conservation of momentum reduces down to the specific impulse, I, within the flow remaining constant.

$$p_1 + \dot{m}\, V_1 = p_2 + \dot{m}\, V_2 \tag{139}$$

$$p_1 + \rho_1\, V_1^2 = p_2 + \rho_2\, V_2^2 \tag{140}$$

$$p_1 + \frac{G^2}{\rho_1} = p_2 + \frac{G^2}{\rho_2} \tag{141}$$

$$I = impulse\ pressure = p + \frac{G^2}{\rho} = constant \tag{142}$$

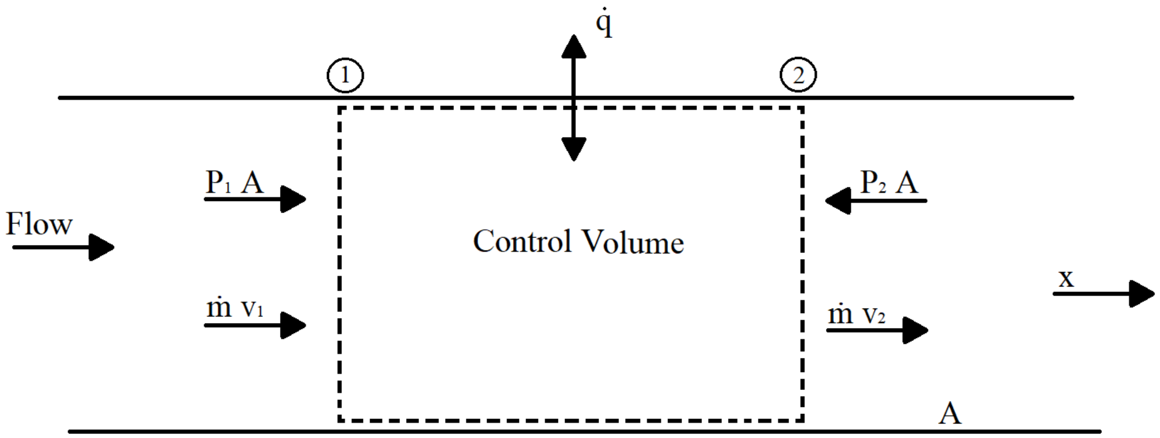


Fig. 19. Conservation of Momentum for a Constant Duct Area

2.3.3 Conservation of Energy

The remaining conservation principle is based on the concept that all the energy within the flow at point 1 plus any heat added or removed from the fluid must equal the energy at point 2, if the effects of friction are ignored.

$$\dot{q} \; + \frac{p_1}{\rho_1} + \frac{\dot{V}_1^2}{2} + \; e_1 \; = \frac{p_2}{\rho_2} + \frac{V_2^2}{2} + \; e_2 \qquad (143)$$

By noting that enthalpy, h = e + p/ρ, this equation reduces to:

$$\dot{q} \; = \; h_2 \; - \; h_1 \; + \frac{V_2^2}{2} - \frac{V_1^2}{2} \qquad (144)$$

$$\dot{q} \; = \; c_p \; (T_2 \; - \; T_1) \; + \frac{V_2^2}{2} - \frac{V_1^2}{2} \qquad (145)$$

$$\dot{q} \; = \left(c_p T_2 \; + \frac{V_2^2}{2} \right) - \left(c_p T_1 \; + \frac{V_1^2}{2} \right) \qquad (146)$$

$$\dot{q} \; = \; c_p \; (T_{o2} \; - \; T_{o1}) \qquad (147)$$

This final equation shows that the stagnation temperature will increase within the flow if heat is added. Flow within a constant area duct with heat addition or removal is also limited by M = 1 conditions. If heat is added to a subsonic flow, the Mach number cannot exceed the choked value of M = 1, although the stagnation temperature can continue to rise. Heating a flow that is initially supersonic will also result in choked flow with M = 1. Other properties can be computed based on the conservation laws.

$$\frac{T_2}{T_1} = \left(\frac{M_2}{M_1} \frac{1 \; + \; k \, M_1^2}{1 \; + \; k \, M_2^2} \right)^2 \qquad (148)$$

$$\frac{V_2}{V_1} \; = \; \frac{M_2}{M_1} \sqrt{\frac{T_2}{T_1}} \qquad (149)$$

$$\frac{\rho_2}{\rho_1} = \frac{V_2}{V_1} \tag{150}$$

$$\frac{T_{o2}}{T_{o1}} = \left(\frac{M_2}{M_1}\frac{1 + k\,M_1^2}{1 + k\,M_2^2}\right)^2 \frac{2 + (k - 1)M_2^2}{2 + (k - 1)M_1^2} \tag{151}$$

If the amount of heat added to the flow is known, then equation (147) can be used to calculate the stagnation temperature at the end of the heating section and equation (151) can be used to compute the Mach number at point 2. All other properties can then be computed from the upstream properties and the Mach number.

2.4 Todreas Method for Simulation of Heat Transfer in a Coolant Channel

The geometry of the coolant channels in Pewee-type reactors is shown in Fig. 20. The fuel material is hexagonal in shape and coolant holes are passed through the fuel matrix.

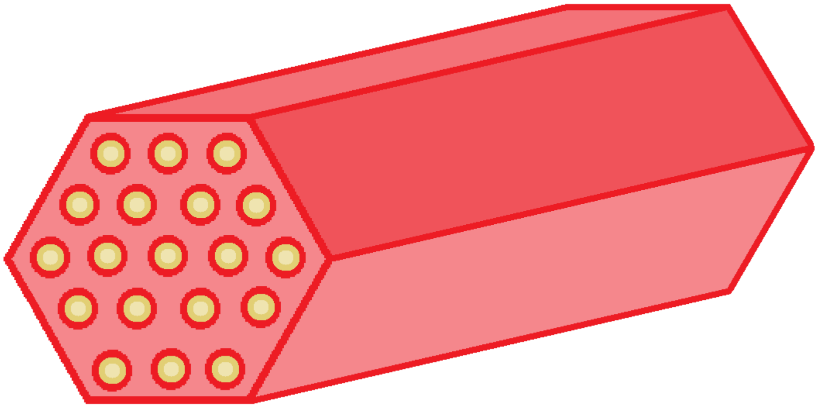


Fig. 20. Typical Fuel Element for a Pewee Type Reactor

To quantify the heat transfer from the fuel into the hydrogen propellant, a simplified model is used based on the work by Todreas and Kazimi [Labib]. Flow through circular channels in the fuel with a pitch of P and a coolant channel radius of R_c is replaced by an equivalent annular geometry with an inner radius of R_{inner} and an outer radius of R_{fuel} for the fuel where the equivalent fuel extends from$R_{inner} < r < R_{fuel}$.

The coolant channel area is:

$$A_{channel} = \pi {R_c}^2 \tag{152}$$

The total unit cell area is:

$$A_{total(equivalent)} = \frac{\sqrt{3}\, P^2}{2} \tag{153}$$

The equivalent fuel area is:

$$A_{fuel(equivalent)} = \left(A_{total(equivalent)} - A_{channel}\right) \tag{154}$$

and the equivalent diameter of the fuel is:

$$D_{fuel(equivalent)} = \sqrt{\frac{A_{fuel(equilavent)}}{\pi} + {R_c}^2} \tag{155}$$

2.5 Compressible Flow Theory

NTP rockets utilize a nuclear reactor to generate energy by nuclear fission and radioactive decay to heat hydrogen propellant. The hydrogen passes through a conventional de Laval nozzle to produce thrust. The improvements to current NTP technology that are discussed in future chapters is highly dependent upon the performance of a de Laval nozzle as described using compressible flow theory. The analysis is started by using isentropic flow theory to analyze pressure, temperature, and Mach number changes within the nozzle.

2.5.1 Ideal Gas Assumption

Compressible flow theory is based on the flow of an ideal gas. The ideal gas law serves as the equation of state and the properties of the gas are described in terms of the specific heat at constant pressure, c_p, and the specific heat at constant volume, c_v.

$$p\,v\ = R_{gas}\,T \tag{156}$$

$$R_{gas}\ = \frac{R_{universal}}{\widehat{M}} \tag{157}$$

$$c_p\ = \left(\frac{\partial\, h}{\partial\, T}\right)_p \tag{158}$$

$$c_v\ = \left(\frac{\partial\, e}{\partial\, T}\right)_v \tag{159}$$

$$k = \frac{c_p}{c_v} \tag{160}$$

Specific enthalpy, h, and the specific internal energy, e, may both be expressed as integrals of the specific heats and c_p, c_v, and k, the ratio of specific heats, are fairly constant over a wide range of temperatures. The ratio of specific heats of an ideal gas is related to the number of degrees of freedom for the atom or molecule making up the gas. For monatomic gases, such as helium, argon, and krypton, the ratio of specific heats is 5/3. For diatomic gases, including air, N_2, O_2, and H_2, the ratio of specific heats remains near 1.4. For polyatomic gases, including CO_2, the ratio can drop to 1.3. Although they remain constant over a wide range of temperatures and pressures, the specific heats do very over the range of temperatures experienced by hydrogen propellant in a nuclear thermal propulsion rocket.

The lowest molecular weight for any gas is diatomic hydrogen with an atomic weight of 2.0159 g/mol. This results in a gas constant of 4124 J/kg-K. This low atomic weight influences the choice of hydrogen as a propellant for NTP rockets, as discussed in Chapter 5.

2.5.2 Flow Through a de Laval Nozzle

Fig. 21 demonstrates the geometry of a de Laval nozzle designed to produce supersonic flow at the exit. The flow enters from a large reservoir or tank. The reservoir is considered to be so large that the velocity of flow within the tank is essentially zero. The temperature and pressure reflecting the stagnant gas within the reservoir are T_o and p_o, referred to as the stagnation or total temperature and pressure. For flow to move from the reservoir to the ambient fluid, the pressure within the reservoir must exceed the ambient pressure, or $p_o > p_\infty$.

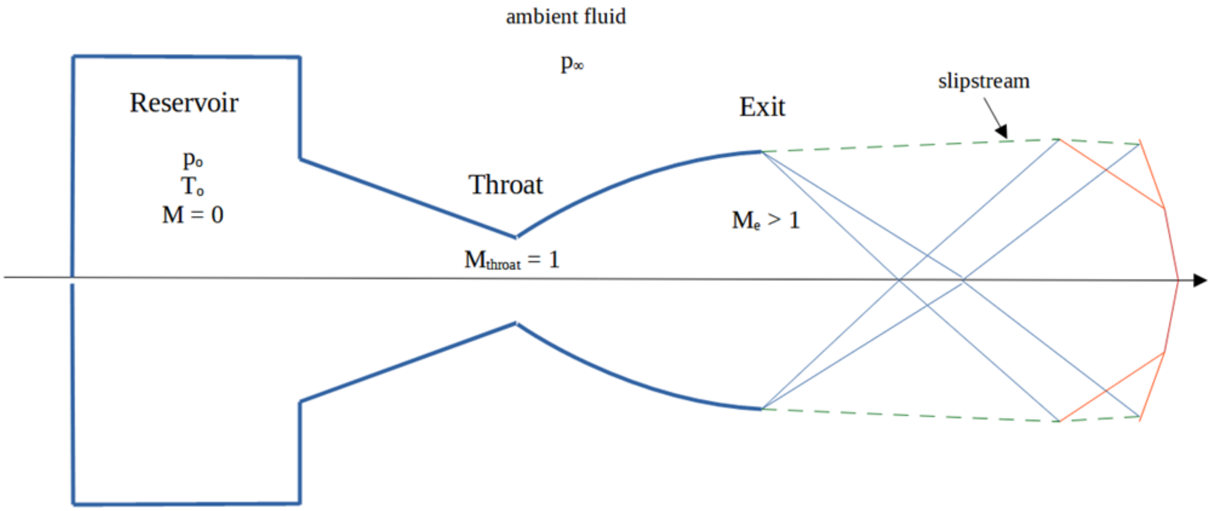


Fig. 21. Design of a de Laval Nozzle

As the flow leaves the reservoir, it flows through a converging section of the nozzle where its velocity and Mach number increase. The Mach number, *M*, is the dimensionless velocity based on the speed of sound in the air, a.

$$M = \frac{V}{a} = \frac{V}{\sqrt{k\, R_{gas}\, T}} \tag{161}$$

The flow through the converging section, the flow remains subsonic (M < 1). As fluid flows through the nozzle, the mass flowrate remains constant:

$$\dot{m} = \rho\, A\, V = constant \tag{162}$$

Since the mass flowrate does not change, a simple relationship exists between the cross-sectional area at any point in the nozzle, the velocity, and the density.

$$\frac{d\,\dot{m}}{\dot{m}} = \frac{d\rho}{\rho} + \frac{dA}{A} + \frac{dV}{V} = 0 \tag{163}$$

This equation can be used to solve for the change in area as a function of the Mach number.

$$\frac{dV}{V} = -\frac{dA}{A}\left(\frac{1}{1 - M^2}\right) \tag{164}$$

Equation (164) shows that, to accelerate a subsonic flow ($^{dV}/_{V} > 0$), the cross-sectional area must decrease ($^{dA}/_{A} < 0$), so a converging nozzle must be used. Once the Mach number reaches 1, the velocity can only be increased if the nozzle is diverging, ($^{dA}/_{A} > 0$). This is shown in Fig. 22 with the converging and diverging sections of the nozzle connected at the throat where the flow reaches M* = 1. The star nomenclature indicates choked flow where the fluid cannot accelerate without a diverging section. Equation (162) can also be expressed in terms of stagnation pressure and temperature, the cross-sectional area, and the Mach number.

$$\dot{m} = \frac{p_o\, A\, M}{\sqrt{T_o}} \sqrt{\frac{k}{R_{gas}}} \left(1 + \frac{k - 1}{2} M^2\right)^{-\frac{k+1}{2\,(k-1)}} \tag{165}$$

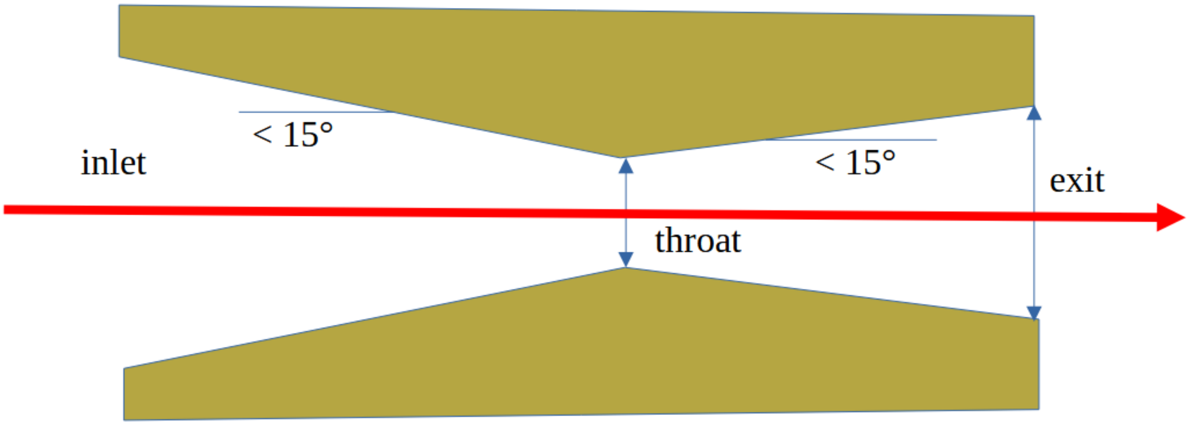


Fig. 22. Throat Area and Exit Area

Mass flowrate is constant, so this value can be referenced to the throat where the flow chokes to M* = 1.

$$\dot{m}_{max} = \frac{p_o\, A^*}{\sqrt{T_o}} \sqrt{\frac{k}{R_{gas}}} \left(\frac{2}{k + 1}\right)^{-\frac{k+1}{2\,(k-1)}} \tag{166}$$

Equation (166) shows that the mass flowrate through the nozzle is fixed by the stagnation temperature and pressure and by the cross-sectional area of the throat. Rocket engines typically operate with fixed throat diameter, so the mass flowrate is constant and the resulting thrust of the rocket is also fixed. Another important equation can be derived from equations (162) and (165).

This equation provides the cross-sectional area within the duct where the Mach number reaches a specific value. The area is presented as a ratio using the area of the nozzle at the throat.

$$\frac{A}{A^*} = \frac{1}{M}\left(\frac{2}{k+1}\left(1 + \left(\frac{k-1}{2}\right) M^2\right)\right)^{\frac{k+1}{2(k-1)}} \tag{167}$$

Equation (167) can be used to determine the exit area of the nozzle with respect to the throat area based on the desired exit Mach number. The ability of the flow to reach supersonic conditions is also dependent upon the ratio of the stagnation pressure and the ambient pressure. To derive the relationship between ${}^{p_o}\!/_{p_\infty}$ and M, the thermodynamic process that occurs as gas is expanded from the reservoir through the nozzle and into the ambient fluid, or into the vacuum of space must be known.

2.5.3 Isentropic Flow for a de Laval

The flow of a compressible gas through a de Laval nozzle is subject to the first and second laws of thermodynamics.

$$\delta w \,+\, \delta q \,=\, de \tag{168}$$

$$ds \,\geq\, \frac{dq}{T} \tag{169}$$

If a process is reversible, then equation (169) becomes an equality. For this case, equations (168) and (169) can be combined to form a version of Gibb's equation.

$$T\,ds \,=\, p\,dv \,+\, de \,=\, -\,R\,T\,\frac{d\rho}{\rho} \,+\, c_v\,dT \tag{170}$$

For the specific case where a flow is both reversible and adiabatic, ($dq = 0$), the change in entropy is zero and equation (170) can be used to define the process in terms of density, pressure, and/or temperature.

$$p/_{\rho^k} = constant \tag{171}$$

Conservation of energy can be applied between the reservoir and at any point within the nozzle. The energy equation can be represented in terms of the specific enthalpy.

$$h_o = h + \frac{V^2}{2} \tag{172}$$

By using the definition of c_p, this equation results in a relationship between the temperature, stagnation temperature, and the Mach number. The resulting equation is valid for adiabatic flow.

$$\frac{T_o}{T} = 1 + \frac{k - 1}{2} M^2 \tag{173}$$

By using equation (173) and assuming isentropic flow, similar equations can be derived for pressure and density.

$$\frac{p_o}{p} = \left(1 + \frac{k - 1}{2} M^2\right)^{\frac{k}{k-1}} \tag{174}$$

$$\frac{\rho_o}{\rho} = \left(1 + \frac{k - 1}{2} M^2\right)^{\frac{1}{k-1}} \tag{175}$$

In isentropic flow, the stagnation pressure and temperature stay constant and can be used as reference values. The minimum pressure ratio between the reservoir and the ambient fluid where flow can reach choked conditions at the throat is given by the following equation.

$$\frac{p_o}{p_\infty} \geq \left(\frac{k + 1}{2}\right)^{\frac{k}{k-1}} \tag{176}$$

For hydrogen, a diatomic gas, *k* = 1.4 and the pressure ratio necessary to cause choked flow is 1.8929. If the pressure ratio exceeds this value, then the flow in the diverging portion of the de Laval nozzle can become supersonic. In the vacuum of space, de Laval rocket nozzles area capable of supersonic flow for any stagnation pressure above vacuum, but the theoretical exit area of the nozzle, according to equation (167), becomes infinitely large. For practical purposes, nozzle area ratio, $^{A_{exit}}/_{A^*}$ is on the order of 300:1 so that the physical size of the diverging nozzle can be launched into space.

Since NTP rocket engines rely upon de Laval nozzles to produce thrust, the equations derived in this section will be useful in analyzing the changes in pressure, temperature, and Mach number. These equations are used in the thermodynamic analysis of the NTP engine documented in Chapter 5.

2.6 Friction Factor Pressure Drop

The flow of hydrogen through the coolant channels in the reactor is highly turbulent with a surface roughness that can range from smooth for new fuel to roughened for used fuel with corroded or fractured coating. To model the coolant channels, the pressure drop is represented by the Darcy-Weisbach friction factor, *f*.

$$\Delta p = f\ \frac{L}{D}\frac{V^2}{2} \tag{177}$$

For turbulent flow in smooth pipes, the friction factor can be represented by the Blasius equation in terms of the Reynolds number of the flow

$$f = \frac{0.316}{Re_D^{\frac{1}{4}}} \tag{178}$$

$$Re_D = \frac{V\ D}{\nu} \tag{179}$$

The friction factor for roughened pipes with surface roughness ε can be found from the Colebrook equation. For the reactor coolant channels analyzed in this study, the flow is always assumed to be turbulent with smooth walls.

$$\frac{1}{\sqrt{f}} = -2\log\left(\frac{2.51}{Re\sqrt{f}}\ \frac{\varepsilon}{3.71\ D}\right) \tag{180}$$

2.7 Optimization Techniques

Optimization techniques are often based on gradients determined using Newton's method to find the minima or maxima of some objective function under investigation. These methods work well if the variables, $\vec{r}$, that describe the objective function, $f(\vec{r})$, are known and are differentiable. Gradient-based optimization works well on convex problems where the objective function reaches a single minimum that can be found using gradients of the variables.

Another category of optimization can be employed when differentiation is not possible and yet optimization of the variables to maximize the objective function is desired. These may also be problems that are not convex where the objective function can reach multiple minima. This requires the use of global minimization methods. Problems that require this second category of optimization techniques may be based on a mechanical or natural system where variables affecting the system can be measured and the performance optimized. Complex mathematical systems can also be optimized using these methods since the computation of gradients may be difficult and the systems may not be convex due to multiple minima in the objective function.

The SIMPLEX algorithm [Dantzig] introduced by George Dantzig in 1947 can be used to optimize complex functions and utilizes three initial guesses for the variables that affect the objective function: $\vec{r}_1$, $\vec{r}_2$, and $\vec{r}_3$. The objective function is calculated at each point and, if the function is to be minimized, then the largest objective function is used to generate a new estimate by pivoting away from the worst value across a line connecting the two better values. This operation is repeated until a minimum is reached. By choosing different initial guesses, false minima can be eliminated, and global optimization can be achieved.

Genetic optimization, a subset of evolutionary optimization, is another method that can be used for global optimization. Genetic algorithms mimic the process of natural selection to find an optimal solution. These algorithms use mutations of solutions that show promise, crossover, or selection to produce more optimal solutions based on previous solutions that appear successful. The specific method used for this project is a version of a genetic algorithm named differential evolution (DE) optimization [Storn].

2.7.1 Differential Evolution Algorithm

This method was designed by Storn and Price and first reported in 1996. This algorithm does not require that variables be differentiable, nor does it require variables and the objective function to be continuous or be constant over time. DE can also be used to continuously update the optimal solution while the system, and its response to the input variables, is changing with time. This allows the system to be treated as a black box where the detailed mathematical functions that govern the system are not known.

The DE algorithm seeks the vector of variables m within the allowable space of all possible vectors, $\vec{r}$, that minimize the objective function so that $f(\vec{m}) \leq f(\overrightarrow{r})$. Each solution vector, r, contains n variables. The optimal solution vector m would represent the position of the global minimum. To seek the optimal solution vector, candidate solution vectors, called agents, are represented by x.

Several new variables need to be introduced. The DE method is very sensitive to the choice of these variables and recommended values are included.

NP Population size. This is the number of agents in each population to be considered. $NP \geq 4$ and typical values are $CR = 10\ n$, where n is the number of variables in each agent.

CR Crossover probability, where $0 \leq CR \leq 1$. A typical value is 0.9.

F Differential weight, where $0 \leq F \leq 2$. A typical setting is F = 0.8.

The algorithm also requires a maximum number of iterations or a tolerance to be defined. It can end when the maximum number of iterations has been reached or at the point where the change in the computed objective function between iterations changes by less than the required tolerance.

The DE algorithm is relatively short and is summarized below.

1. Initialize each agent, $\vec{x}$, with random positions in allowable space, $\vec{r}$.
2. Define the maximum number of iterations allowable, the tolerance allowable, or the maximum runtime.
3. For each agent in the population, x in the population of *NP* agents, do the following.
    a. Randomly choose 3 agents out of the population of possible agents. They must be distinct from each other and from x. Identify these agents as a, b, and c.
    b. Form a new candidate agent.
        a. Pick a random number, R such that $1 \leq R \leq n$.
        b. For each variable index i where $1 \leq i \leq n$:

i. If $\big((r_i < CR)\ or\ (i = R)\big)$ then set $y_i = a_i + F\ (b_i - c_i)$, otherwise, set $y_i = x_i$.

ii. If f(y) <= f(x) then $\vec{x} = \vec{y}$, otherwise $\vec{x} = \vec{x}$.

iii. If the number of iterations is less that the maximum number allowable or $\frac{f(\overrightarrow{x_{best}} - \overrightarrow{x_{previous}})}{f(\overrightarrow{x_{best}})} \leq tolerance$, then print out the variables for the best candidate and halt execution. Otherwise, iterate back to step 3.

CHAPTER 3

FUEL CORROSION WITHIN REACTOR FUEL AND FUEL COATING

3.1 Introduction

The NERVA tests produced excellent results with specific impulse values approaching 900 seconds, however fuel corrosion tended to be a problem. The fuel was composed of uranium carbide and zirconium carbide with a melting temperature of 2600 K. The hexagonal fuel elements also contained coolant channels to allow the hydrogen propellant to pass through and be heated by the reactor. To minimize corrosion of the fuel due to the hot hydrogen, a coating of niobium carbide was applied to the inside of each coolant channel.

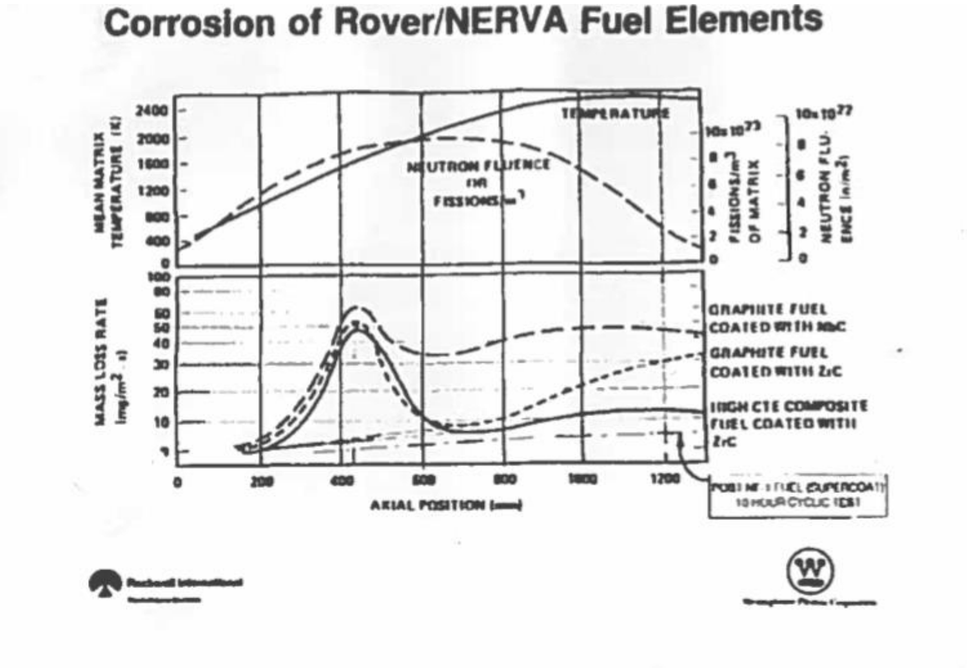
The hydrogen propellant entered the reactor at about 120 K and quickly reached 2700 K before exiting the de Laval nozzle at the end of the reactor. This produced very high temperature gradients in both the axial direction and in the radial direction extending from the center of each coolant channel through the coating and into the fuel. Although the niobium carbide coating and the uranium carbide fuel had similar temperature coefficients of expansion, the coating eventually cracked, and some was removed from the coolant channel causing interaction and ejection of the reactor fuel. The coating was then changed to zirconium carbide and further experiments showed decreased corrosion rates. The corrosion data obtained from the Pewee experiments [Pelaccio] is shown in Fig. 23. NERVA Rocket Fuel Corrosion Rates.

An analysis conducted by Rockwell [Rockwell International] in 1992 indicated their opinion that corrosion towards the middle of the reactor could be due to fission fragment damage to the graphite fuel matrix. This would have decreased the thermal conductivity of the fuel and higher thermal gradients would have caused higher thermal stresses leading to cracking of the coating and allowing hydrogen to react with the fuel. They noted that cracking of the coating did not occur during electrical heating tests of the reactor.

The goal of this section was to utilize corrosion data for nuclear fission rocket engines to determine the thermodynamic conditions that lead to maximum corrosion rates. These results were then used to determine the appropriate changes that can be made in the reactor design to minimize corrosion rates.

3.2 Procedure

During the 1960's and 70's, the United States conducted extensive testing of nuclear fission rockets [Benensky]. Corrosion of the reactors by hot hydrogen propellant led to the loss of carbide-based uranium fuel from the reactor and limited the rocket lifetime. Different coatings were applied to minimize corrosion with zirconium carbide providing the greatest success. Corrosion data is available from Project ROVER and the NERVA program based on experiments conducted at the Nevada Test Site. This data, shown in Fig. 23, provided corrosion rates for composite carbon fuel with ZrC (zirconium carbide) coatings and showed a peak corrosion rate of about 50 mg/m^2-s at about 33% of the reactor length [NASA]. Although limited data is available on the corrosion of the Pewee reactor, this data was curve fitted and related to temperatures and temperature gradients within the rocket engine to determine if there was a correlation. The data was then used to determine possible methods to alter NTP rockets to minimize fuel and fuel coating corrosion.



Corrosion of Rover/NERVA Fuel Elements

Fig. 23. NERVA Rocket Fuel Corrosion Rates

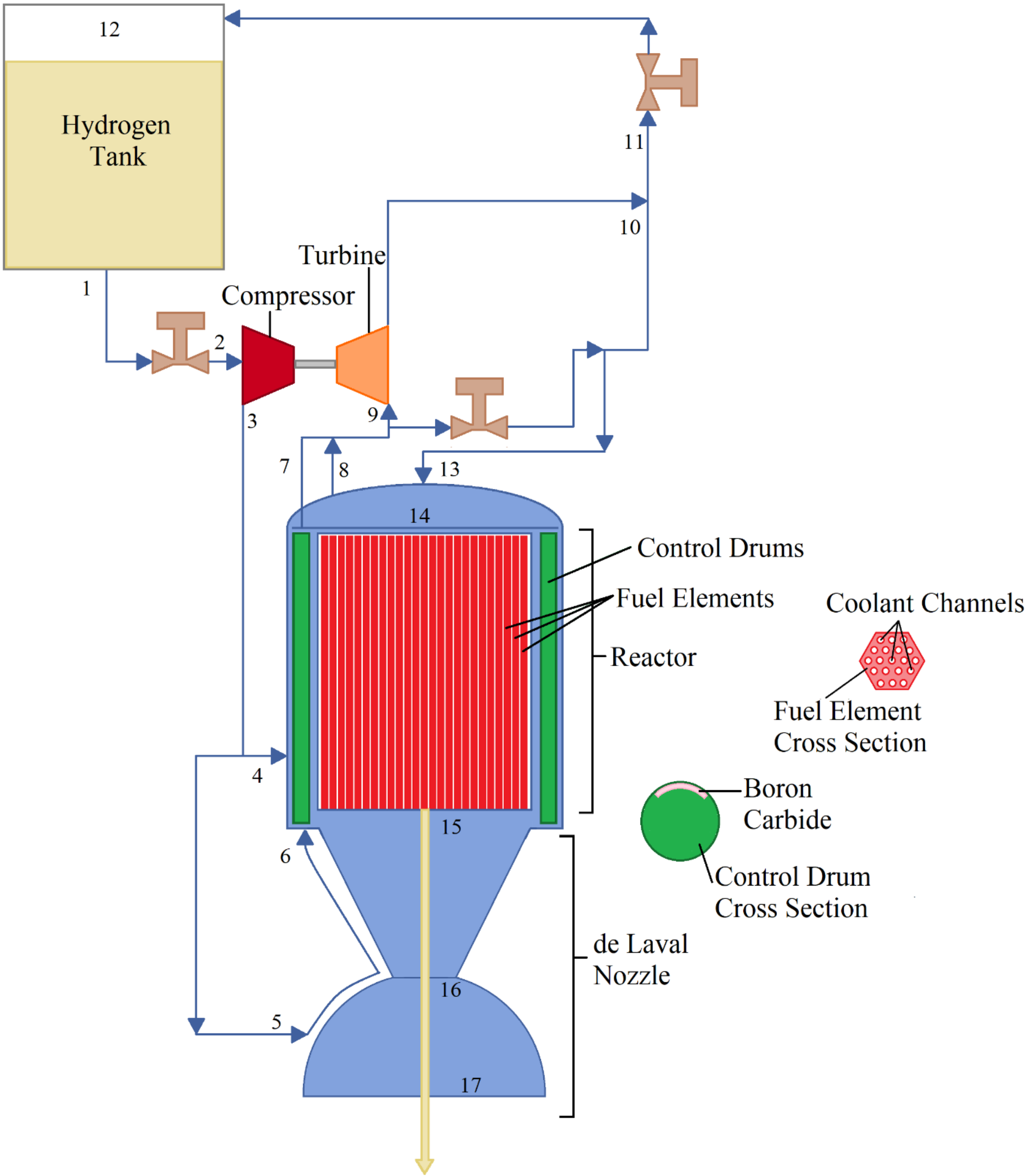


Fig. 24. Diagram of the Hydrogen Propellant Flow Through the NTP Reactor

Curvefitted plots of the bulk fluid temperature and the neutron flux within the reactor core were taken from Fig. 23 and converted into Excel plots. The axial distance through the reactor, z, was normalized by the length of the reactor core with the nondimensional coordinate $z^* = \frac{z}{L}$, where L = 1.32 m for the Pewee reactor core. The z^* axis begins at the top of the reactor, shown in Fig. 24, and extends to the bottom of the core where the hydrogen propellant enters the de Laval nozzle. Fig. 25 contains a simple curvefit of the fuel temperature along the axis of the reactor and Fig. 26 demonstrates the fuel fission density.

$$\frac{dR'''_{fission}}{dt} = \Sigma_f \ \varphi(z) \tag{181}$$

This term is proportional to the neutron flux and Fig. 26 serves as an indication of the change in neutron flux along the z-axis of the reactor. The fuel temperature curve is also predicted by the nuclear rocket computer model, UNLV-SNRE that was developed at UNLV for this research. UNLV-SNRE analyzes the heat generated by fission in the reactor, the heat transfer that occurs to the hydrogen propellant, and the flow of hot gas through the de Laval nozzle to generate thrust. UNLV-SNRE follows the analysis outlined by [Labib and Khatry]. Data from the Pewee experiments [N-Division] was used to model the reactor core. The results of this program provided the bulk hydrogen temperature along the axis of the reactor as the hydrogen heated from its inlet temperature of about 120 K to its final temperature of over 2500 K.

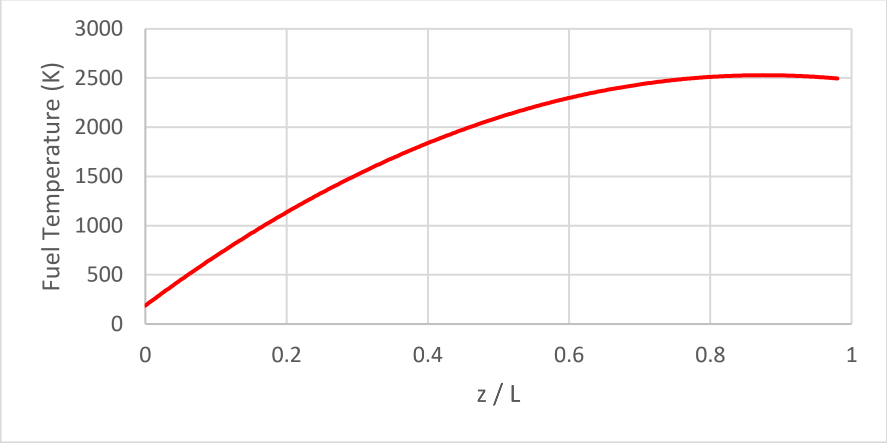


Fig. 25. Fuel Maximum Temperature vs. Axial Distance

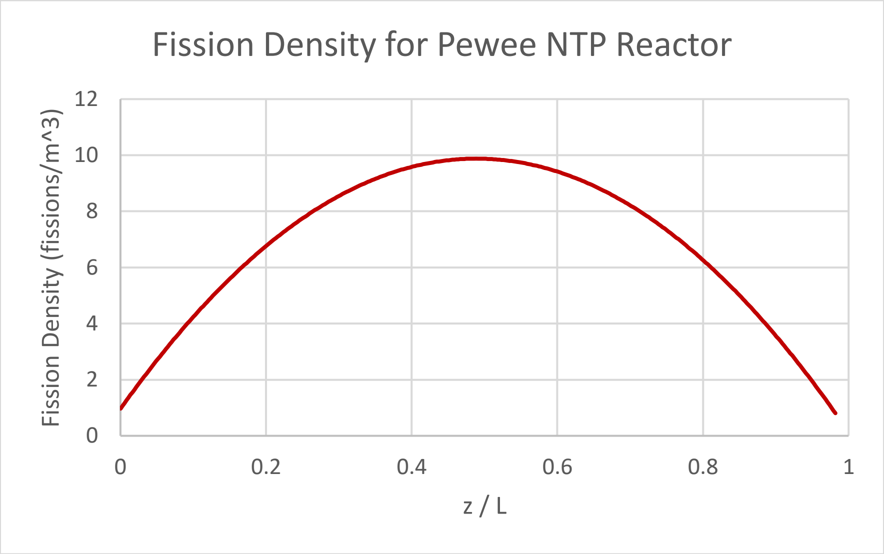


Fig. 26. Fuel Fission Density Proportional to Neutron Flux

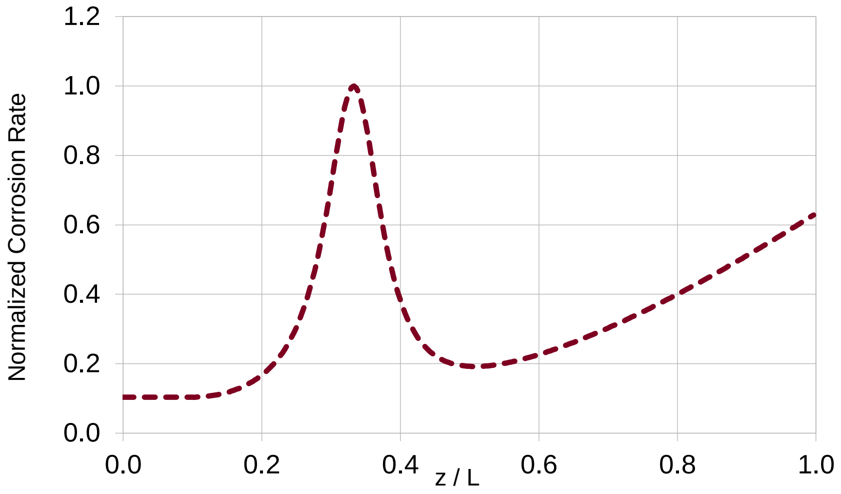


Fig. 27. Nondimensional Fuel Corrosion Rate vs. Axial Distance

Fig. 27 shows the raw data from Fig. 23 for the case of the graphite fuel that Pewee employed. To determine a function that describes the dependence of the corrosion rate upon temperatures and temperature gradients within the reactor, a curve fit of the data in Fig. 27 was needed. The curve fit was complicated by the peak that occurs in the corrosion curve near $z^* = 0.33$. Neutron cross-sections have peaks that are similar to the one seen in Fig. 27 and these have successfully been curve fitted using the Breit-Wigner equation. The curve fit also included a third-order polynomial fit to accommodate the gradual rise in corrosion rate along the z^* axis.

The corrosion rate data was curve fitted as shown in Fig. 28 with the following coefficients in units of (mg/m^2-s) using GNUPLOT, since this software is capable of nonlinear curve fitting. The standard error for the curve fit was ±0.989 or about ±2%.

$$\frac{dR_c}{dt}(z^*) = \frac{A\sqrt{B/_{z^*}}}{1+\left(\frac{2}{C}\,(z^*-B)\right)^2} + D + E\,z^* + F\,z^{*2} + G\,z^{*3} \tag{182}$$

Table 1. Curve fit Coefficients for Corrosion Data

Coefficient	Value
A	48.0712
B	0.334215
C	0.097741
D	3.23837
E	-23.3346
F	68.1885
G	-16.2579

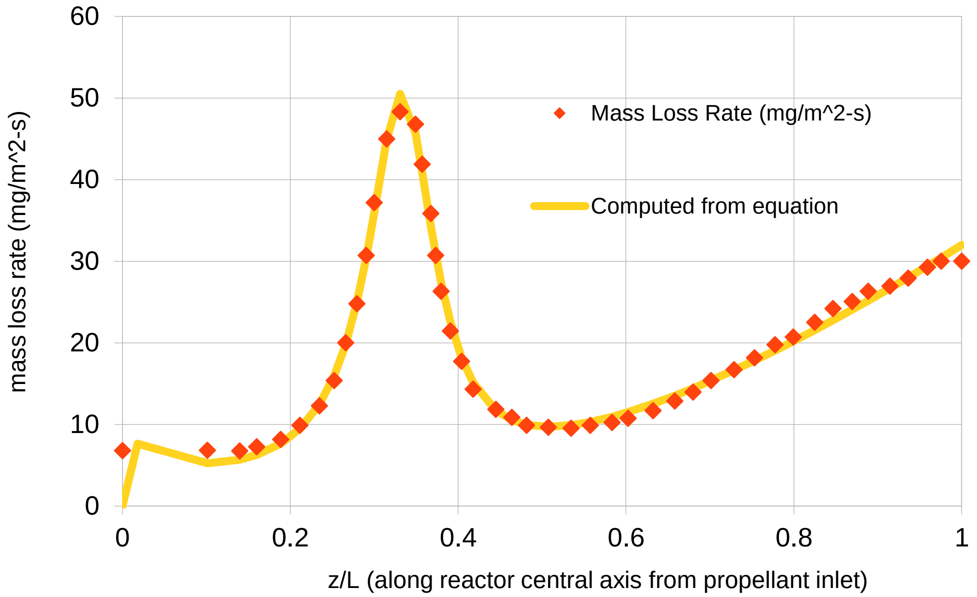


Fig. 28. Comparison of Measured Corrosion Data and Equation (182)

The corrosion data and temperatures obtained from Fig. 23 were combined with results from UNLV-SNRE.c and with temperature gradients computed numerically. The results are shown in Fig. 28.

3.2.1 Analysis of the Corrosion Data

In this study, the measured data was used to obtain a curve fit of corrosion rate versus temperature and temperature gradients within the core. The curve fit was used to explore possible changes to the neutron flux within the core that would reduce corrosion rates.

The computed temperature distribution within the Pewee I core is shown in Fig. 29. The corrosion rate is shown as a dashed line and the peak of the corrosion curve occurs at roughly the same point as the maximum in $(T_{wall} - T_{bulk})$. Temperatures and temperature gradients can lead to diffusion of hydrogen into the fuel matrix and produces mechanical stresses in fuel that can lead to coating or fuel damage. It was assumed that the corrosion rate was a function of the axial fuel temperature gradient in the reactor, the radial fuel gradient within each fuel element, the bulk fluid temperature, and the difference in temperature between the wall (coating surface) and the bulk fluid. These variables were nondimensionalized for the curve fit.

$$\frac{dR_{corrosion}}{dt}(z^*) = f\left(\begin{array}{cc} \frac{dT_{fuel}}{dz}\left(\frac{L_{reactor}}{T_{bulk-inlet}}\right), & \frac{dT_{fuel}}{dr}\left(\frac{R_{channel}}{T_{bulk-inlet}}\right), \\ \frac{T_{bulk} - T_{bulk-inlet}}{T_{bulk-inlet}}, & \frac{T_{wall} - T_{bulk}}{T_{bulk-inlet}} \end{array}\right) \tag{183}$$

$$\frac{dR_{corrosion}}{dt}(z^*) = f({T_z}^*, {T_z}^*, {T_{bulk}}^*, \Delta T^*) \tag{184}$$

$${T_z}^* = \frac{dT_{fuel}}{dz}\left(\frac{L_{reactor}}{T_{bulk-inlet}}\right) \tag{185}$$

$$T_r{}^* = \frac{dT_{fuel}}{dr}\left(\frac{R_{channel}}{T_{bulk-inlet}}\right) \tag{186}$$

$$T_{bulk}{}^* = \frac{T_{bulk} - T_{bulk-inlet}}{T_{bulk-inlet}} \tag{187}$$

$$\Delta T^* = \frac{T_{wall} - T_{bulk}}{T_{bulk-inlet}} \tag{188}$$

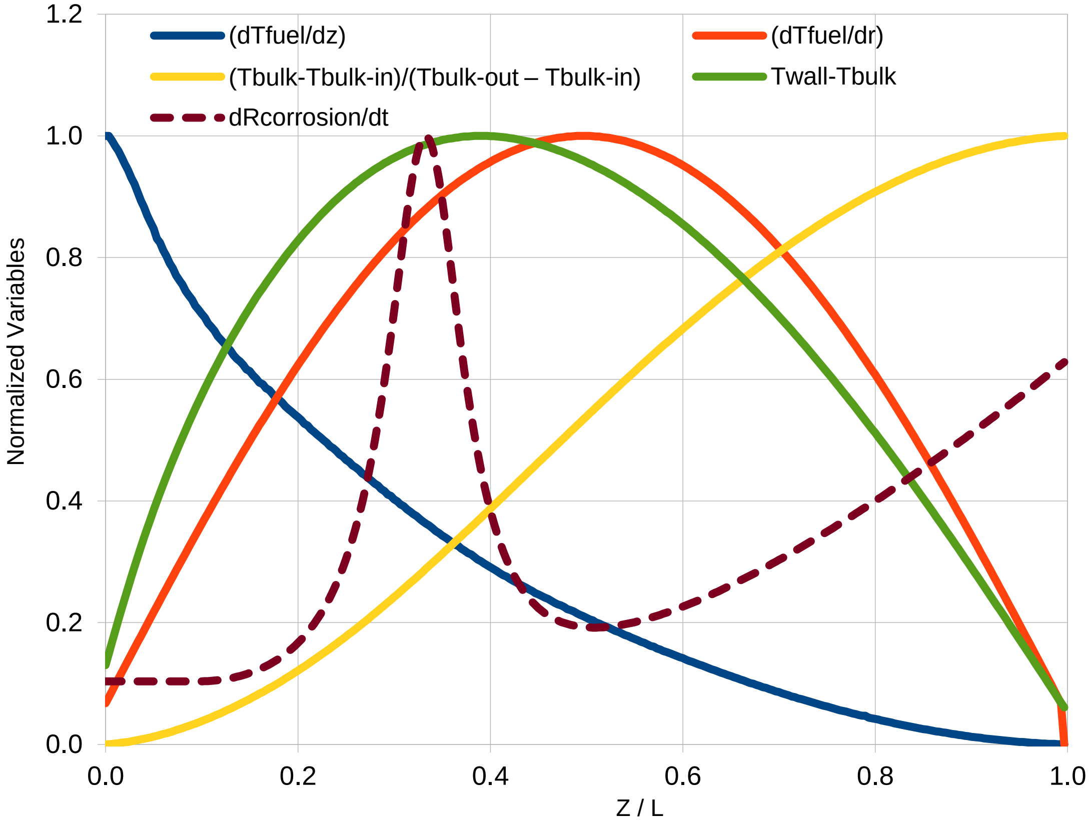


Fig. 29. Predicted Temperatures, Gradients, and Corrosion Rates in the Pewee I Reactor

While GNUPLOT has an excellent ability to fit non-linear equations like equation (183) to data, it could not converge on the corrosion data set available for Pewee. The use of a differential evolution (DE) optimization program that was written at UNLV was used to complete the curve fit. This program, DE-DATAFIT.bas, considered over two million possible sets of coefficients to arrive at the final equation. The program was written in FreeBASIC, which produces compiled code. A copy of the program, the input file, input.cor, and the corrosion data file, "corrosion-pewee-dim.txt" are included in the appendices.

3.2.2 Development of the Relationship for the Corrosion Rate

The corrosion rate was curve fitted with respect to reactor temperatures and temperature gradients using a differential evolution algorithm. Two million generations of estimates for the curve fit were used by the algorithm to arrive at the following distribution shown in Fig. 27. The values of dT/dz used in the curve fit were smoothed but lead to fluctuations in the predicted corrosion rates. The curve fit indicates that dT/dr is the most important contributor to corrosion and the bulk, or hydrogen propellant temperature increase to 2700 K within the rocket leads to the gradual rise in the corrosion rate as z/L approaches 1.0.

Temperature increases lead to expansion of the fuel and the fuel coating, and the bulk temperature of the hydrogen propellant increases significantly, from 120 K to over 2500 K, along the axial length of the reactor. Temperature gradients also lead to thermal stresses and these stresses occur in the radial and axial directions.

First, the temperature functions that could most effect cracking and corrosion of the fuel and fuel coating are nondimensionalized. Four variables were selected based on their possible influence on coating degradation and corrosion.

The curve fit and coefficients are given by:

$$\frac{dR_{corrosion}}{dt}(z) = f(T_z^*, T_r^*, T_{bulk}^*, \Delta T^*) \tag{189}$$

$$\frac{dR_{corr.}}{dt}(z) = a_o + \sum_{i=1}^{3} a_i (T_z^*)^i + \sum_{i=1}^{3} b_i (T_r^*)^i + \sum_{i=1}^{3} c_i (T_{bulk}^*)^i + \sum_{i=1}^{3} d_i (\Delta T^*)^i \tag{190}$$

Table 2. Curve Fit Coefficients for Equation (190)

i	a_i	b_i	c_i	d_i
0	106.1350554			
1	-11.86010171	-8927.378788	-14.14092956	8.813951566
2	-0.139700081	28478264.62	0.943439759	0.62247229
3	0.017453564	-2444520702	-0.029010876	-7.339240217

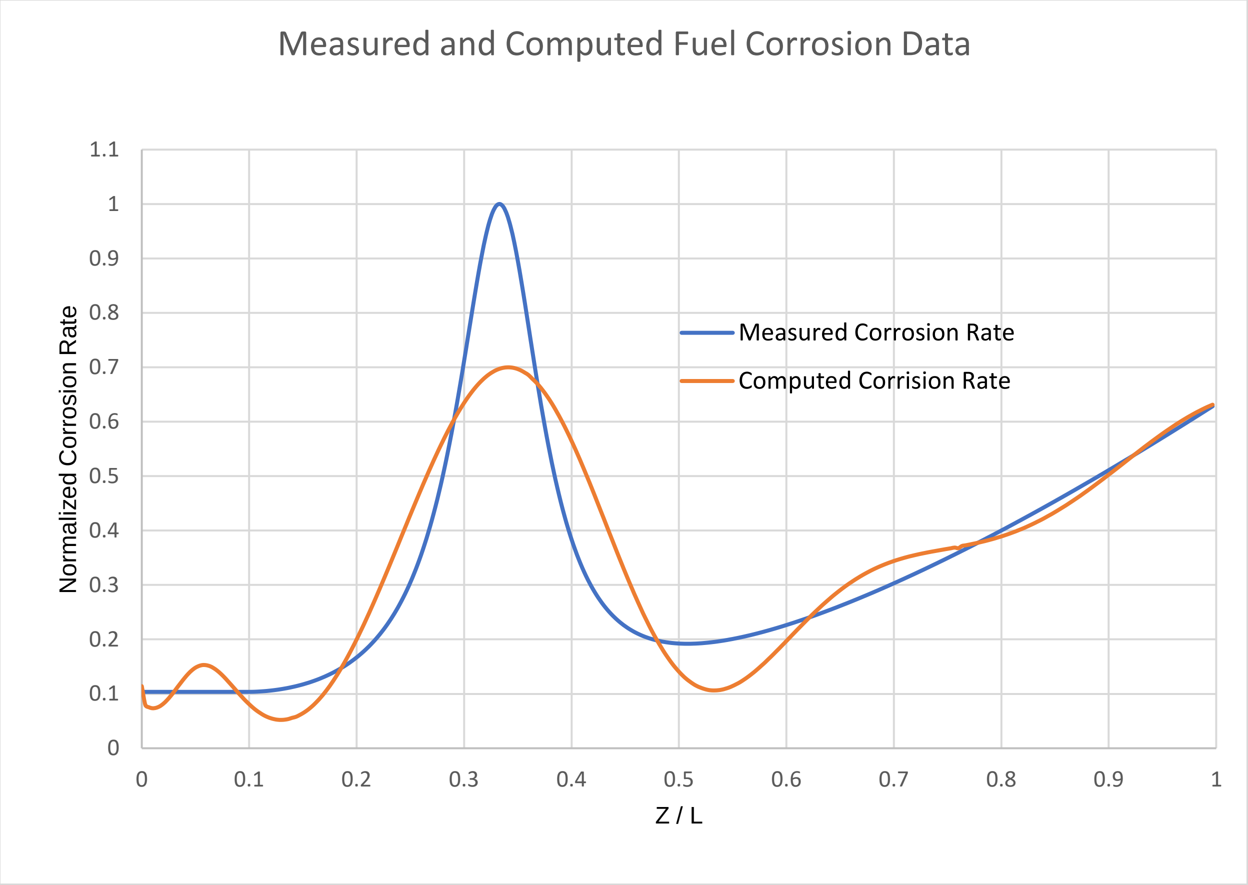


Fig. 30. Curve Fit of the Corrosion Data Based on Equation (190)

The standard error of the curve fit is ±0.0831 or ±8.31% and was achieved after the program considered 2 million generations of variables. The equation took 234 seconds to fit.

3.3 Results of the Analysis

The curve fit shows that the corrosion rate has some dependence on T_z^*, and the response is fairly linear with z^*. This, however, is one of the variables that can likely be controlled since the addition of larger reflectors at the top and bottom of the core can create a flatter, more uniform neutron flux resulting in a temperature distribution in the z-direction that is more constant. This solution was used to see how corrosion rates could be affected by increasing the neutron reflector thicknesses and by applying reflectors at each end of the reactor core. To model the impact of larger reflectors, the extrapolation distance at each end of the reactor core was increased. This had the same effect as increasing the reflector thicknesses and was easy to implement in the UNLV-SNRE code. It should be noted that a more uniform neutron flux within the core changes how both the fuel and bulk hydrogen temperatures change along the axis of the core. The four variables listed in equations (185) through (188) are not independent and all change with the neutron flux distribution.

The most significant impact on corrosion rate was seen from $T_r^*(z^*)$. This term uses the maximum fuel temperature and the temperature at the interface with the coating to calculate the derivative. The highest impact, according to the curve fit, occurs at $z^* = 1\%$ within the reactor core measured from the propellant inlet and this impact decreases rapidly after this point. The best recommendation to mitigate this term would be to use a graduated fuel enrichment or uranium concentration in the z-direction to decrease the fuel temperature at the inlet of the reactor. This would also, unfortunately, increase T_z^*, but the radial temperature change and the subsequent radial stresses within the fuel should be mitigated first.

The normalized bulk fluid temperature, ${T_{bulk}}^*$, has a low impact on corrosion rates and decreases slightly from the reactor inlet to the outlet.

The last term studies the impact of the normalized difference between the bulk fluid temperature and the coolant channel wall temperature, given by $\Delta T\ ^*$. This term has a parabolic impact on the corrosion rate with the minimum values occurring at the inlet and outlet and the maximum value occurring at $z^* = 67\%$. This term is the likely cause of the increase in corrosion rate from z^*= 50% to 100% of the reactor length.

3.3.1 Reactor Changes that can Lead to Lower Corrosion Rates

The curve fitted experimental data suggest that modifications to the neutron flux profile within the core can decrease corrosion rates. The neutron profile is one parameter that can be modified with relative ease by altering the neutron reflectors at the axial ends of the core. NTP designs do include a beryllium reflector near the inlet of the core, but the de Laval nozzle makes it difficult to add a neutron reflector near the outlet of the core.

In the first case, the neutron flux is modified by reducing the extrapolation length to zero. This alters the neutron flux to a cosine distribution that reaches zero flux at the top and bottom of the core. The estimated corrosion rates using equation (190) are shown in Fig. 31. These results are compared to the Pewee core with an extrapolation length of 2.52 cm as shown in Fig. 30. Equation (190) predicts that the peak corrosion would decrease to 82% of the value measured during the Pewee experiments which occurs at $z^* = 0.31$, close to the original position.

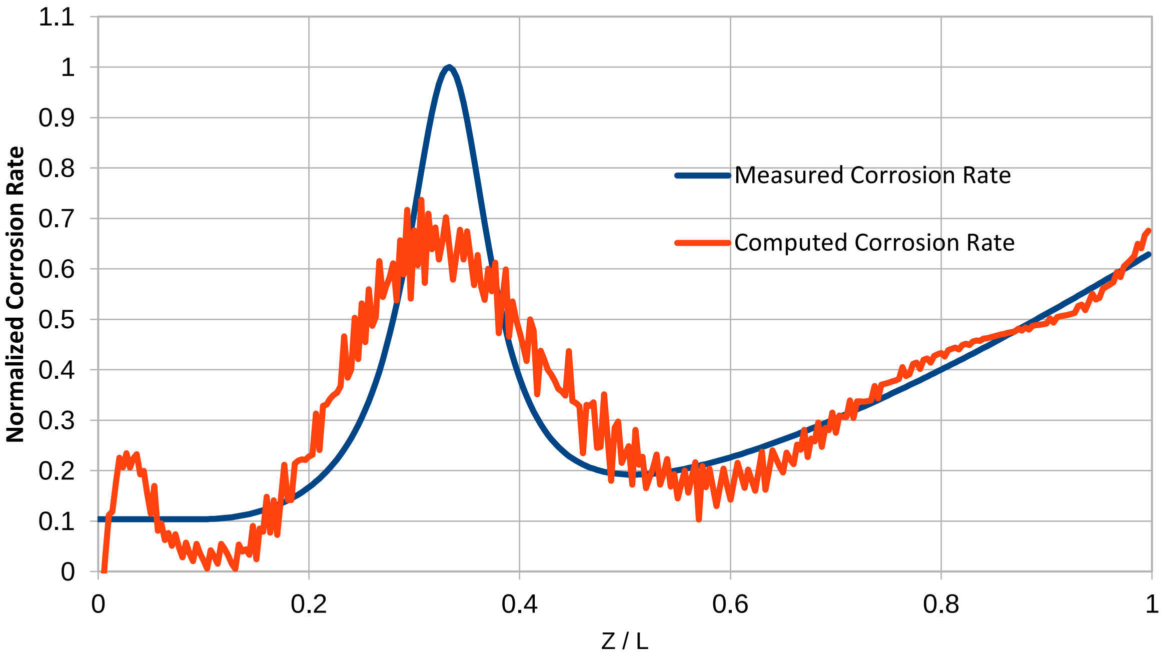


Fig. 31. Estimated Corrosion Results for a Sinusoidal Neutron Flux Profile (Zero Extrapolation Length)

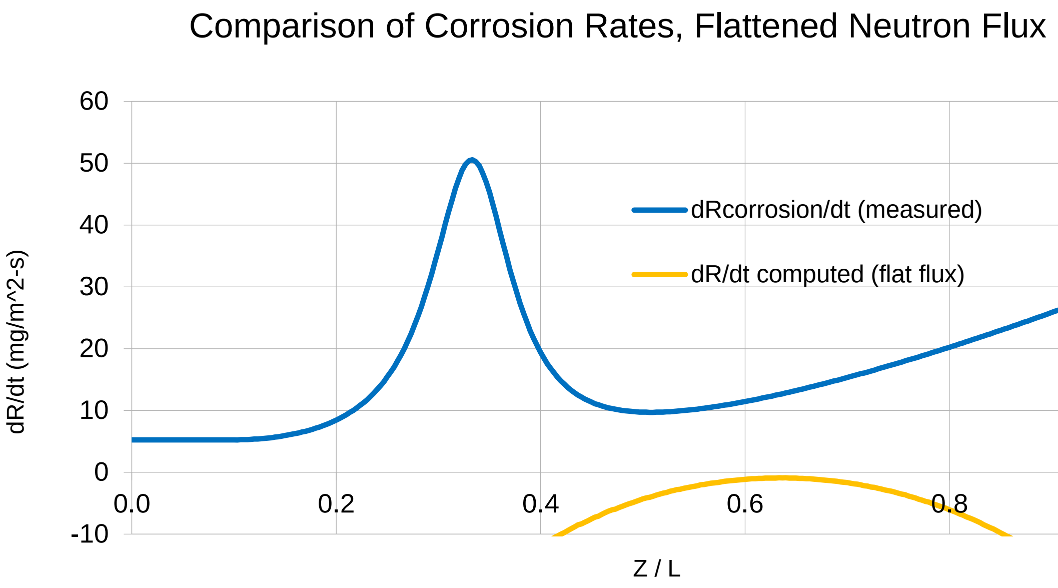


Fig. 32. Corrosion Results for a Uniform Neutron Flux in the Curve

The extreme case of a perfectly uniform neutron flux was attempted next. This case would correspond to perfect neutron reflectors at each end of the core. The results, seen in Fig. 32, had the corrosion peak at z* = 0.643, and equation (190) predicted no corrosion within the core. While this case is unrealistic, it appeared that enhancement of the neutron reflectors at each end of the core could decrease corrosion rates.

Fig. 33 shows the effect of doubling the extrapolation length at each end of the core. This mimics the effect of increasing the reflector thicknesses. The estimated corrosion rate dropped to 75% of the rate measured in the Pewee experiments with peak corrosion occurring at z* = 0.34. The corrosion rate for this case reached 25.5 (mg/m^2-s).

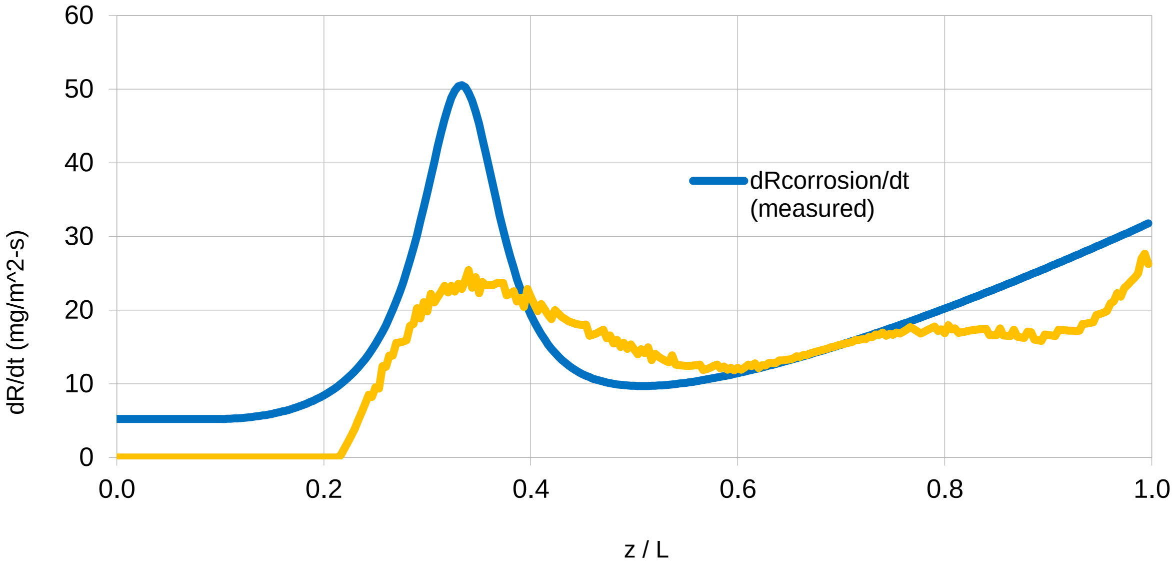


Fig. 33. Corrosion Results for Extrapolation Lengths Doubled

The last case considered the impact of increasing the extrapolation distances by a factor of 4. The estimates from equation (190) are shown in Fig. 34. The estimated corrosion rate decreased to 53% of its original value with a peak that moved to z* = 0.49.

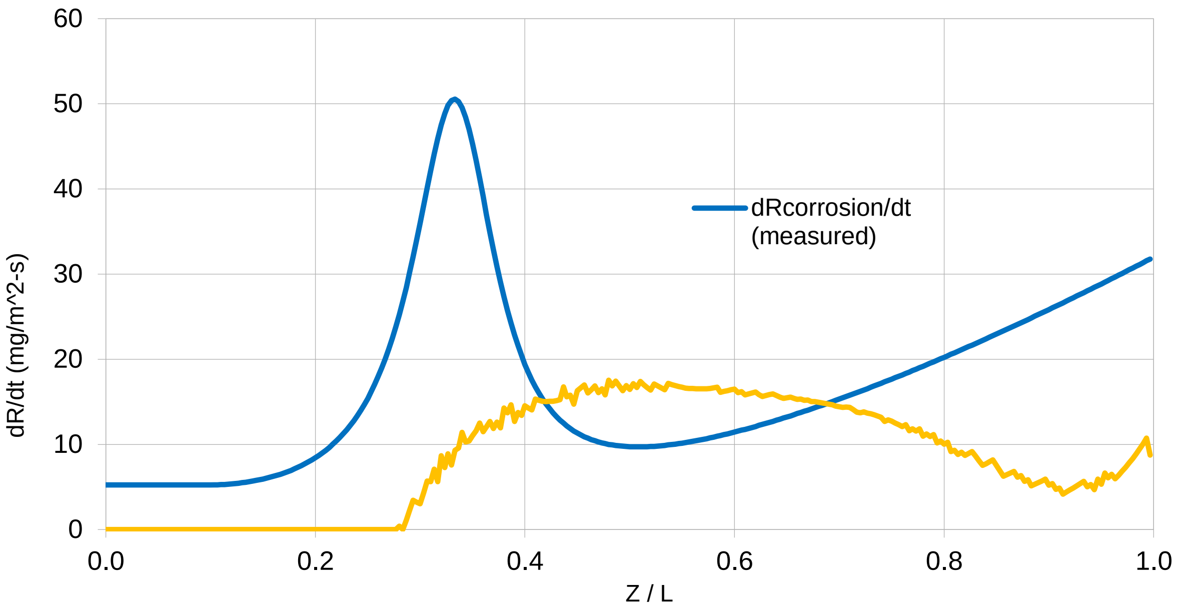


Fig. 34. Corrosion Results for Extrapolation Lengths Quadrupled

The corrosion rates estimated from equation (190) show that the neutron flux can be decreased by using methods to make the neutron flux more uniform through the reactor core. In this work, the flux was altered by increasing the neutron reflection at each end of the core. Alternative methods to accomplish this same goal include using a uranium concentration that varies along the z-axis to enhance heat generation at the ends of the core or by varying the uranium enrichment. Changes in coolant channel diameter as a function of axial length could also help to vary fission heat production by changing the cross-sectional area available for fuel.

3.4 Conclusions

The results indicate that by adding neutron reflectors to both the top and bottom of the reactor, the corrosion rate can be significantly reduced. This may not be practical on the outlet side of the reactor, but improvements can be made at the entrance to flatten the neutron flux and several methods are recommended to accomplish this. The data in Table 3 shows that, by making the neutron flux more uniform through the core, the estimated corrosion rate decreased to 53% of the rate experienced in the NERVA experiments.

Table 3. Corrosion Mitigation Results

Case	**Peak Corrosion Rate**	**Peak Location**
-	Relative to Base	z/L
Pewee (Base Case)	100%	0.34
Sinusoidal (Zero Extrapolation Distance)	82%	0.31
Uniform Neutron Flux	0%	0.64
Extrapolation Length Doubled	75%	0.34
Extrapolation Length Quadrupled	53%	0.49

CHAPTER 4

HYDROGEN PROPELLANT SAVINGS USING A VARIABLE AREA THROAT NOZZLE

Nuclear thermal rockets (NTR) have been explored as an alternative to chemical rockets to provide propulsion for missions to Mars and throughout the solar system. They employ a thermal nuclear reactor to heat hydrogen propellant which is ejected through a de Laval nozzle to generate thrust. Nuclear reactors generate radioactive fission products during operation and the decay of these isotopes create significant decay heat once the reactor is shut down. In a nuclear thermal rocket, some hydrogen propellant must be passed through the reactor after shutdown to remove decay heat and prevent overheating of the reactor components. The mass flowrate of this propellant is insufficient to generate supersonic exhaust velocities, and an insignificant amount of thrust is produced.

Hydrogen propellant is a precious resource in a nuclear rocket since large amounts are consumed during each burn of the rocket engine. This section investigates if it is possible to use a variable area de Laval nozzle to generate useful thrust for the rocket by looking at the increase in rocket impulse and thrust based on the use of a variable area nozzle, and the results are presented as a function of time after engine burn for burn durations of 1, 10, 30, and 60 minutes.

4.1 Theory

4.1.1 Compressible Flow

In a chemical rocket, combustion of the fuel provides the heat source and the expanding exhaust gases serve as the propellant to produce thrust. A de Laval nozzle accelerates the subsonic gases to Mach 1 at the throat of the nozzle. The gases then expand in the diverging section to accelerate to supersonic speeds resulting in a large Mach number at the exit of the nozzle producing significant thrust. Compressible flow theory shows that, to attain supersonic flow, the Mach number at the throat must choke to a value of 1. Rocket nozzles are usually designed with a fixed throat diameter, and this also establishes the maximum mass flowrate of propellant through the engine. The mass flowrate through the nozzle is a function of the stagnation pressure and temperature in the engine combustion chamber, the ratio of specific heats, the gas constant of the propellant, and the area of the throat, A*.

$$\dot{m}_{max} = \frac{p_o A^*}{\sqrt{T_o}} \sqrt{\frac{k}{R}\left(\frac{2}{k+1}\right)^{\frac{k+1}{2(k-1)}}} \qquad (191)$$

It is possible to design a nozzle with a variable area throat, but high temperatures and flowrates at the throat can lead to erosion of the metal walls. A variable area throat would allow the mass flowrate of propellant to be changed to maintain supersonic flow at the de Laval nozzle exit.

4.1.2 Decay Heat Production

As a nuclear reactor operates, nuclear fission results in the production of radioactive fission products, photons, and subatomic particles. Through collisions within the fuel, the kinetic energy of the fission products and particles is converted into heat within the engine and is transferred to the propellant. The fission products also create heat as they radioactively decay, and reactors must be cooled once the fission reaction is shut down to prevent reactor components from melting. In an NTR, thrust is generated by the heated hydrogen propellant during a burn as thermal power is generated by fission. After shutdown, propellant must continue to be circulated through the reactor core to keep the fuel from exceeding a maximum temperature where damage to the fuel or reactor components would result. The decay heat generation rate can be found from equation (192) based on the work of Pond and Matos where $\mathrm{t_i}$ is the irradiation, or burn time, and $\mathrm{t_d}$ is the post-irradiation time, both measured in days.

$$\frac{\mathrm{Power_{decay\ heat}}}{\mathrm{Power_{reactor}}} = 0.00685[\mathrm{t_d}^{-0.2} - (\mathrm{t_i} + \mathrm{t_d})^{-0.2}] \tag{192}$$

This equation was based on ORIGEN models of spent uranium-based nuclear fuel taken from TRIGA, MTR, and DIDO reactors. They noted that the decay heat generation rate was independent of the fuel assembly type. The fuel burnup in their studies was up to 80% and enrichments ranged from 17% to 93%. Pond and Matos also included two other equations from previous experimental studies, and an average of these three equations allows gives the ability to compute the rate of production of decay heat over time after a typical burn, as shown in Fig. 35.

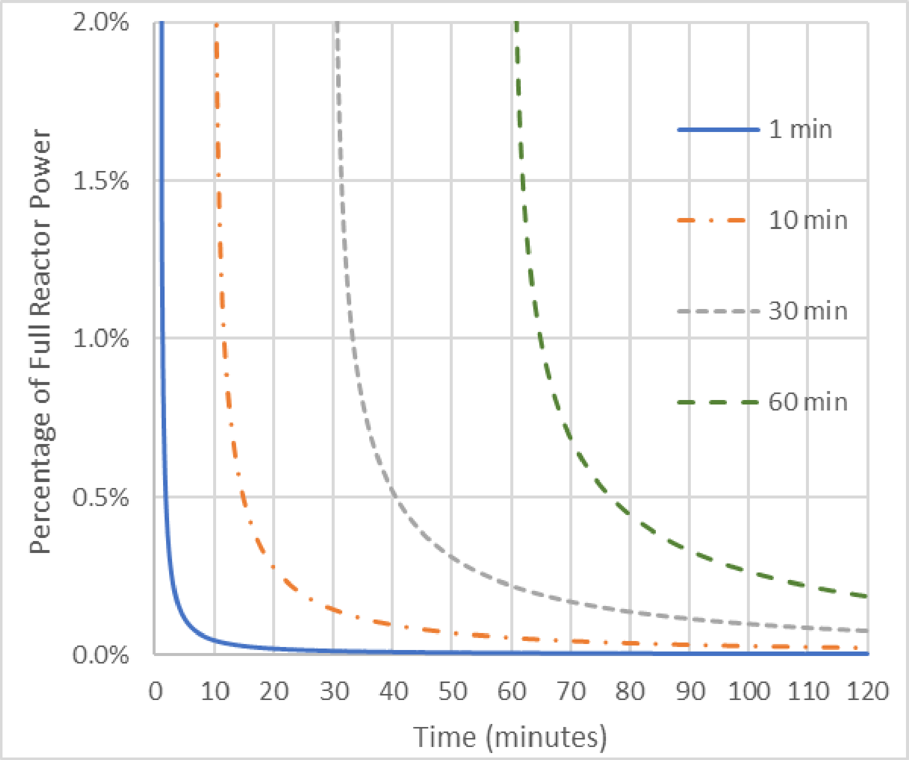


Fig. 35. Decay Heat Generation for Burn Times of 1, 10, 30, and 60 Minutes

After a burn of 60 minutes and 5 minutes after the reactor is shut down, an NTR will still generate decay heat equivalent to 1% of full reactor power. For a 500 MW rocket, over 5 mega-watts of decay heat will be generated for over an hour after the reactor is shut down. The mass flowrate of hydrogen that must be used to cool the reactor to prevent damage can be estimated from a heat balance as shown in equation (193).

$$\mathrm{Power}_{\mathrm{Decay\ Heat}} = \dot{m}\int_{T_{\mathrm{Reactor\ Inlet}}}^{T_{\mathrm{Reactor\ Maximum}}} c_p(T)dT \qquad (193)$$

The specific heat of the hydrogen propellant is integrated over the temperature change in the bulk fluid as it flows through the reactor. The mass flowrate of hydrogen is far below the design flowrate of the de Laval nozzle as dictated by equation (191). In current designs, hydrogen would flow through the reactor to remove decay heat after shutdown and flow at subsonic velocities into space, producing very little thrust. A computational model of the rocket engine was developed to compute the required throat area based on the mass flowrate of hydrogen that is required to cool the reactor after shutdown.

4.1.3 Variable Area Nozzles

The idea of using a variable area de Laval nozzle was patented in 1945 by Roach and Duemler. The inventors referenced a 1933 patent, U.S. 1,201,852 by Stolfa, as the earliest patent related to variable area nozzles. They proposed using a spring-loaded regulator cone in the nozzle upstream of the throat that would create an annular throat for the propellant to pass through. An example of this regulator cone is shown for the NTR in Fig. 36. In this example, the cone would be raised or lowered into the throat area by a mechanism attached to the upstream side of the reactor. The regulator cone could be cooled by passing cold hydrogen through passages in the cone, as is done in the Pewee I to prevent the tie tubes and control drums in reactor from melting.

Other researchers have also explored the design of nozzles. Twardy developed designs with a throttling range of 10:1 and a constant combustion chamber pressure by using a conical or bell-contoured combustion chamber wall. This was in conjunction with a mushroom shaped center body that is shifted to change the annular nozzle area in a hydrogen/oxygen engine. Prince patented multiple designs to improve the flow in variable area throats by using optimized center bodies to vary the annular throat area in the nozzle. Llewellyn pioneered mechanisms that could automatically vary the throat area by inserting obstructions downstream of a fixed center body to modify the throat area of a de Laval nozzle.

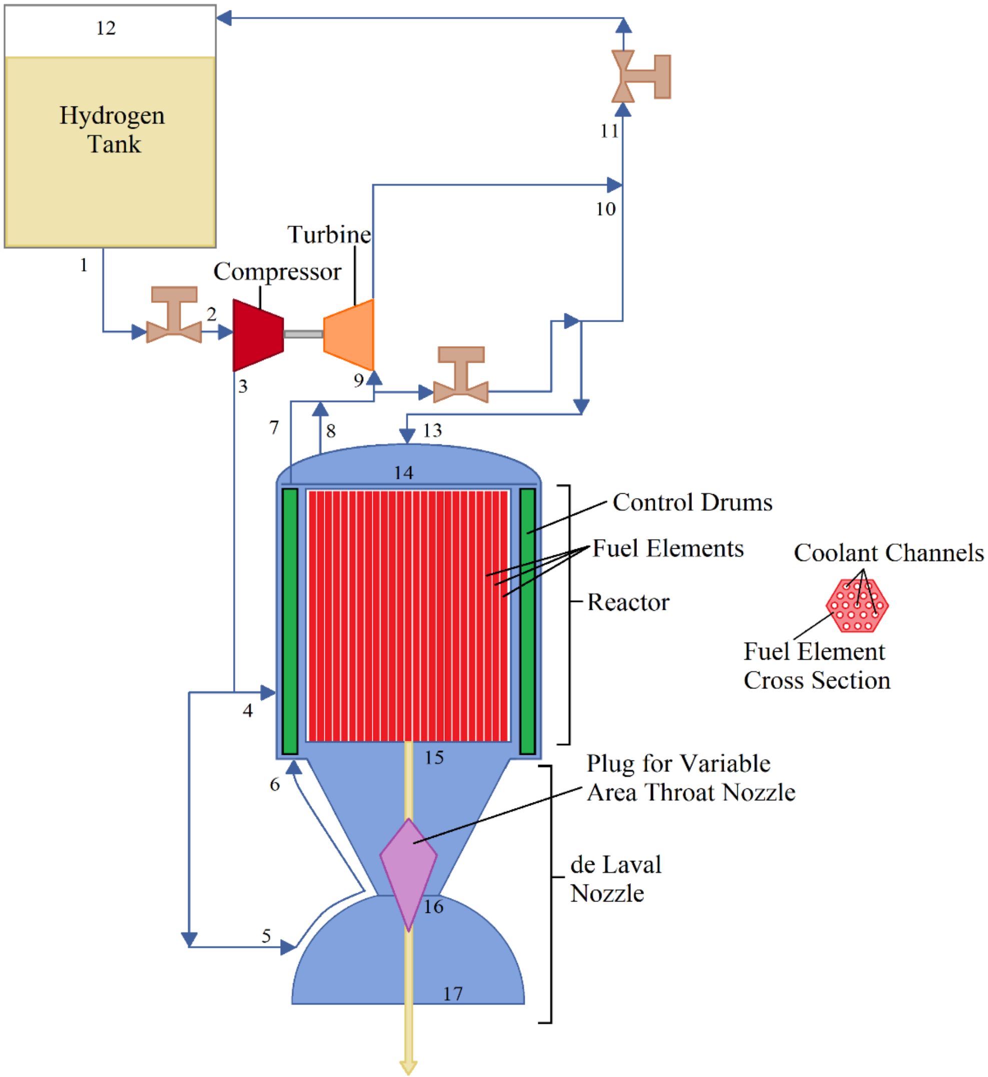


Fig. 36. NTP Rocket Design with a Variable Area Nozzle Placed near the Throat

4.2 Procedure

A computational model was developed to calculate the thrust that can be provided by using a variable area de Laval nozzle to capture decay heat. The model was based on the Pewee I and II designs from Project Rover. This small reactor was successfully tested in the 1970's and produced 25,000 lbf of thrust with a reactor power of 500 MW and a specific impulse of 900 s. The reactor was made of uranium and zirconium carbide in a graphite matrix with a diameter of about 0.5 m and a length of 1.32 m. The uranium fuel was enriched to 93%. Zirconium carbide was used as a plating material to prevent fuel damage. Fuel temperatures of over 2700 K were achieved. The design of Pewee is shown in Fig. 36. Liquid hydrogen is pumped from a storage tank with a temperature below 20 K. The hydrogen flows around the de Laval nozzle and past the control drums to provide cooling to prevent component damage. The hydrogen then flows down through coolant passages in the fuel elements to the reactor chamber and through a de Laval nozzle into space. In the simulations used, the exhaust Mach number exceeded 6.4 and resulted in over 130 kN of thrust.

To model the temperature distribution within the fuel, a technique outlined by Labib and King was used to treat the fuel elements as equivalent cylindrical fuel pins. The power distribution within a cylindrical reactor follows a cosine distribution in the axial direction as described in El-Wakil. Application of equation (193) provided the change in bulk hydrogen temperature within the core and convection and conduction heat transfer was used to compute the temperature distribution within the fuel coating and in the fuel. Fig. 37 shows the distribution of temperature along the reactor from the cold inlet at 120 K to the outlet at over 2,200 K. The coating and maximum fuel temperatures overlap in the figure.

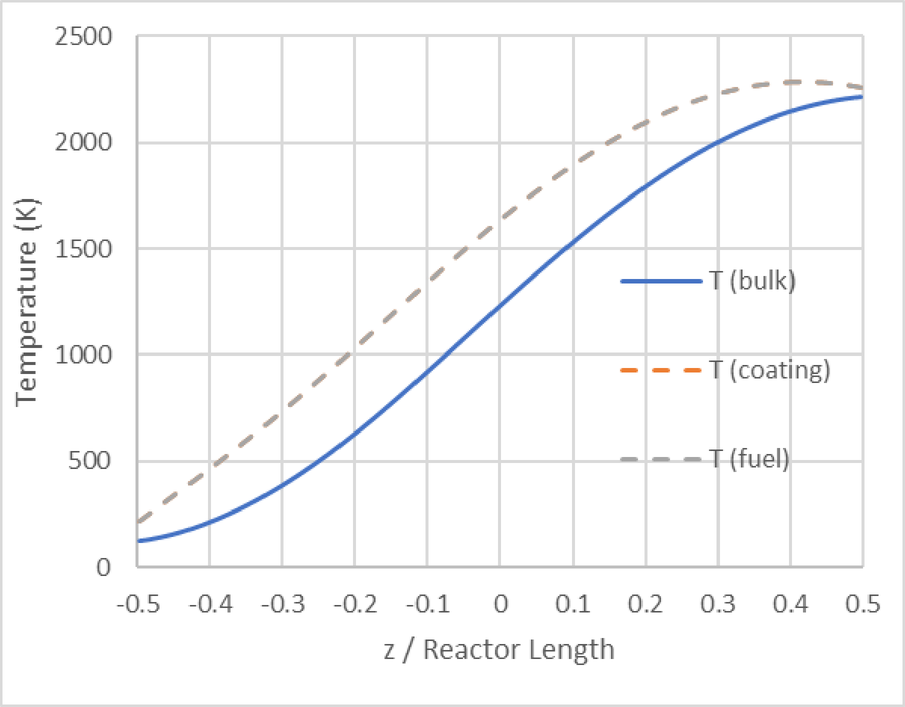


Fig. 37. Temperature Distribution in the Pewee Reactor

Hydrogen properties vary considerably over the temperature range of 20 K to 3000 K. This is due, in part, to the monatomic behavior of hydrogen gas below 300 K where k = 5/3. Data from NIST, as provided by Chase and others, was used to create a set of property functions for hydrogen.

The model was used to analyze the steady state performance of the Pewee reactor. The same model was modified to predict the decay heat generation after the rocket burn was completed. This data was used to predict the cross-sectional area of the throat, mass flowrate, and thrust as a function of time after the burn. Decay heat can be used to generate thrust, although this thrust varies over time. The impact of the additional thrust can be seen by computing the impulse that is added. Impulse is the integral of thrust over time. The impulse during the reactor burn is given by equations (194) and (195) which provide the additional impulse that is generated by the decay heat over time.

$$\text{impulse}_{\text{reactor}} = \int_{t=o}^{t_{\text{burn}}} \text{thrust}(t)dt \tag{194}$$

$$\text{impulse}_{\text{decay heat}} = \int_{t_{\text{burn}}}^{t_{\text{burn}}+t_{\text{post burn}}} \text{thrust}_{\text{decay heat}}(t)dt \tag{195}$$

4.3 Results

The impact of thrust from decay heat is most significant for long reactor burns. For a 60-minute burn, the rocket will still generate 1% of full power from decay heat after the first 5 minutes of shutdown. For Pewee, this amounts to 5 MW of decay heat. The normalized throat area is shown in Fig. 38 based on the mass flowrate of hydrogen required to cool the reactor after shut-down.

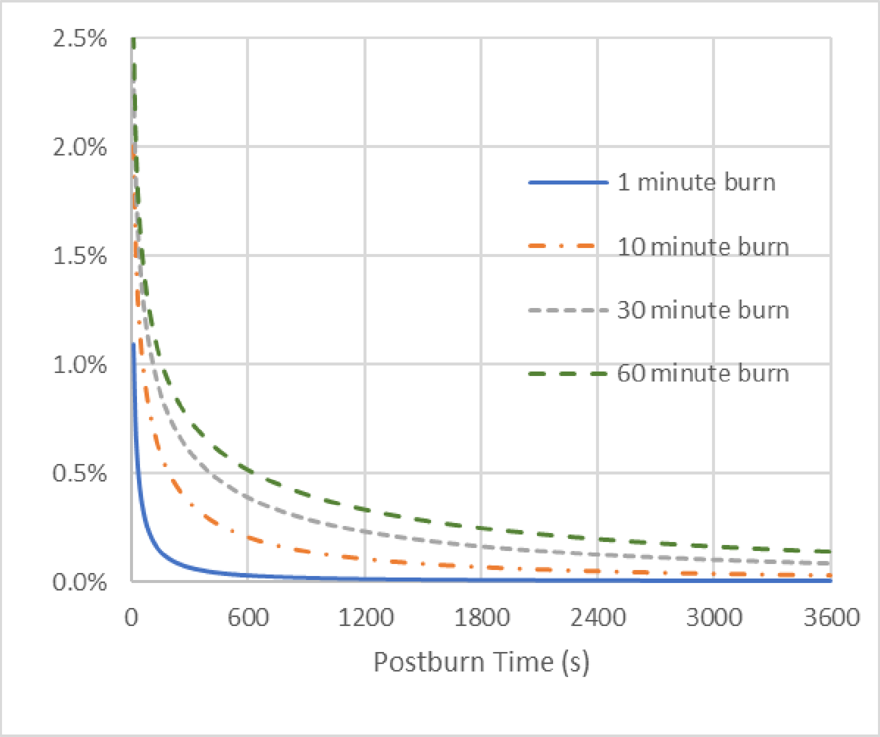


Fig. 38. Throat Area Ratio (Vertical Axis) as a Function of Postburn Time and the Length of the Burn

The thrust also varies considerably with burn time, as shown in Fig. 39. For a 60 minute burn, the rocket could still generate 1% of its full thrust for just over four minutes after reactor shutdown.

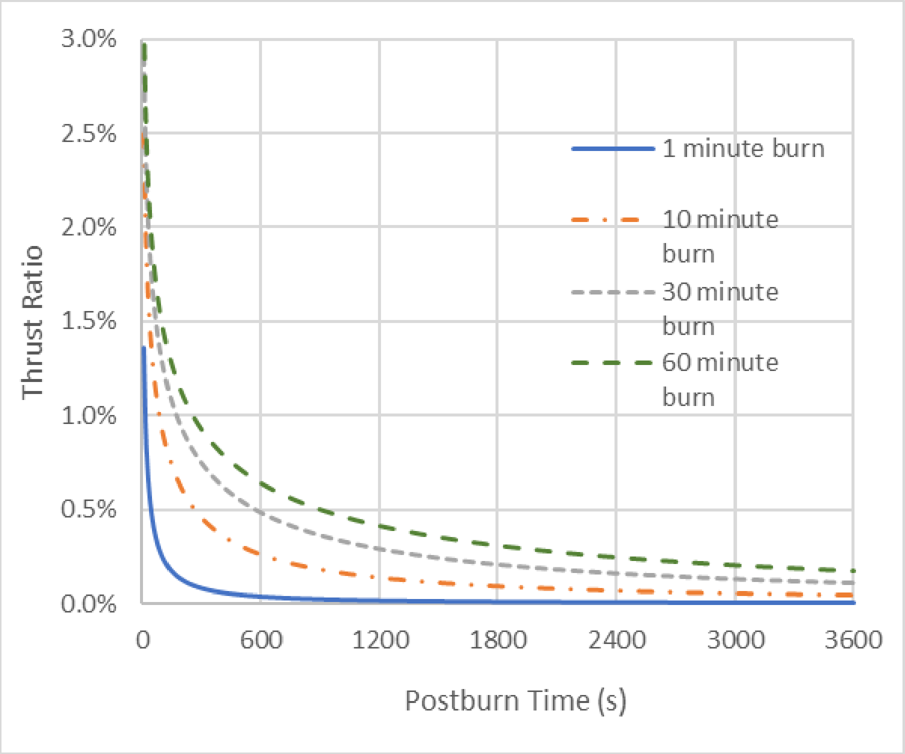


Fig. 39. Ratio of Thrust Produced by Decay Heat to Full Reactor Power Thrust

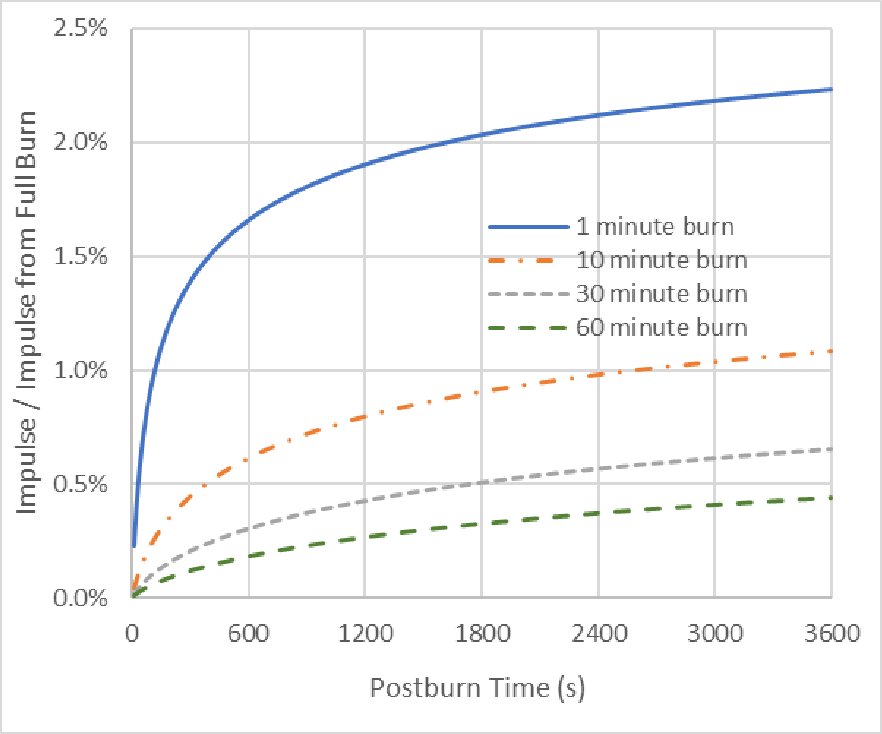


Fig. 40. Cumulative Impulse Computed by Integrating the Thrust Over Time

The impulse calculations show the cumulative effect of generating thrust from decay heat long after the rocket reactor has been shut down. Fig. 40 shows the cumulative impulse ratio determined by dividing equation (195) by (194). If a variable area nozzle is used to harvest decay heat for one hour after the reactor is shut down, the cumulative impulse increases by 2.25% for a 1-minute burn, 1.1% for a 10 minute burn, 0.7% for a 30 minute burn, and 0.4% for a 60 minute burn. At some point in time, the decay heat will not be sufficient to damage the fuel or reactor and the flowrate of hydrogen can be halted.

The specific impulse was also monitored for this engine after shutdown, and it rose from 850 seconds during reactor operation to about 1,006 seconds after reactor shutdown. The specific impulse was fairly independent of the length of the burn. This increase in I_{sp} is likely caused by the finite size of the exit of the nozzle. The optimal theoretical exit area for a de Laval nozzle in the vacuum of space is infinite, so a compromise value was used for Pewee where the exit area to throat area ratio was 300:1. When decay heat is used, the mass flowrate of hydrogen is much lower than during reactor operation and, subsequently, the throat area is smaller. This significantly raises the ratio of exit area to throat area since the exit area is constant. For the flowrates needed to remove decay heat, the de Laval nozzle is more efficient.

4.3.1 Hydrogen Savings for Missions to Mars using Variable Area Throat Nozzle

In this analysis, two example missions outlined by NASA were analyzed to compute the hydrogen propellant savings if the variable area throat nozzles are employed. The rocket modeled for this work was based on the Space Nuclear Rocket Engines (SNRE) that are patterned after NERVA Pewee-2 rocket which have a thrust of 25,000 lbf, and a specific impulse of 900 s. Additionally, three separate engines were used. The two different missions that were analyzed [Finseth, Borowski] were the Copernicus Zero *g* mission and the Cargo, Short Round Trip to Mars. More information about these missions can be found in the tables below.

Table 4. Copernicus Zero g Mission to Mars Outline

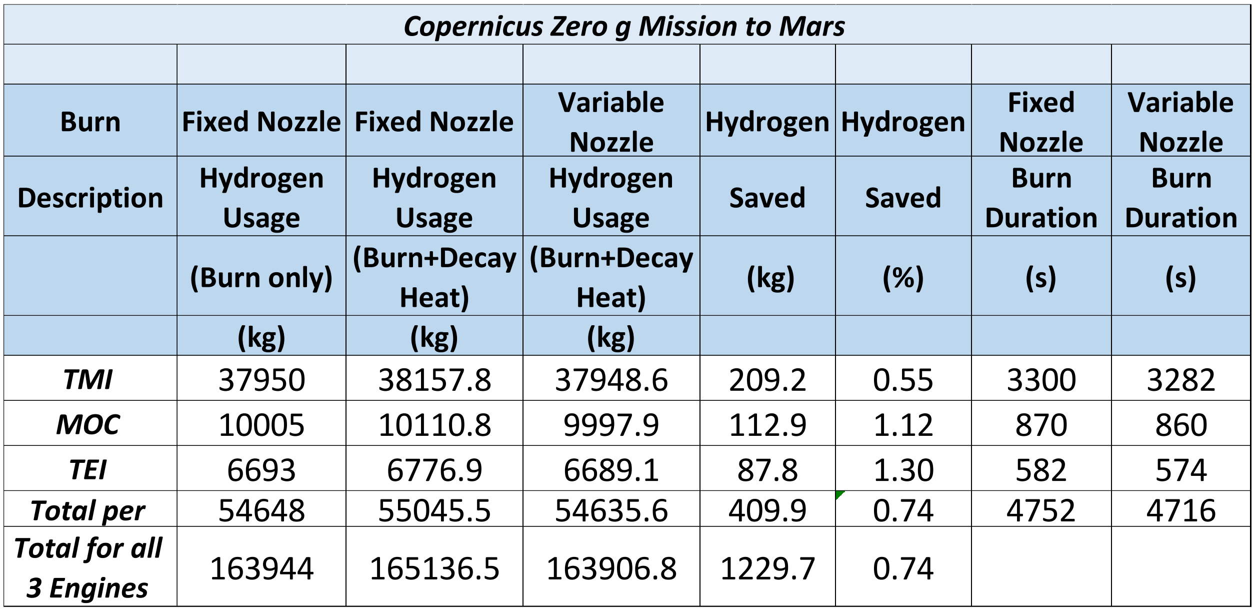
Copernicus Zero g Mission to Mars							
Burn	**Duration**	**Duration**	**Thrust per Engine**	**Thrust per Engine**	**Number**	**Impulse per Engine**	**Impulse**
	(minutes)	**(s)**	**(klbf)**	**(N)**	**of Engines**	**(GN-s)**	**(GN-s)**
TMI	55	3300	25000	111200	3	0.3670	1.1009
MOC	14.5	870	25000	111200	3	0.0967	0.2902
TEI	9.7	582	25000	111200	3	0.0647	0.1942
Total	79.2	4752				0.5284	1.5853

Table 5. Split Cargo, Short Round Trip Mission to Mars Orbit

Split Cargo, Short Round Trip Mission to Mars Orbit							
Burn	**Duration**	**Duration**	**Thrust per Engine**	**Thrust per Engine**	**Number**	**Impulse per Engine**	**Impulse**
	(minutes)	**(s)**	**(lbf)**	**(N)**	**of Engines**	**(GN-s)**	**(GN-s)**
TMI	35.2	2112	25000	111200	3	0.2349	0.7046
MOC	7.9	474	25000	111200	3	0.0527	0.1581
TEI	21.1	1266	25000	111200	3	0.1408	0.4223
Total	64.2	3852				0.4283	1.2850

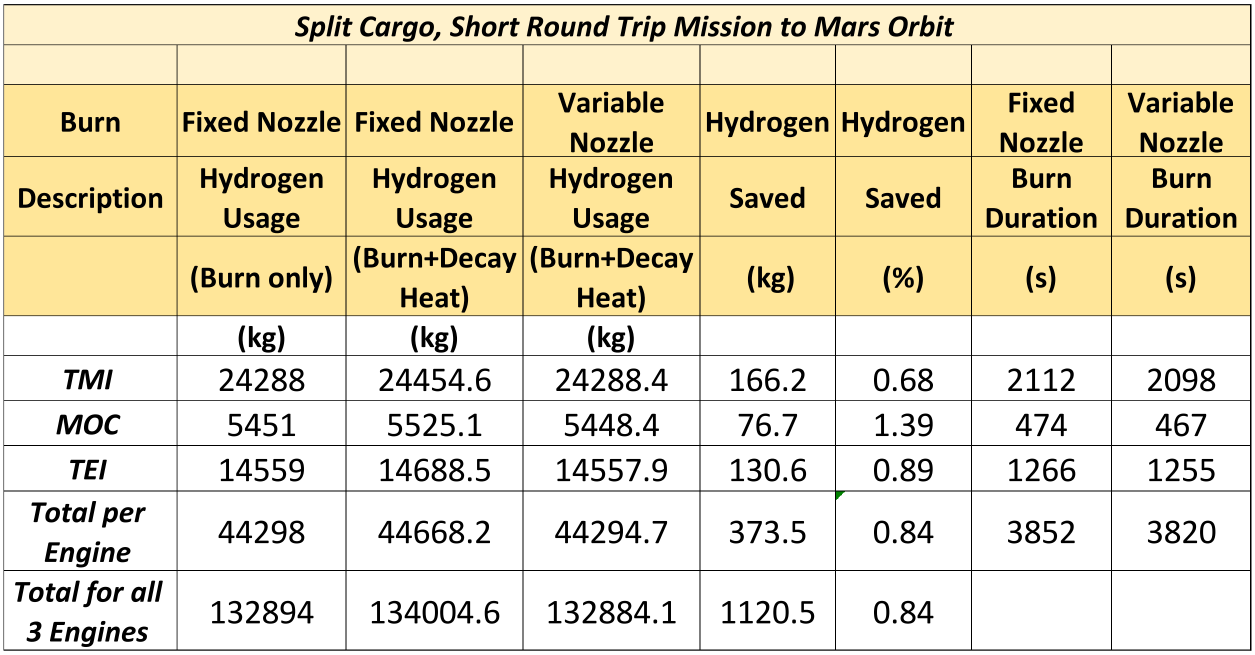
In the Copernicus Zero g Mission to Mars, variable area nozzles could save up to 0.74% (1229 kg) of hydrogen propellant, and in the Split Cargo Round Trip Mission to Mars Orbit, using variable area nozzles would save 0.84% (1120 kg) of hydrogen propellant. Hydrogen savings may be small relative to the amount of hydrogen propellant carried for each mission, but the savings could be used to increase the duration of the mission or to increase the payload.

Table 6. Hydrogen Savings for Copernicus Zero g Mission to Mars



Copernicus Zero g Mission to Mars							
Burn	**Fixed Nozzle**	**Fixed Nozzle**	**Variable Nozzle**	**Hydrogen**	**Hydrogen**	**Fixed Nozzle**	**Variable Nozzle**
Description	**Hydrogen Usage**	**Hydrogen Usage**	**Hydrogen Usage**	**Saved**	**Saved**	**Burn Duration**	**Burn Duration**
	(Burn only)	**(Burn+Decay Heat)**	**(Burn+Decay Heat)**	**(kg)**	**(%)**	**(s)**	**(s)**
	(kg)	**(kg)**	**(kg)**				
TMI	37950	38157.8	37948.6	209.2	0.55	3300	3282
MOC	10005	10110.8	9997.9	112.9	1.12	870	860
TEI	6693	6776.9	6689.1	87.8	1.30	582	574
Total per	54648	55045.5	54635.6	409.9	0.74	4752	4716
Total for all 3 Engines	163944	165136.5	163906.8	1229.7	0.74		

Table 7. Hydrogen Savings for Split Cargo, Short Round Trip Mission to Mars Orbit



Split Cargo, Short Round Trip Mission to Mars Orbit							
Burn	**Fixed Nozzle**	**Fixed Nozzle**	**Variable Nozzle**	**Hydrogen**	**Hydrogen**	**Fixed Nozzle**	**Variable Nozzle**
Description	**Hydrogen Usage**	**Hydrogen Usage**	**Hydrogen Usage**	**Saved**	**Saved**	**Burn Duration**	**Burn Duration**
	(Burn only)	**(Burn+Decay Heat)**	**(Burn+Decay Heat)**	**(kg)**	**(%)**	**(s)**	**(s)**
	(kg)	**(kg)**	**(kg)**				
TMI	24288	24454.6	24288.4	166.2	0.68	2112	2098
MOC	5451	5525.1	5448.4	76.7	1.39	474	467
TEI	14559	14688.5	14557.9	130.6	0.89	1266	1255
Total per Engine	44298	44668.2	44294.7	373.5	0.84	3852	3820
Total for all 3 Engines	132894	134004.6	132884.1	1120.5	0.84		

4.4 Conclusions

The addition of a variable area nozzle to a nuclear thermal propulsion rocket will be complicated since the temperatures in the reactor chamber typically exceed 2,700 K. The reactor has sufficient uranium fuel to heat much more hydrogen than can be held in the rocket's storage tanks, so hydrogen propellent is the limiting factor in the use of nuclear rockets for distant missions into space.

A UNLV senior design team worked on the design and testing of a variable area rocket nozzle based on the theoretical work outlined in this chapter. They used a chemical rocket engine burning acrylic in pure oxygen. They used mesh blocks that form a hexagonal throat area and actuated the variation in throat diameter by turning an external ring. The variable throat did work in their experiments and a second senior design team modified the system so that the throat area could be controlled by a computer.

Experimental and theoretical results prove that although the design may be difficult to implement, it is possible. The results also suggest adding a variable area nozzle to save hydrogen is worth the expense to allow more cargo to be carried on space missions, to decrease the time in transit to Mars, or to extend the mission flight plan.

CHAPTER 5

MOLTEN URANIUM CARBIDE CORE MODEL

5.1 Introduction

Nuclear thermal rockets (NTR) promise a significant performance enhancement over chemical rocket technology for space exploration. Heat generated by a nuclear fission reactor is employed to heat hydrogen propellant, and the gas is then expanded through a DeLaval nozzle to produce thrust. The specific impulse of nuclear rocket engines tested through NERVA were double that of the Saturn V rocket used to launch humans to the moon. Performance of NTR can be further increased by designing the reactor to operate at higher temperatures. Materials with higher melting temperatures are required, and a new engine design based on encapsulated molten uranium carbide fuel can be used to achieve a specific impulse 27% higher than the solid fuel rockets tested under Project Rover.

To increase the performance of NTP engines, carbide ceramics that melt at temperatures of 4000 K have been studied to evaluate their nuclear and thermal properties for use in a nuclear reactor. Tantalum and hafnium carbides are candidate ceramics for use in the reactor and the high neutron absorption of hafnium is addressed. All uranium compounds melt at temperatures well below 4000 K and a reactor design is proposed that houses uranium carbide in molten form within fuel rods of $Hf_{180}C$. The low vapor pressure of uranium carbide minimizes the internal pressure within the fuel rods, and current commercial fabrication of fuel rods made from SiC present a possible method for the fabrication of HfC fuel rods. Finally, a draft reactor design based the Pewee I NTP rocket and utilizing molten UC fuel in $Hf_{180}C$ was tested using MCNP to verify that this design could be made critical.

5.2 Theory

Improvements in the NERVA/Rover design for an NTP rocket have been recommended since experiments verified the feasibility of using NTP in the 1970's. The equation for specific impulse, I_{sp}, suggests ways to make further improvements to the technology.

$$I_{sp} = \frac{thrust}{\dot{m} \cdot g_{earth}} = \frac{1}{g_{earth}} \sqrt{\frac{2kR}{(k-1)} \frac{T_o}{\widehat{M}} \left(1 - \left(\frac{p_e}{p_o}\right)^{\frac{k-1}{k}}\right)} \tag{196}$$

The choice of hydrogen gas with $\widehat{M} = 0.0020157$ kg/mol is the smallest molecular weight of any gas and the pressure ratio, $\frac{p_e}{p_o}$, is very small for an optimally designed nozzle. The remaining term that can be enhanced is T_o, the reactor exit stagnation temperature of the hydrogen gas. In Project Rover, the Pewee engine was tested with a total burn time of about one hour. With a thrust of 25,000 lbf, the specific impulse of this engine approached 900 seconds and was limited by the maximum propellant temperature of 2500 K, about 100 degrees below the melting temperature of the (U, Zr) C fuel.

Higher specific impulse can be achieved by increasing T_o, as shown in Fig. 41. The melting temperature of uranium fuels vary with chemical composition, as shown in Table 8.

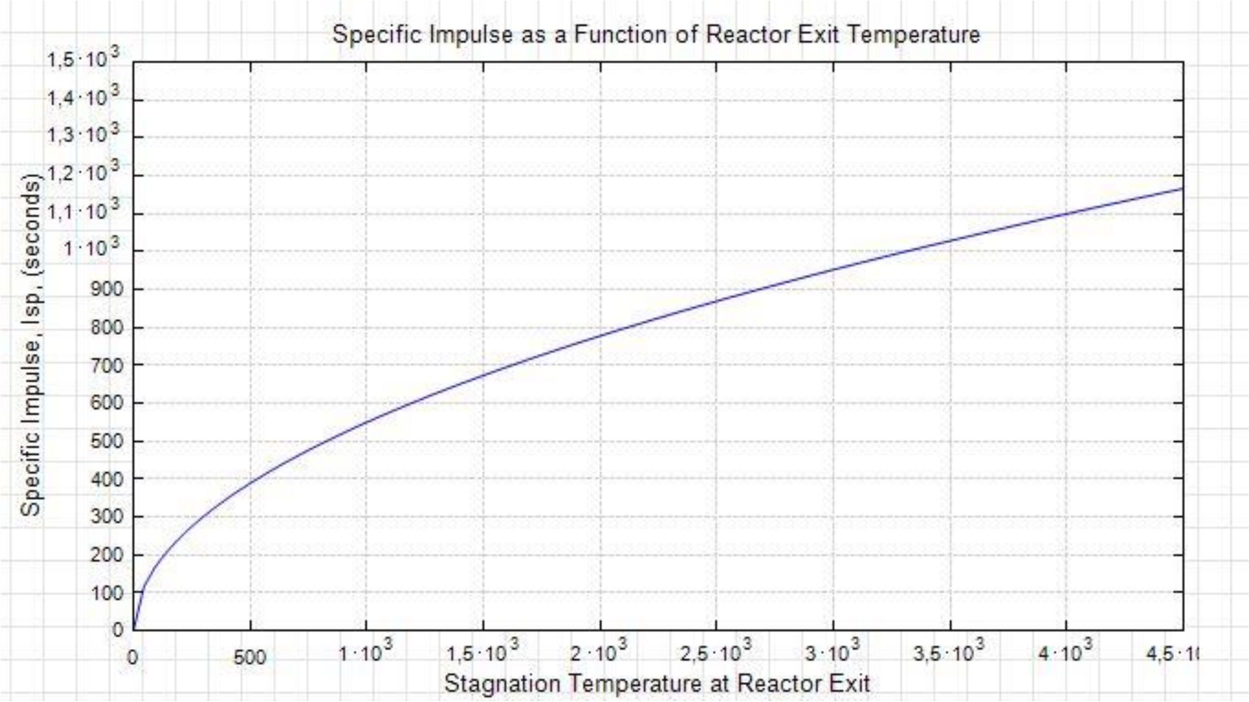


Fig. 41. Specific Impulse for a de Laval Nozzle as a Function of the Reactor Exit Temperature

Table 8 Melting Temperature of Ceramic Uranium Fuels

Fuels	**Melting Temperature**
UO_2	3138 K
UC	2620 K
UN	1170 → 1370 K

To achieve higher temperatures, molten uranium fuel can still be used if it is encapsulated in a container that can withstand the higher temperatures. To prevent melting of the fuel container and internal reactor components, high melting temperature ceramics can be employed. Fig. 42 shows the melting temperature of nine carbides, including uranium carbide fuel. Silicon carbide has a high thermal conductivity and has been used to fabricate fuel pins for reactor use. The highest melting temperature materials include refractory compounds of hafnium and tantalum. The highest melting temperature ever recorded is hafnium tantalum carbide, Ta_4HfC_5 at 4,263 K. This ceramic may make a good candidate for high temperature NTP reactors, but the neutron absorption cross-section for tantalum is high and hafnium is a neutron poison due to its extremely high cross-section. Hafnium nitrogen carbide theoretically has the highest melting temperature of any substance, but it has not yet been fabricated or tested.

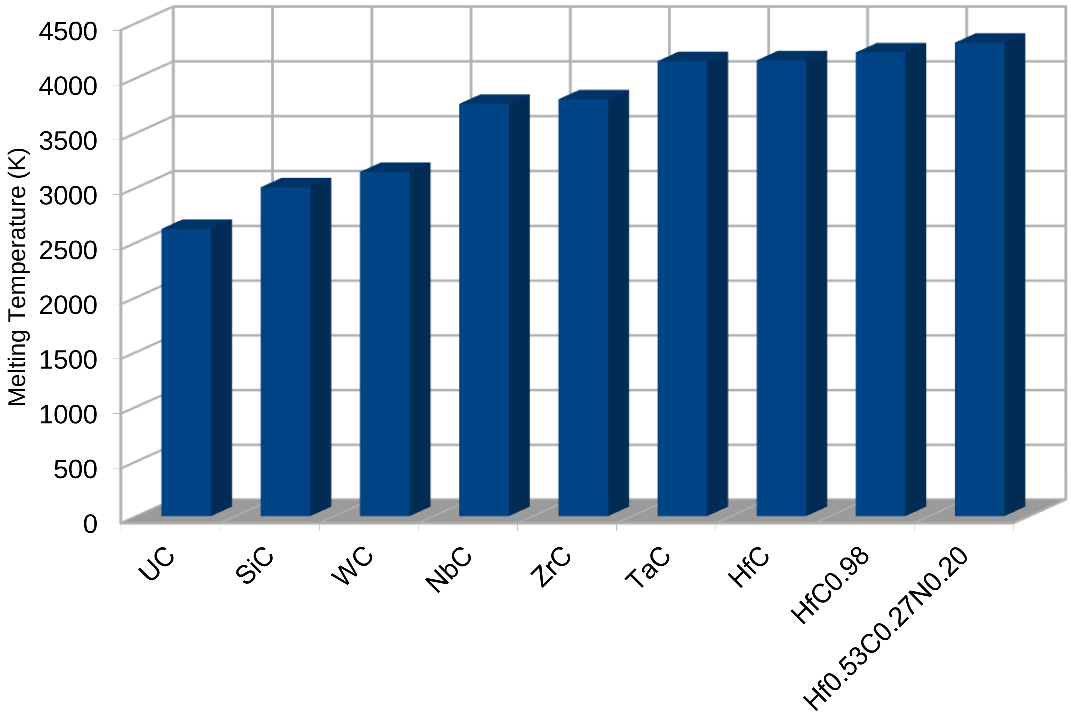


Fig. 42. Melting Temperatures of Candidate Carbides for Use in NTP

5.3 TaC or $Hf_{180}C$ for Use as NTP Fuel Pin/Ampule

Both tantalum and hafnium carbide could be used to make fuel rods capable of containing molten uranium fuel at very high temperatures. The candidate fuel is uranium carbide which melts at 2623 K and has a low vapor pressure in the liquid phase. Creating fuel elements out of TaC or $Hf_{180}C$ would be a challenge, but inspiration can be drawn from the work done by General Atomics where they developed the SIGA process to create complex components out of SiC (silicon carbide) for high temperature applications. They also developed a proprietary process to seal SiC fuel rods and join SiC to other materials. They can fabricate tubes from 5 to 50 mm in diameter and use sintered SiC fabricated from a ceramic fabric. General Atomics has made SiC fuel elements designed to withstand temperature of over 1970 K, far above the operating temperature of current metal alloy fuel rods.

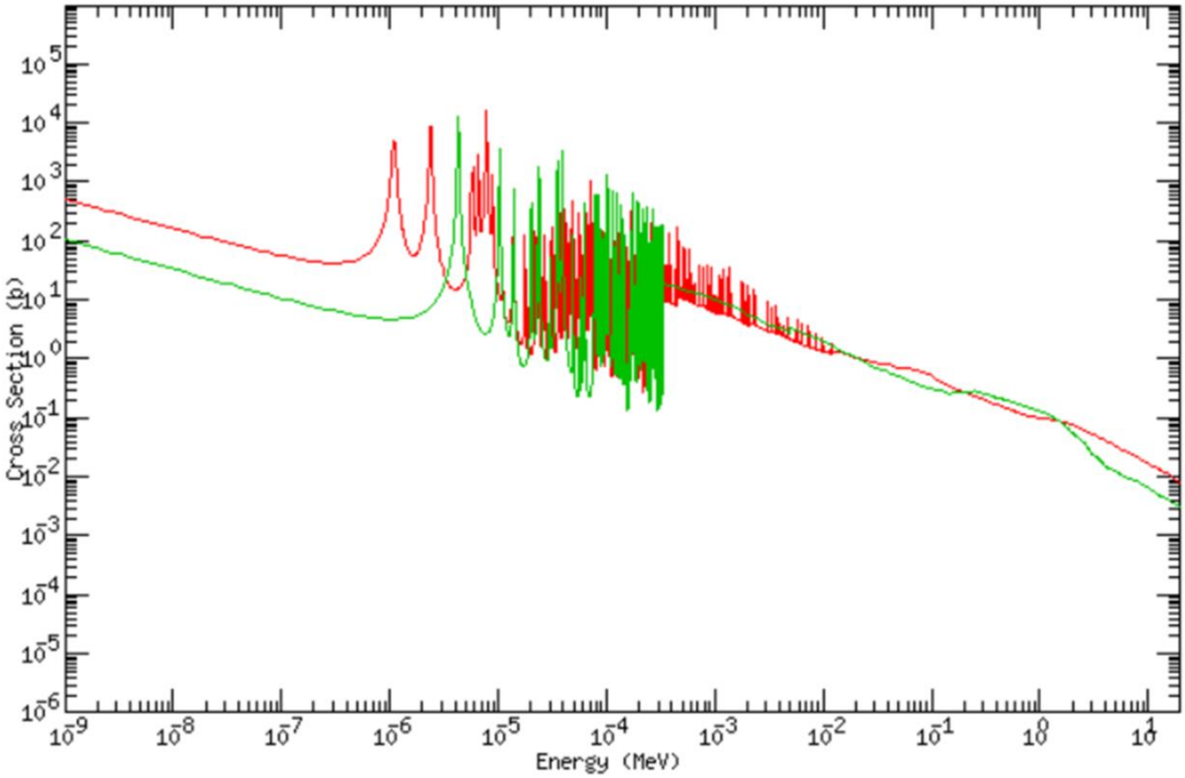


Fig. 43. Radiative Capture Cross-Sections for Natural Ta (green) and Hf-180 (red) [KAERI]

The neutron absorption cross-section for tantalum is 20.67 b, much higher than zirconium (0.185 b) or carbon (0.0032 b) used in conventional NTP reactors. This high cross-section makes it difficult to design a critical reactor with tantalum present. Hafnium is a stronger neutron absorber and is often used in reactor control rods to capture neutrons

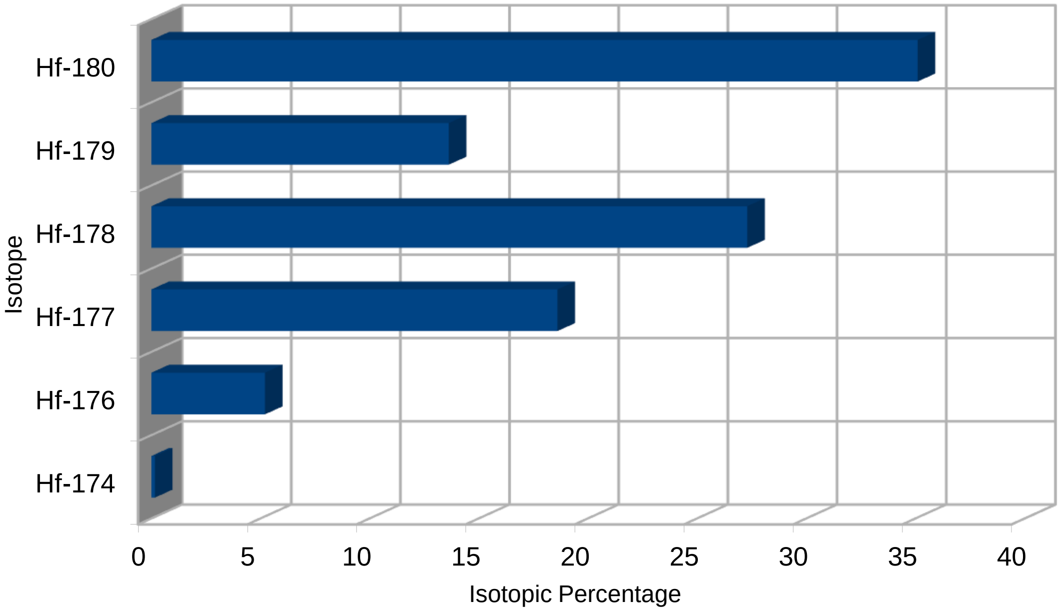


Fig. 44. Isotopic Abundance in Natural Hafnium

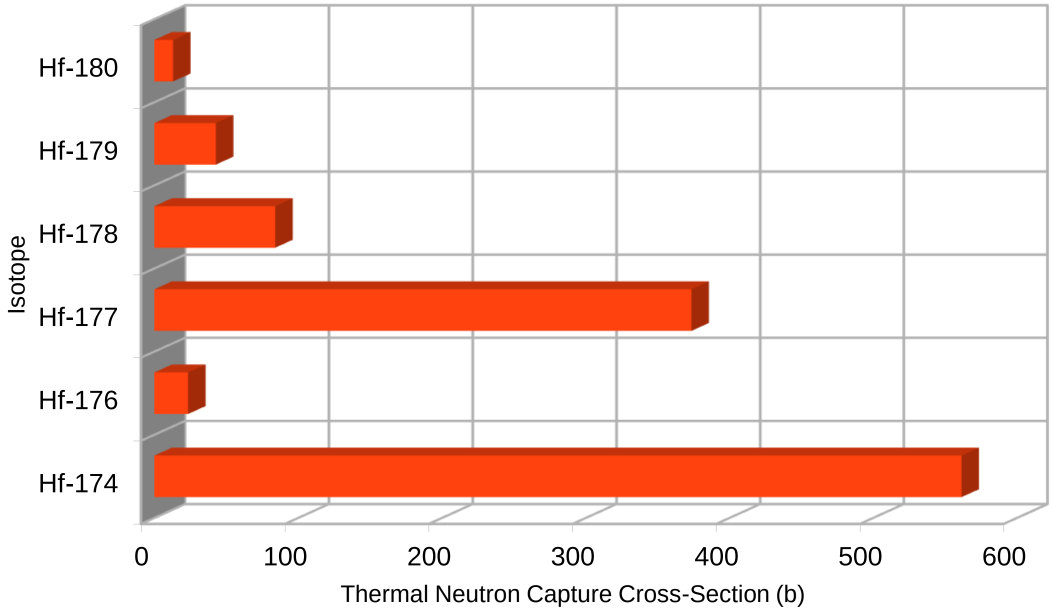


Fig. 45. Radiative Capture Cross-Sections for Natural Hafnium Isotopes

As shown in Fig. 44, natural hafnium is composed of six isotopes and their relative absorption cross-sections are listed in Fig. 45. Hafnium-180 has a relatively low absorption cross-section (13 b) and makes up 35.09% of all naturally occurring hafnium.   Although a very expensive substance, Hf-180 is commercially available in small quantities and production of larger quantities may be possible. In this concept for a molten uranium carbide reactor, $Hf_{180}C$ would be made into fuel rods, possibly using the process developed by General Atomics. Uranium carbide placed within the sealed fuel rods, or ampules, would melt in the reactor during normal operation and sustain fission. The thermal conductivity of $Hf_{180}C$ is 22 W/m-K, similar to that of TaC and ZrC. Due to the high melting temperature of the $Hf_{180}C$ (4,170 K) compared with UC (2,623 K), the hydrogen propellant flowing through the reactor could reach temperature higher than 4,000 K resulting in a significant increase in specific impulse (20%) to 1,122 seconds for an NTP rocket when compared with solid uranium carbide fuel.

Another advantage to this design may be offered by the UC fuel within each $Hf_{180}C$ ampule. Convection currents within the UC would help reduce the steep temperature gradients that occur in solid uranium ceramic fuels. This may allow the maximum propellant temperature to approach the melting temperature of the ampule wall resulting in increased specific impulse and a uniform axial temperature distribution within the fuel. It may be noted that the axial temperature gradient in the Pewee I engine was likely the leading cause of fuel coating loss within the reactor during operation.

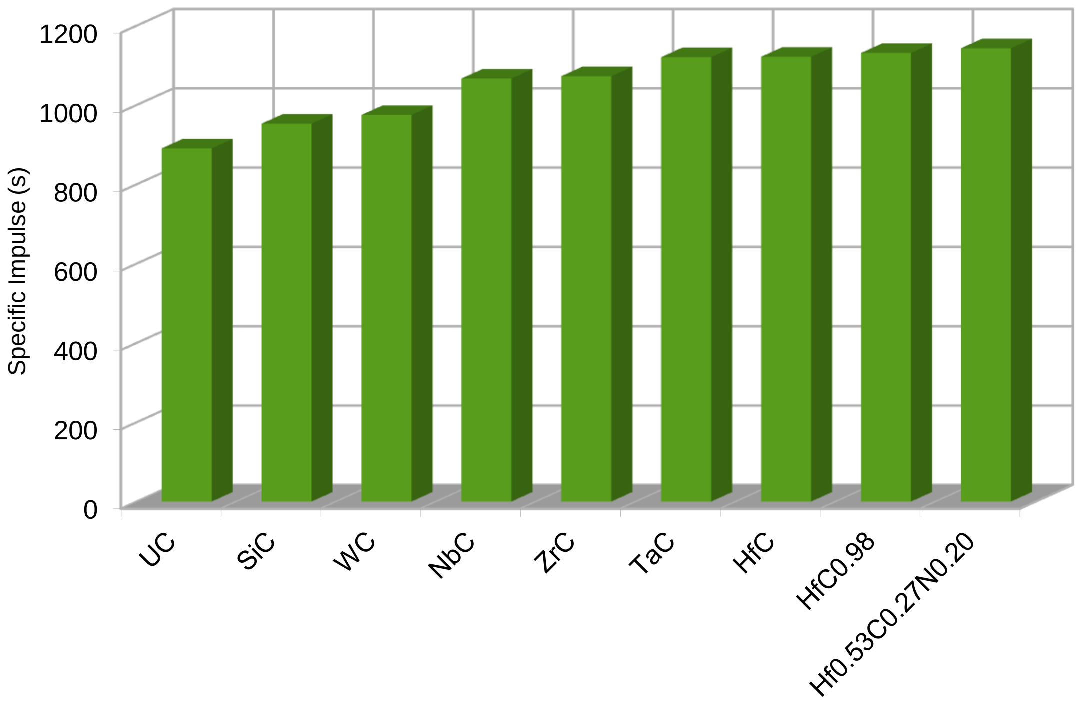


Fig. 46. Maximum Specific Impulse Possible with Various Carbides

5.4 Molten Fuel NTP Reactor

A preliminary design is proposed for the arrangement of coolant channels and fuel elements in a reactor that is similar in design to Pewee I. $Hf_{180}C$ fuel rods filled with 93.1% enriched UC are housed in the reactor. Hydrogen propellant flows around the cylindrical fuel elements and serves as both a moderator and a coolant. A critical reactor design in MCNP is shown in Fig. 47 with the following specifications:

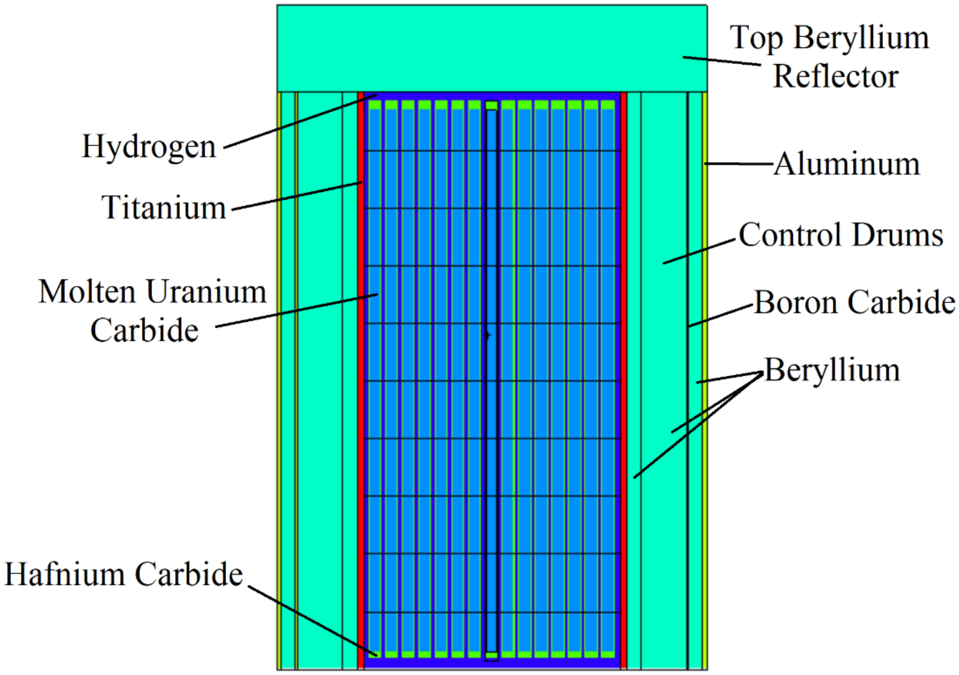


Fig. 47. Vertical Cross-Sectional Image of a Molten Fuel Thermal Reactor

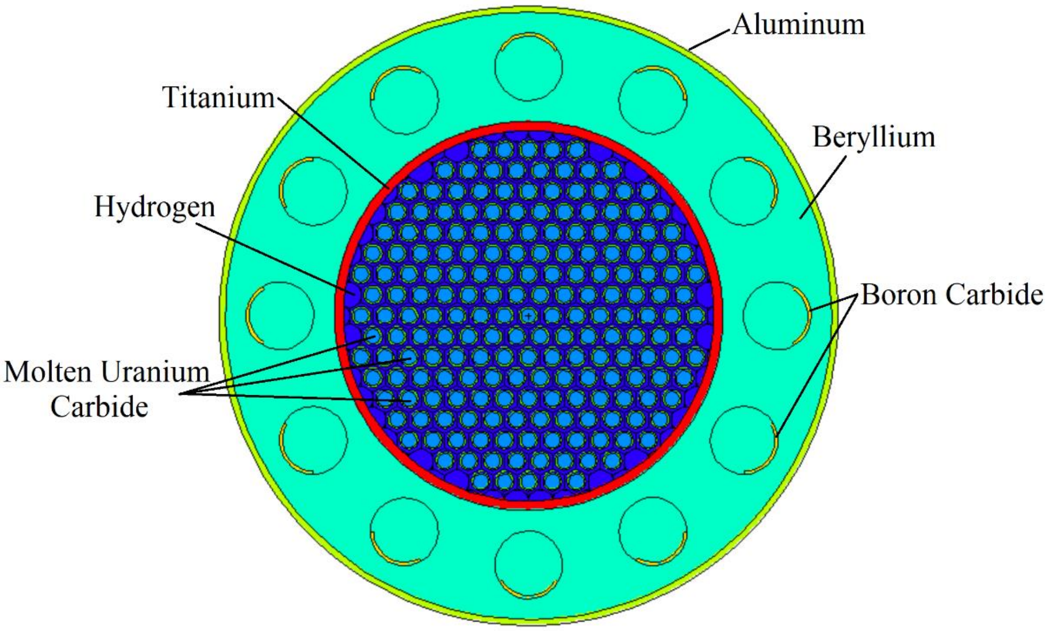


Fig. 48. Horizonal Cross-Sectional Image of the Molten Fuel Thermal Reactor Core

5.5 Performance Evaluated with UNLV-SNRE.c

To analyze the performance of a rocket powered by a molten core reactor, the computer program UNLV-SNRE was modified with values for hafnium carbide cylindrical ampules surrounded by hydrogen propellant. The reactor geometry would have a design similar to a conventional pressurized reactor where water flows around cylindrical fuel pins that are constrained within a fuel assembly. The uranium carbide fuel is contained within hafnium carbide ampules that function as fuel pins, and a triangular pitch was chosen, as shown in Fig. 49. Thermodynamic properties of HfC and UC were added to the computer program [Pioro, Sanders].

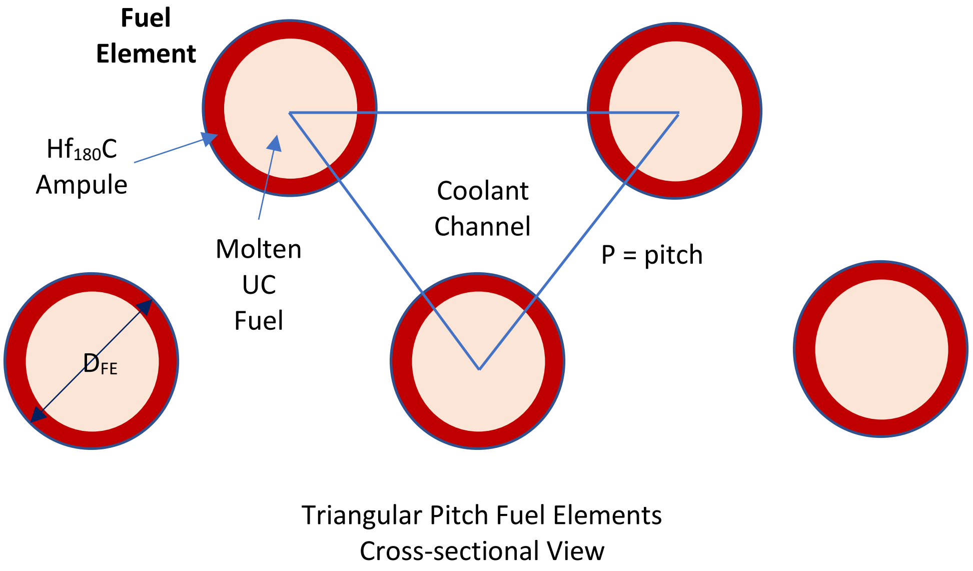


Fig. 49. Triangular Pitch Fuel Ampules in a Molten Core Reactor

Table 9. Thermal Properties of Hafnium and Uranium Carbide

	Density	**Thermal Conductivity**	**Melting Temperature**	**Ultimate Strength**	**Expansion Coefficient**
	kg/m^3	**W/m-K**	**K**	**MPa**	**1/K**
UC	13630	21.3	2780		6.00E-06
HfC	12700	22	4163	96-291	8.00E-06

For ampules in a triangular pitch, the hydraulic diameter, cell area, fuel element area, and coolant channel areas are given in equations (197) through (202).

$$A_{cell} = \frac{\sqrt{3}}{2} P^2 \tag{197}$$

$$D_h = \frac{2\sqrt{3}\, P^2 - \pi D_{Fe}^2}{24\,(P - D_{FE}) + 4\,\pi\, D_{FE}} \tag{198}$$

$$\frac{A_{FE}}{A_{cell}} = \frac{\pi}{2\sqrt{3}} \left(\frac{D_{FE}}{P}\right)^2 \tag{199}$$

$$\frac{A_{cc}}{A_{cell}} = 1 - \frac{\pi}{2\sqrt{3}} \left(\frac{D_{FE}}{P}\right)^2 \tag{200}$$

$$\frac{A_{cc}}{A_{cell}} = 1 - \frac{\pi}{2\sqrt{3}} \left(\frac{D_{FE}}{P}\right)^2 \tag{201}$$

$$A_{FE} = \frac{\pi}{8} D_{FE}^2 \tag{202}$$

The thermal conductivity of solid uranium carbide was also used in the program for temperatures below the melting point of the fuel [Pioro].

$$\begin{gathered} k_{UC}(T) = \big(19.5 + 3.57 * (T(K) - 1123.15\ K)\big)\ \frac{W}{m^2K}, \\ 800\ K < T < 2000\ K \end{gathered} \tag{203}$$

5.6 Results

A draft design for the molten fuel reactor was developed and based on the Pewee reactor model. Pewee generated 25,000 lbf (0.1112 MN) of thrust using 503.9 MW of heat produced by nuclear fission. The specific impulse was 900 seconds. The reactor was small with a length of 1.32 m and a diameter of 59 cm. Pewee was viewed by the NERVA engineers as an ideal candidate for space exploration due to the size of the reactor and the thrust generated. For larger rockets, multiple Pewee engines could be used.

The melting temperature of hafnium carbide has been reported to be about 4,163 K with some estimates as high as 4,200 K. Pewee was operated with the peak fuel temperature maintained 100 K below the melting temperature of the uranium carbide fuel of 2,780 K. Following this guideline, the molten fuel reactor was simulated to reach no more than 4,100 K and the performance was evaluated using another modification of the UNLV-SNRE.c computer program. A design based on Pewee was tested using MCNP, a Monte Carlo neutron and subatomic particle tracking computer program developed by the Los Alamos National Laboratory. The design parameters are listed in the table below, and k_{eff} = 1.06047 with a standard deviation of 0.00046 or 0.04%.

Table 10. Input Properties for the Candidate Molten Fuel NTP Core

dm/dt	**Reactor Power**	**Reactor Diameter**	**Reactor Length**	**Ampule Diameter**	**Ampule Pitch**	**Ampule Wall Thickness**	**de Laval Nozzle Exit Diameter**	**Reactor Inlet Pressure**
kg/s	**MW**	**m**	**m**	**cm**	**cm**	**cm**	**m**	**MPa**
9.813	503.9	0.59	1.32	3	3.81	0.3	1.87	5.56

The UNLV-SNRE code used the data in this table to determine the rocket performance. The mass flowrate of hydrogen through the reactor core was adjusted so that the peak fuel temperature reached 4,100 K. The final mass flowrate was 9.813 kg/s of hydrogen flow, compared to 12 kg/s for the simulation of the Pewee reactor. The propellant reached a temperature of 3,274 K at the exit of the reactor, which dropped to 380 K at the exit of the de Laval nozzle. This design produced a specific impulse of 1,069 seconds for an improvement of 18.8% over Pewee which used solid fuel. The temperature profile for the bulk hydrogen flow and the fuel along the axis of the reactor are shown in the figure below.

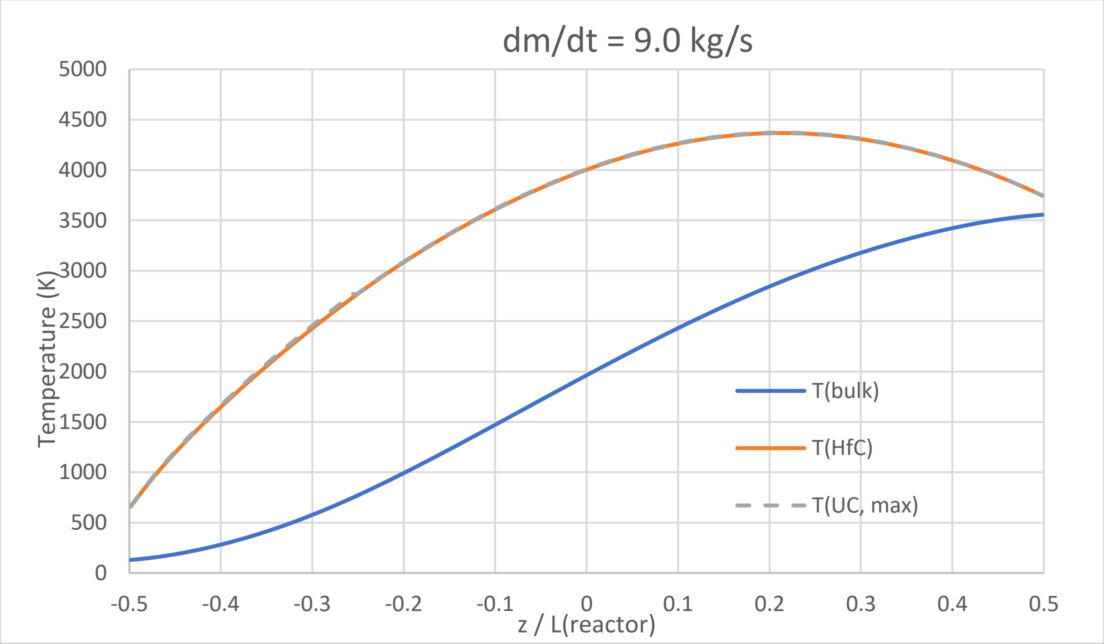


Fig. 50. Axial Temperature Distribution in the Fuel (UC), the Ampule Wall (HfC) and the Propellant (Bulk)

The peak temperature in the fuel occurred at *z/L* = 0.275. In the figure, the temperature difference between the inside wall of the HfC ampule wall and the centerline temperature of the uranium carbide molten fuel are too small to see in the graph.

When the mass flowrate was modeled below a value of 9 kg/s, the maximum fuel temperature rapidly rose above 5,000 K and the entire reactor would melt.

The UNLV-SNRE program computed the diameter of the throat of the de Laval nozzle to be 8.6 cm. The flow will choke to Mach 1 at the throat, so the maximum flowrate will be limited to 9.813 kg/s. The exit Mach number of the nozzle is 7.34, the exit velocity was 10,400 m/s, and the thrust is 23,138 lbf, similar to the value for Pewee.

The diverging nozzle diameter of 1.87 m is limited by the size of a nozzle that can be carried into space. Theoretically, the ideal nozzle would have an infinite diameter to optimally eject hydrogen into the vacuum of space. The exit pressure of this nozzle is 318 Pa, which is sufficiently low to indicate that little improvement would be gained by using a larger exit nozzle.

How sensitive is the performance of this molten fuel reactor to mass flowrate? The graph below explores small changes in the mass flowrate for which the rocket is designed.

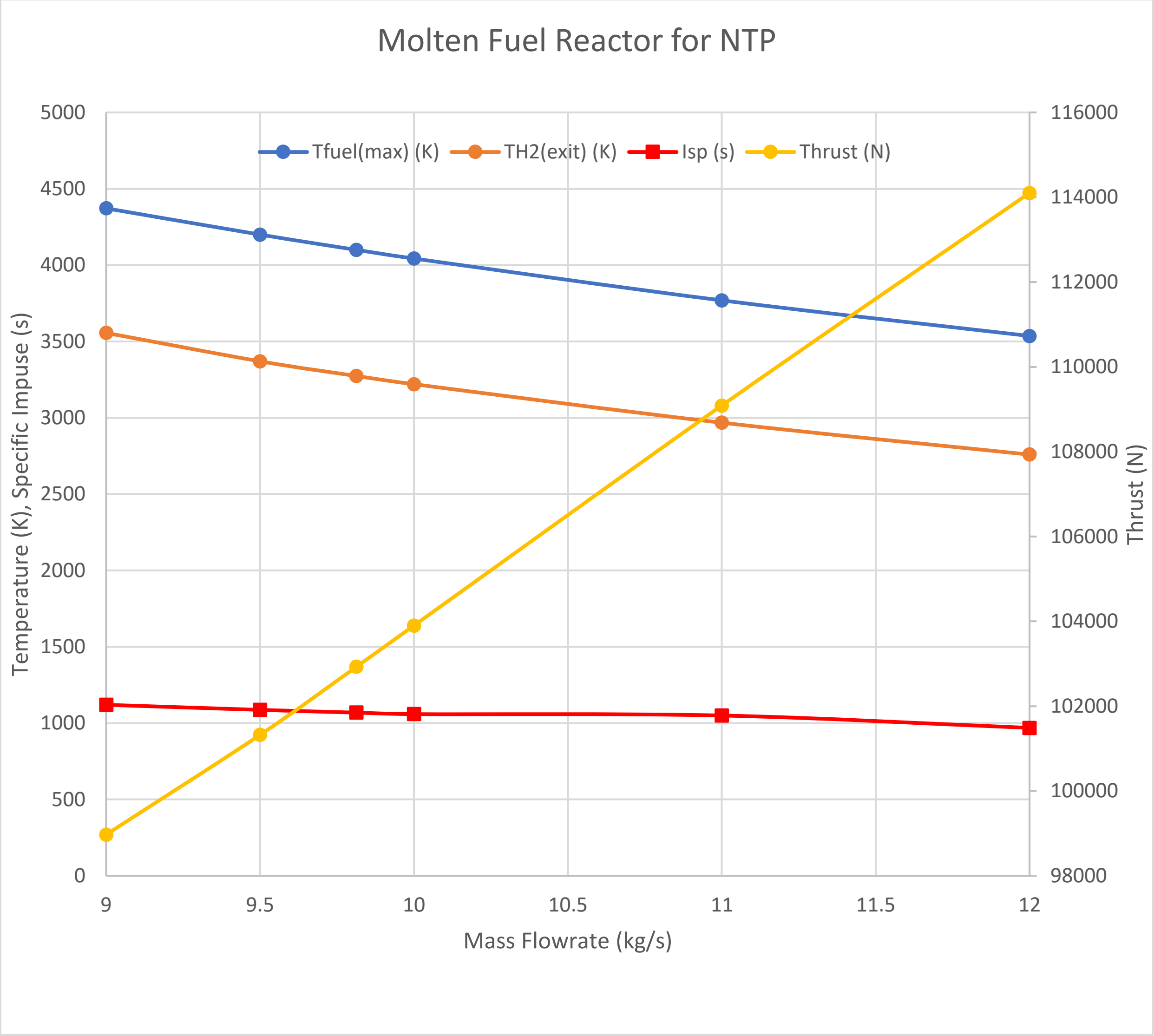


Fig. 51. Parametric Analysis of Reactor Temperatures and Specific Impulse to Mass Flowrate

Table 11. Results of the Parametric Analysis Based on Changes in Mass Flowrate

dm/dt	**T_{fuel}(max)**	**T_{H2}(exit)**	**I_{sp}**	**Thrust**
kg/s	**K**	**K**	**s**	**N**
9	4372	3556	1120	98972
9.5	4200	3370	1087	101323
9.813	4100	3274	1069	102928
10	4043	3220	1059	103901
11	3769	2968	1050	109089
12	3536	2759	969	114101

The optimal mass flowrate for the molten fuel reactor is 9.813 kg/s, as seen in Table 11, and the graph shows that the specific impulse changes little if the flowrate is shifted ±10%. As the mass flowrate is increased, the thrust increases significantly, and the temperature of the hydrogen propellant and the molten fuel decreases. If the mass flowrate is set below 8.5 kg/s, the reactor will melt.

5.7 Conclusions

A molten fuel nuclear thermal rocket is possible using modern materials. Hafnium-180 has a relatively low neutron absorption cross-section and can be used in a uranium carbide reactor with a beryllium reflector to attain criticality. The same process used by General Atomics to manufacture fuel pins from silicon carbide could potentially be adapted to the manufacture of hafnium-180 carbide fuel pins containing highly enriched uranium carbide fuel. The expansion coefficients of HfC and UC are similar, and the vapor pressure of molten uranium carbide is relatively low. Thus, rupture of the ampules during reactor operation should not be a significant problem. The buildup of fission gases could pose a problem and should be investigated. The improvement in specific impulse is significant at 18.8% when compared to the Project Rover Pewee reactor, the best performing NTP engine actually tested. This improvement in specific impulse means that additional cargo can be taken on space missions, or the missions will take less time in transit, decreasing the cosmic radiation that future astronauts will be exposed to.

CHAPTER 6

OPTIMIZATION USING A DIFFERENTIAL EVOLUTION ALGORITHM

6.1 Introduction

When nuclear thermal propulsions rockets were tested in the 1960's and 70's, computer modeling was progressing rapidly, and computer simulations aided the researchers working on Project Rover and NERVA in the design of their experiments. The classic computer used by engineering departments in 1971, for example, was the PDP VAX 11/780. The performance of this machine could reach 20 MFLOPS (millions of floating point operations per second). Today, the typical smart phone has a performance of 19,000 MFLOPS for a speed multiple of about 1,000. Computing power today allows researchers to use slower algorithms, including genetic optimization, to analyze complicated systems like the nuclear thermal propulsion rockets designs outlined in this dissertation. The molten fuel reactor reviewed in Chapter 5, for example, followed the design path shown in Fig. 52.

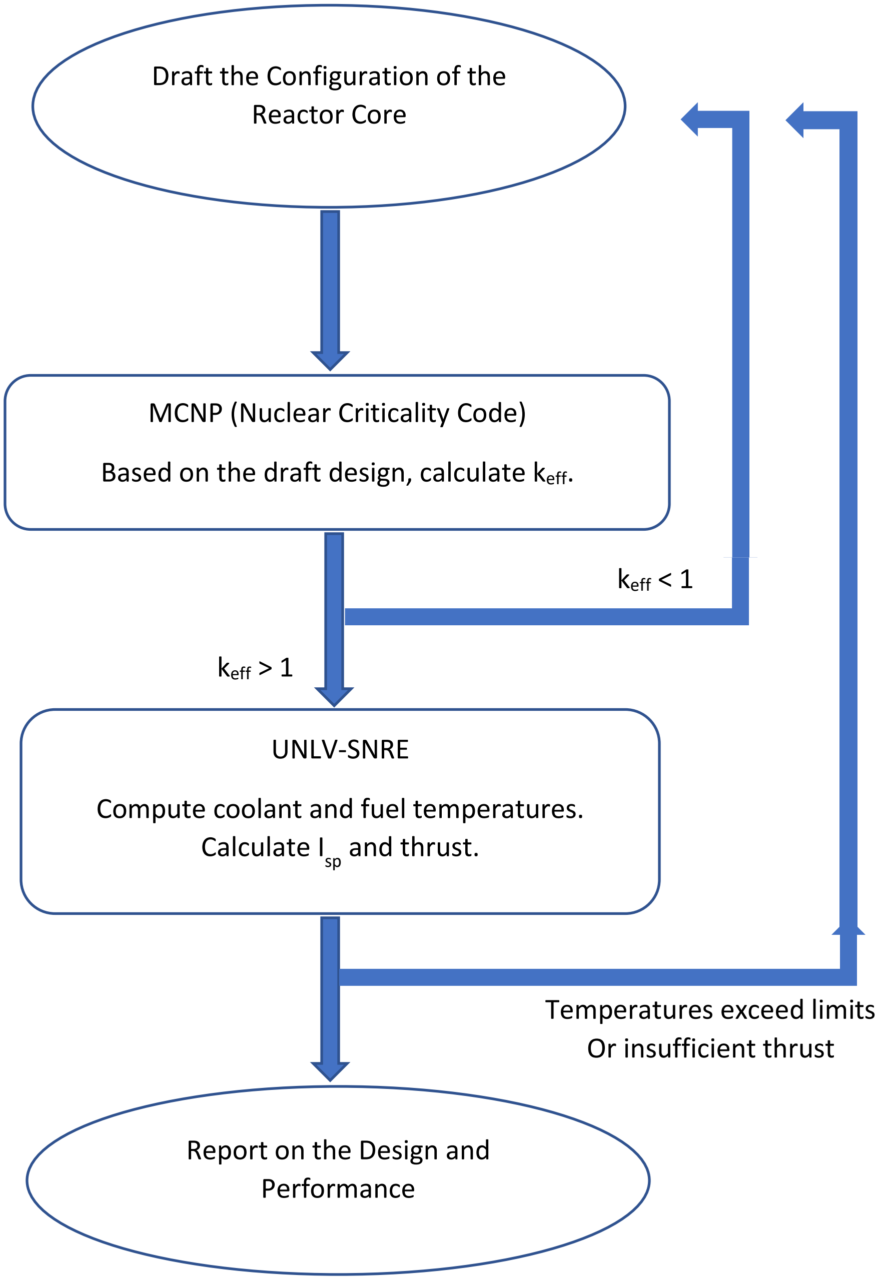

Draft the Configuration of the Reactor Core
MCNP (Nuclear Criticality Code)
Based on the draft design, calculate k_{eff}.
$k_{eff} < 1$
$k_{eff} > 1$
UNLV-SNRE
Compute coolant and fuel temperatures.
Calculate I_{sp} and thrust.
Temperatures exceed limits
Or insufficient thrust
Report on the Design and Performance

Fig. 52. Current Design Procedure

The draft configuration of the core was based upon Pewee, a successful solid reactor core that was experimentally verified. MCNP was used to determine whether the candidate design was capable of sustaining a chain reaction ($k_{eff} > 1$). This design was then passed through a heat transfer and fluid flow program, UNLV-SNRE, to model the heat transfer to the propellant and estimate thrust and specific impulse. Constraints were set on specific design variables, and the rocket thrust was set as a criterion that had to be met by the successful candidate design.

6.2 Optimization Procedure

The differential evolution optimization program discussed in section 2.7 and used in Chapter 3 was also used in this chapter to optimize the choice of variables for the design of a molten fuel NTP reactor. The key design variables for the rocket are shown in

Table *12* and 3 of these variables were chosen for the optimizer to work with: propellant mass flowrate, reactor power, and the pressure at the inlet of the reactor produced by the turbopump. The analysis was also based on the selection of an objective function that was a function of the three chosen design variables.

Table 12. Design Variables for a Molten Fuel NTP Rocket Reactor

Design Variables	**Varied in this Study**
Mass Flowrate of Hydrogen Propellant	*
Reactor Power	*
Height of Reactor Core	Fixed
Diameter of Reactor Core	Fixed
Ampule Diameter	Fixed
Ampule Pitch	Fixed
Ampule Wall Thickness	Fixed
Radial Reflector Thickness	Fixed
Axial (Inlet) Reflector Thickness	Fixed
Uranium Enrichment	Fixed
Maximum Allowable Core Temperature	Fixed
Reactor Inlet Pressure from Turbopumps	*
Diameter of de Laval Nozzle Exit	Fixed
Core Material Properties	Fixed

$$f_{objective} = f\left(\frac{dm}{dt}, \qquad P_{reactor}, \qquad p_{turbopump}\right) \tag{204}$$

Three different objective functions were chosen, and the differential evolution algorithm was set up to minimize the function by changing the values of the three design variables. The objective functions were chosen to maximize the specific impulse of the engine, maximize the thrust, and minimize the reactor power required.

The optimizer also required limits on each design variable and other parameters determined by UNLV-SNRE. These constraints are given by the table below.

Table 13. Constraints on the NTP Design to be Optimized with the Thrust set as a Criteria

Variable	Maximum Temperature of HfC Ampule Wall	Thrust	Reactor Inlet Pressure from Turbopump	Mass Flowrate of Hydrogen Propellant	Reactor Power
Units	K	MN	MPa	kg/s	GW
Lower Limit	0	0.1121	0	0	0
Upper Limit	4100	No Limit	13.8	No Limit	No Limit

The HfC ampule wall was limited to 4,100 K in temperature, about 100 degrees lower that the melting temperature of the material. The thrust was required to be no lower than 0.1121 MN which corresponds to the 25,000 lbf thrust generated by Pewee. In Chapter 5, the computed thrust was slightly lower than this constraint.

6.3 Analysis and Results

The results of the optimization study are shown in Table *14* and Table *15* below. The three objective functions are shown as cases A, B, and C. Table *14* lists case A with the most complicated objective function. The objective function took 61 generations of candidate agents to reach a stable solution. Case B used reactor power divided by the specific impulse as the objective function and arrived at a solution similar to case A. Case C minimized the reciprocal of the specific impulse and produced the most reasonable, optimal solution. The optimal thrust, turbopump pressure, and mass flowrate of hydrogen very closely match the molten fuel reactor from Chapter 5. Case C required 10.66 kg/s of hydrogen propellant, 543 MW of reactor heat, and an inlet pressure of 10.33 MPa, well below the limit of 13.8 MPa (2,000 psi). The specific impulse was 1064 seconds, very close to the 1069 seconds for the reactor discussed in Chapter 5.

The results for cases A and B were interesting and both cases reflect that the optimizer was asked to minimize the reactor power and maximize the specific impulse. The optimizer found an easy way to do this. Provide enough heat from the reactor to vaporize the hydrogen liquid in the storage tank and allow it to pass through the reactor core. The expanding gas passes through the de Laval nozzle and generates thrust. Unfortunately, this requires very high mass flowrates, up to 20 times the flowrate for Pewee or the molten fuel reactor design of Chapter 5. The hydrogen would soon be depleted since the specific impulse only reached 174 seconds for both cases A and B. This analysis demonstrates that the objective function for NTP design should be limited to the specific impulse.

Table 14. Results from the Optimization

Case	**Objective Function (OF)**	**OF Units**	**OF Value**	**Generations to Stabilize**	**dm/dt (kg/s)**
A	$\frac{Power}{I_{sp} \cdot Thrust}$	m/s^4	0.000003	61	233.5
B	$\frac{Power}{I_{sp}}$	W/s	87.1	53	64.4
C	$\frac{1}{I_{sp}}$	s	0.00094	30	10.66

Table 15. Results from the Optimization (Cont.)

Case	Objective Function	dm/dt of H_2	Reactor Power	Pressure from Turbopump	I_{sp}	Thrust
		kg/s	kW	MPa	s	kN
A	$\frac{Power}{I_{sp} \cdot Thrust}$	233.5	<1	2.86	174.3	399
B	$\frac{Power}{I_{sp}}$	64.4	15	9.97	176.5	111.4
C	$\frac{1}{I_{sp}}$	10.66	5.43E05	10.088	1064	111.21

6.4 Conclusions

The use of genetic optimization did work well for the molten fuel reactor and agreed with the molten fuel reactor results of Chapter 5 where the design variables were adjusted by hand to seek an optimal solution. The optimizer could be employed to vary more of the design variables in

Table *12*, such as the fuel ampule diameter and pitch as well as the reactor height and diameter. The current optimization program requires approximately 10 minutes to arrive at a stable solution on an Intel I-5 processor.

CHAPTER 7

OVERALL SUMMARY AND CONCLUSIONS

Nuclear thermal fission rockets were designed and tested through Project Rover and NERVA in a program that culminated in a design that NASA intended to use for manned flights to Mars by 1981. While human flight to the moon ended with Apollo 17 on December 7, 1972, there is a current resurgence in interest in human flights to the moon, Mars, Ceres, and the icy moons of Saturn and Jupiter. The purpose of this work was to apply modern engineering methods to enhance the NTP engines tested in the 1970's, mitigate problems that the engineers of that time discovered, determine if the tested designs were optimal, and move the technology forward by applying new materials to increase NTP performance.

The first problem that was investigated involved fuel corrosion of the uranium carbide experienced in the NERVA tests. The engineers at the time of testing changed the niobium carbide coating to zirconium carbide which significantly reduced corrosion. For this work, the published corrosion data was combined with the NTP rocket engine computer model to see if thermal gradients, fuel temperature, or hydrogen propellant temperature were responsible for this corrosion. By applying differential evolution optimization, it was determined that the axial temperature gradient was principally responsible for corrosion, and the gas temperature towards the end of the reactor also contributed. By using a flatter neutron flux distribution in the reactor core, it was possible to decrease this temperature gradient and the resulting corrosion. In practice, axial variation in fuel enrichment or concentration may be used to provide this flatter flux profile.

Nuclear reactors produce vast amounts of heat that is transferred to the propellant to produce thrust, but de Laval nozzles typically have a fixed throat diameter, and this requires a specific mass flowrate of hydrogen through the nozzle to produce thrust. In addition to heat produced by fission, radioactive fission products are also produced and continue to generate a significant amount of heat through radioactive decay even after the reactor burn is completed. To prevent the fuel from melting once a burn is completed, small amounts of hydrogen propellant must be passed through the reactor core and the de Laval nozzle. This propellant provides very little thrust since the mass flowrate is insufficient to choke the flow at the throat of the nozzle as it exits the rocket at subsonic velocities. Variable area rocket nozzles were proposed as a solution to this problem. The variable throat area will allow the rocket to continue producing a supersonic exhaust with significant thrust by utilizing the propellant that would normally be wasted to remove decay heat. The resulting savings in propellant can be significant. Posters and papers were presented at conferences documenting this propellant savings technique and how the technique was applied to typical space missions to Mars. The additional thrust produced from the decay heat was factored into the calculation of the total impulse required during a rocket burn. It was also found that using a variable area nozzle would save several thousand kilograms of hydrogen when compared to a fixed nozzle for missions to Mars. This study also influenced two senior design projects at UNLV where students built and tested variable area nozzles for a bench-mounted chemical rocket engine burning acrylic in pure oxygen. The projects were both successful and the second project used a microcontroller to vary the throat area. The variable area throat was constructed from interlocking jaws machined from 304 stainless steel.

Investigations into improving existing NTP technology led to a study of methods to move beyond the temperature limitations of the NERVA rocket designs. Uranium carbide melts at about 2700 K, and the Pewee rocket tested at the Nevada Test Site heated hydrogen propellant to that same temperature. Solid fuel NTP engines are limited by the melting temperature of the fuel and the engine components. The current material with the highest melting temperature is hafnium tantalum carbide at 4200 K. Hafnium has a very high neutron absorption cross-section and could not directly be used in large quantities in a reactor. Fortunately, natural hafnium is composed of five different isotopes and the most abundant isotope has a relatively low cross-section. While it is an expensive proposition, Hf-180 can be isotopically separated from the other isotopes in natural hafnium and be used to make fuel rods containing the uranium carbide fuel. The fuel will melt within each fuel rod during rocket operation but allows the specific impulse of an NTP rocket to increase from 900 seconds to about 1100 seconds. This 18.8% improvement in performance would allow rockets to reach Mars quicker or decrease the use of hydrogen propellant by the rocket.

Finally, the Rover and NERVA engineers did outstanding work in producing NTP rocket designs. Fermi tested the first nuclear reactor built by humans in 1942 and by 1968, the basis for the NERVA rocket for missions to Mars was designed, built, and tested.

7.1 Future Work

The optimization program written for this project can be employed to further optimize NTP engine designs. Two areas for future work include axial variation of the uranium enrichment or the concentration of uranium carbide in a reactor to maximize specific impulse and minimize reactor mass. For the same reason the program can be used to determine if coolant channel diameter can be varied along the axis of the reactor to maximize performance. Project DRACO funded by the U.S. government will involve testing of new NTP rocket designs based on the NERVA technology and I hope that the work done for this project will help increase the performance of new NTP designs and bring us one step closer to exploration of the stars.

APPENDIX A

COMPUTER PROGRAMS

A.1 Computer Programs and Data Files Used for Corrosion Studies

Program DE-DATAFIT.bas

This program, written in FreeBasic, uses differential evolution optimization to accomplish a non-linear curve fit of corrosion data to experimental data. The objective function used in the program is the standard error between the original experimental curve and the curve fit predicted by the program.

```
dim NN as integer
    NN = 1000  '  Maximum number of data pairs.

 dim as double x, y, z, u, response, deviation
 dim as double column(6, NN)
 dim eta_max as double
 dim maxorder as integer, number_data_pairs as integer
 dim i as integer, j as integer, k as integer
 dim NP as integer, D as integer
 dim a_upper as double, a_lower as double
 dim F as double, CR as double
 dim minimum_cost as double, index_min_cost as integer
 dim best_index as integer
 dim generation as integer, max_generations as integer
 dim r1 as integer, r2 as integer, r3 as integer
 dim as integer n, L
 dim as double sum, trial_cost, mean_cost, cost_standard_error

 dim as integer debug, right_index
 dim as double  tolerance
 dim as string  input_line
 dim as double  start_time, stop_time

 dim as double aa


 declare function objective_function(a()               as double,  _
                                     number_data_pairs as integer, _
                                     column()  as double)  as double
```

```
'------------------------------------------------------------------------
'
'  PROGRAM:  DE-NEW.BAS
'
'  PURPOSE:  This program is developed to calculate optimal coefficients
'            in a vector "a" based on a differential evolution algorithm.
'            The program reads data from an input file.  The program
'            will prompt the user for an input filename or you can add
'            it at the command line as:  "DE-FINAL input.txt"
'
'  Input:    1)  data.dat - file containing pairs of data to be fitted.
'            2)  input.txt - user data input file.
'            3)  external function containing the objective function
'                for the run.
'
'  Output:   1)  text to screen containing results of the run.
'            2)  optional debugging output as text to screen.
'
'  Version:  5, 10/20/2018, WGC KG.
'            Updated 6/1/2022, KG and WGC.
'            Copyright (2022) William Culbreth and Kimberly Gonzalez.
'
'-------------------------------------------------------------------------

      dim as string  input_file, data_file, answer


   '----------------------------------------------------------------------
   '
   '  A.  Read in initial data.
   '
   '----------------------------------------------------------------------


     '  A.1  Read in user data from 'input.txt.'

      input_file = "input.cor"
      print "debug = "; debug


      open input_file for input as #1

      do while(eof(1) = 0)

         line input #1, input_line

         right_index = len(input_line) - 70 + 1

         if mid(input_line,8,3) = "A.1" then tolerance     _
         = val(mid(input_line,70,right_index))

         if mid(input_line,8,3) = "B.1" then NP            _
         = int(val(mid(input_line,70,right_index)))
         if mid(input_line,8,3) = "B.2" then F             _
         = val(mid(input_line,70,right_index))
         if mid(input_line,8,3) = "B.3" then CR             _
         = val(mid(input_line,70,right_index))
         if mid(input_line,8,3) = "B.4" then max_generations_
         = int(val(mid(input_line,70,right_index)))

         if mid(input_line,8,3) = "C.1" then debug          _
         = int(val(mid(input_line,70,right_index)))

      loop

         close #1

         print "The input values are:  NP, F, CR, max_gen = "; _
                                       NP, F, CR, max_generations
         print " "


     '  A.2  Define the data pairs from "data_file".

      data_file = "corrosion-pewee-dim.txt"

      open data_file for input as #1

         i = 0
      do while(eof(1) = 0)

         input #1, column(0,i), column(1,i), column(2,i), column(3,i), _
                   column(4,i), column(5,i)
         print  i, column(0,i), column(1,i), column(2,i), column(3,i), _
                column(4,i), column(5,i)

         i += 1
      loop

         close #1
         number_data_pairs = i

         print "The number of items in corrosion.dat is:  "; _
                number_data_pairs

      D = 13       '  This is the number of coefficients to compute.


           if(debug = 1) then print "   number_data_pairs = "; _
                                        number_data_pairs
           if(debug = 1) then print " "



   '----------------------------------------------------------------------
   '
   '  B.  Initialization
   '
   '----------------------------------------------------------------------

      dim c(NP, D) as double, cost(NP) as double, c_vector(D) as double
      dim best_vector(D) as double, value as double
      dim as double mutation_vector(D), old_ensemble(NP,D), _
             new_ensemble(NP,D)
      dim a(D) as double


      '  B.1  Make random guesses for the solution vectors assembled
      '       into "NP" populations.

      for i=0 to (NP-1)
         for j=0 to (D-1)
            c(i,j) = (2*rnd-1)*1
            c_vector(j) = c(i,j)
         next j
            cost(i) = objective_function(c_vector(), _
                      number_data_pairs, column() )
      next i


      '  B.2  Find the minimum objective function for these
      '       guessed solution vectors.

         minimum_cost = cost(0)
      for i=0 to (NP-1)
         if(cost(i) < minimum_cost) then
            minimum_cost   = cost(i)
            index_min_cost = i
         end if
      next i


      '  B.3  Now, move the best solution vector into "best_vector"
      '       and store the index.

         for j=0 to (D-1)
            best_vector(j) = c(index_min_cost,j)
         next j
            best_index = index_min_cost


      '  B.4  Move the ensemble "c(NP,D)" into the "old" ensemble.

         for i=0 to (NP-1)
            for j=0 to (D-1)
               old_ensemble(i,j) = c(i,j)
               new_ensemble(i,j) = c(i,j)
            next j
         next i


   '----------------------------------------------------------------------
   '
   '  C.  Process through each generation of optimal solutions.
   '
   '----------------------------------------------------------------------

         start_time = timer
```

```
for generation = 0 to max_generations

   if (debug = 1)  then
      print "                                                  "
      print "Generation = ", generation;"/";max_generations;" _
             (D, NP) = ";D, NP
   end if


   '  C.1  Pick 5 random integers to refer to different
   '       populations (out of NP).


   for i=0 to (NP-1)

      do
        r1 = rnd * (NP-1)
        if (r1 <> i) then exit do
      loop

      do
         r2 = rnd * (NP-1)
         if(r2 <> i and r2 <> r1) then exit do
      loop

      do
         r3 = rnd * (NP-1)
         if(r3 <> i and r3 <> r2 and r3 <> r1) then exit do
      loop

      if (debug=1) then
         print "     i, r1, r2, r3 = ", i, r1, r2, r3
         print "                     "
      end if


   '  C.2  Complete the Differential Evolution (DE) strategy.
   '       Replace elements of the best
   '       vector to form a new mutation vector.


      for j=0 to (D-1)
         mutation_vector(j) = old_ensemble(i,j)
      next j


      n = rnd * (D-1)
      L = 0

   do
      mutation_vector(n) = old_ensemble(r1,n) + F * _
           (old_ensemble(r2,n) - old_ensemble(r3,n))

      if (debug=1) then
         print "   mutation_vector(";n;")   = ";mutation_vector(n)
         print "   best_vector(n)           = ";best_vector(n)
         print "   old_ensemble(";r2;",n)   = ";old_ensemble(r2,n)
         print "   old_ensemble(";r3;",n)   = ";old_ensemble(r3,n)
         print "   n, L, mutation_vector(n) = ";_
                   n,L,mutation_vector(n-1)
      end if

      n                  = (n+1) mod (D)
      L                 += 1

   loop while ((rnd < CR) and (L < D))


   '  C.3  Try the mutation to see how well it works.


      trial_cost = objective_function(mutation_vector(), _
                   number_data_pairs, column())
      if (debug=1) then print "   Trial_cost = ", trial_cost


   if (trial_cost <= cost(i)) then

      cost(i) = trial_cost

      for j=0 to (D-1)
         new_ensemble(i,j) = mutation_vector(j)
      next j

      if (trial_cost < minimum_cost)  then
         minimum_cost = trial_cost
         index_min_cost = i

         for j=0 to (D-1)
            best_vector(j) = mutation_vector(j)
         next j

      end if

   else

         for j=0 to (D-1)
            new_ensemble(i,j) = old_ensemble(i,j)
         next j

   end if


next i   '  End the mutation loop.


   '  C.4  Swap ensembles replacing the new generation with
   '       the old one.


      for k=0 to (NP-1)
         for j=0 to (D-1)
            value              = old_ensemble(k,j)
```

```
               old_ensemble(k,j)  = new_ensemble(k,j)
               new_ensemble(k,j)  = value
            next j
         next k



      '  C.5  Compute the mean and the variance of the
      '       objective "cost" function.


         sum = 0
      for j=0 to (NP-1)
         sum += cost(j)
         if (debug=1) then print "      cost(";j;") = ",cost(j)
      next j
         mean_cost = sum/NP

         sum = 0
      for j=0 to (NP-1)
         sum += (cost(j) - mean_cost)^2
      next j
         cost_standard_error = sqr(sum/(NP-1))

      if (debug = 2) then
         print "     generation, mean_cost, %st.error = ", _
         generation, mean_cost, cost_standard_error*100/mean_cost
      end if

      if (mean_cost < tolerance) then exit for


   next generation

       stop_time = timer


'----------------------------------------------------------------------
'
'  D.  Print out the results.
'
'----------------------------------------------------------------------

   print "                                                         "
   print "---------------------------------------------------------"
   print "|                                                       |"
   print "|        Differential Evolution Results                 |"
   print "|                                                       |"
   print "---------------------------------------------------------"
   print "                                                         "
   print "  A.  Print out the coefficients of the curvefit.        "
   print "      A.1  Coefficients:  "
   for i=0 to (D-1)
      a(i) = best_vector(i)
      print "                a(";i;") = "; a(i)
   next i
   print"                                                          "
```

```
print"   B.  Comparison with original data.                     "
print"                                                          "
print" i   x   y   z   response  computed response  deviation(%)"
print" -   -   -   -   --------  -----------------  ------------"

   sum = 0.0

for i = 0 to (number_data_pairs - 1)

   x         = column(1, i)
   y         = column(2, i)
   z         = column(3, i)
   u         = column(4, i)
   response  = column(5, i)

   deviation = a(0) + a(1)*x + a(2)*y + a(3)*z + a(4)*u + a(5)*x*x _
               + a(6)*y*y + _
               a(7)*z*z + a(8)*u*u + a(9)*x*x*x + a(10)*y*y*y + _
               a(11)*z*z*z + a(12)*u*u*u - response
   sum      += deviation * deviation

   print i, x, y, z, u, response, deviation * 100 / response

next i

   print "Standard error is:  "; sqr(sum/(number_data_pairs - 1))
   print " "
   print " "


   sum = 0.0
   print "z/l, response, computed response"
for i = 0 to (number_data_pairs - 1)

   x         = column(1, i)
   y         = column(2, i)
   z         = column(3, i)
   u         = column(4, i)
   response  = column(5, i)

   deviation = a(0) + a(1)*x + a(2)*y + a(3)*z + a(4)*u + a(5)*x*x _
               + a(6)*y*y + _
               a(7)*z*z + a(8)*u*u + a(9)*x*x*x + a(10)*y*y*y _
               + a(11)*z*z*z + a(12)*u*u*u - response
   sum      += deviation * deviation

   print column(0, i), ",", response, ",", (response + deviation)

next i

   print "Standard error is:  "; sqr(sum/(number_data_pairs - 1))

   print "                                                          "
   print "  C.  Specifics of the Run                                "
   print "      C.1  Number of generations:      "; generation - 1
   print "      C.2  Final 'cost':               "; _
         cost(index_min_cost)
```

```
      print "      C.3  Tolerance (based on cost):  "; tolerance
      print "      C.4  Mean cost:                  "; mean_cost
      print "      C.5  Standard Error in Cost:     "; _
            cost_standard_error
      print "      C.6  Runtime (s):                "; _
            stop_time - start_time
      print "      C.7  Date:                       "; date
      print "      C.8  Time:                       "; time
      print "      C.9  Number of data pairs:       "; _
            number_data_pairs

end


function objective_function(a()               as double,  _
                            number_data_pairs as integer, _
                            column()  as double)  as double

'-----------------------------------------------------------------------
'
'  This is a fit of the entered data sets.
'  The equation is of the form:  f(x) = A * col1^B * col2^C *
'  col3^D * ...
'
'-----------------------------------------------------------------------

   dim as integer i, j, k, aa
   dim as double  x, y, z, u, f, deviation, sum, standard_error


      sum = 0.0

   for i = 0 to (number_data_pairs - 1)

      x         = column(1, i)
      y         = column(2, i)
      z         = column(3, i)
      u         = column(4, i)
      f         = column(5, i)

      deviation = a(0) + a(1)*x + a(2)*y + a(3)*z + a(4)*u + a(5)*x*x _
                  + a(6)*y*y + _
                  a(7)*z*z + a(8)*u*u + a(9)*x*x*x + a(10)*y*y*y _
                  + a(11)*z*z*z + a(12)*u*u*u - f

      sum      += deviation * deviation

   next i

      standard_error = sqr(sum)

      return standard_error

end function
```

A.2 Input File: input.cor

This file provided input to the DE-DATAFIT.bas program and contained information specific to the differential evolution algorithm.

```
'-----------------------------------------------------------------------
'
'   Differential Evolution Algorithm Input File
'
'-----------------------------------------------------------------------

   A.  Define Information on the Data to be Analyzed.

       A.1  Tolerance in the cost function (std. error):             1e-20

   B.  Define Parameters for the Differential Evolution Algorithm.

       B.1  Number of populations in the ensemble (NP):              10
       B.2  F (Weighting Ratio) [0,2]:                               0.7
       B.3  CR (cross-over ratio) [0,1]:                             0.5
       B.4  Maximum number of generations for the evolution:         1e6

   C.  Printouts

       C.1  Debug?  (0 = off, 1 = on, 2 = summary, only):            2
```

A.3 Input File: input-SNRE.txt

This file contains the input for the computer program UNLV-SNRE.c used for the corrosion analysis. This particular input file was used with the program in section A.4 for post-burn analysis of a variable area nozzle.

```
*************************************************************************
*                                                                       *
*                             Program SNRE                              *
*                           User Input File                             *
*                                                                       *
*                                                                       *
*************************************************************************


A.  Reactor Conditions During Operation

    A.1  Mass flowrate of H2 (kg/s):                              11.5
    A.2  Thermal power production (W):                            503.9e6
    A.3  Duration of burn (s):                                    500
    A.4  Postburn time (s):                                       36000
    A.5  Pressure supplied by turbopump (Pa):                     6.91e6

B.  Reactor Core Geometry

    B.1  Reactor diameter (m):                                    0.5334
    B.2  Reactor length (m):                                      1.32
    B.3  Number of fuel elements in the core:                        402
    B.4  Number of tie tubes in the core:                            120

C.  Define the Fuel Element Geometry

    C.1  Coolant reactor inlet temperature (K):                      127.78
    C.2  Reactor inlet coolant pressure (Pa):                        5.557e6
    C.3  Fuel element length (m):                                    1.32
    C.4  Number of coolant channels:                                 19
    C.5  Hexagonal fuel element width (flat to flat) (m):            0.01905
    C.6  Weight fraction of U-235 in fuel:                           0.06
    C.7  Weight fraction of U-238 in fuel:                           0.0046
    C.8  Weight fraction of Zr in fuel:                              0.5533
    C.9  Weight fraction of C in fuel:                               0.382
    C.A  Coolant channel diameter (m):                               0.28e-2
    C.B  Coolant channel pitch (m)                                   0.43e-2
    C.C  Coolant channel coating thickness (m):                      50.0e-6
    C.D  Fuel extrapolation length, delta (m):                       5.0e-2
    C.E  Hydrogen storage tank temperature (K):                      20.0

D.  Define the Tie Tube Geometry

    D.1  Average hydrogen coolant temperature (K):                   85
    D.2  Average hydrogen coolant pressure (Pa):                     6.8948e6
    D.3  Radius, inner hydrogen channel (m):                         0.235e-2
    D.4  Outer radius, inner inconel tube (m):                       0.2605e-2
    D.5  Outer radius, outside inner inconel tube gap (m):           0.2665e-2
    D.6  Outer radius of moderator (ZrH2) layer (m):                 0.584e-2
    D.7  Outer radius of outer hydrogen channel (m):                 0.68825e-2
    D.8  Outer radius of second inconel tube (m):                    0.6985e-2
    D.9  Outer radius of the second inconel tube gap (m):            0.705e-2
    D.A  Outer radius of insulator (ZrC) layer (m):                  0.8065e-2
    D.B  Outer radius of outside insulator layer gap (m):            0.813e-2
    D.C  Hexagonal tie tube width (flat to flat) (m):                1.913e-2

E.  Laval Nozzle

    E.1  Nozzle exit diameter (m):                                   1.87

F.  Material densities

    F.1  ZrC (kg/m^3):                                               6.73e3
    F.2  ZrH2 (kg/m^3):                                              5.62e3
    F.3  C (kg/m^3):                                                 2.25e3
    F.4  Uranium, zirconium, carbon fuel mix (kg/m^3):               3.51e3
    F.5  Inconel 718 (kg/m^3):                                       8.44e3

G.  Simulation Parameters

    G.1  Number of increments in the z (axial) direction:            300
    G.2  Temperature increment (K):                                  0.1
    G.3  Increment in dm/dt (kg/s):                                  0.001
    G.4  Time increment for post-burn (s):                           10.0
    G.5  Maximum time for the post-burn simulation (s):              3600.0
    G.6  Maximum allowable fuel temperature (K):                     2700.0

H.  Iteration to Match the Total Impulse

    H.1  Current fixed nozzle burn time (s), x(upper):               3300.0
    H.2  Extimate for the lower nozzle burn time (s), x(lower):      3000.0
    H.3  Tolerance in the final value of burn time (%):              0.1
```

A.4 Program: UNLV-SNRE.c

This is the version of the UNLV computer program that models the flow of hydrogen propellant through the NTP reactor, the changes in fuel, coating, and bulk temperatures through the core, pressure losses, flow through the DeLaval nozzle, and calculation of thrust and specific impulse. This program was used in different forms for each of the four research tasks. This particular version was used for variable area nozzles with post-burn integrated impulse added to the impulse from the main burn of the rocket engine.

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
#include <string.h>

#define NN 2000

#include "H2_thermodynamic_properties.h"


   double PI = 3.14159265358;


/*  Returns the sign of the value T.  */

int sgn(double x) {
    if (x >  0)  return( 1);
    if (x <  0)  return(-1);
                 return( 0);
}



double decay_heat_ratio(double t_i, double t_d)  {

   /***********************************************************/
   /*                                                         */
   /*   Function:  decay_heat_ratio()                         */
   /*                                                         */
   /*   Purpose:   Compute the ratio of decay heat to reactor */
   /*              power (H/P) for a reactor or a fuel        */
   /*              assembly based on the irradiation time and */
   /*              the decay time after the irradiation has   */
   /*              stopped.                                   */
   /*   Input:     a)  t_i, irradiation time (s).             */
   /*              b)  t_d, decay time (s) after irradiation. */
   /*   Output:    Decay heat generation rate / reactor power */
   /*   Source:    R. B. Pond and J. E. Matos, ANL/RERTR/TM26 */
   /*   Author:    WGC and KG                                 */
   /*   Version:   1, 11/9/2018.                              */
   /*                                                         */
   /***********************************************************/

   double value, R_1, R_2, R_3, R_avg;
   double T_i, T_d;

      T_i = t_i / (24.0 * 3600.);    // Convert from seconds to days.
      T_d = t_d / (24.0 * 3600 );    // Convert from seconds to days.

      R_1 = 0.00685 * (pow(T_d,-0.2) - pow((T_i + T_d), -0.2));
      R_2 = 0.00495 * pow(T_d, -0.06) * (pow(T_d, -0.2) - pow((T_i + T_d), -0.2));
      R_3 = 0.1 * (pow((t_d + 10), -0.2) - pow((t_i + t_d + 10), -0.2))
          - 0.087 * ( pow((t_d + 2e7), -0.2) - pow((t_i + t_d + 2e7), -0.2));

      value = (R_1 + R_2 + R_3) / 3.0;

      return(value);

}



double Area_Ratio(double M, double k)  {

/*****************************************************************************/
/*                                                                           */
/*  Function:  Area_Ratio                                                    */
/*                                                                           */
/*  Purpose:   Compute the area ratio corresponding to selected values       */
/*             of the Mach number and for k (ratio of specific heats).       */
/*                                                                           */
/*  Input:     a)  M          = desired Mach number.                         */
/*             b)  k          = ratio of specific heats.                     */
/*                                                                           */
/*  Output:    a)  A / A(throat).                                            */
/*                                                                           */
/*  Version:   1, 12/17/2018.                                                */
/*                                                                           */
/*****************************************************************************/

   double value, power, argument;

   argument = 2/(k+1) * (1 + (k-1) * M * M/2);
   power    = (k+1) / (2 * (k-1));

   value = (1/M) * pow(argument, power);
   return(value);

}



double Mach_Area_Ratio_to_Throat(double A_ratio, double k, int supersonic)  {

/*****************************************************************************/
/*                                                                           */
/*  Function:  Mach_Area_Ratio                                               */
/*                                                                           */
/*  Purpose:   Compute the Mach number corresponding to selected values      */
/*             of the area ratio, A / A(throat), of k (ratio of specific     */
/*             heats) and for a subsonic or supersonic solution.            */
/*                                                                           */
/*  Input:     a)  A_ratio    = desired value of A / A(throat).              */
/*             b)  k          = ratio of specific heats.                     */
/*             c)  supersonic = 1, find supersonic flow solution.            */
/*                 supersonic = 0, find subsonic flow solution.              */
/*                                                                           */
/*  Output:    a)  Mach number.                                              */
/*                                                                           */
/*  Version:   1, 12/17/2018.                                                */
/*                                                                           */
/*****************************************************************************/

   double M, M_old, power, argument, f, f_prime;
   double Error, Tolerance, minimum;

   int iterations;

      Tolerance = 1e-6;
      minimum   = 1e-6;

   /*  A. Use these seed values to start.  */

      if (supersonic == 1)    M = 5.0;
      if (supersonic == 0)    M = 0.5;

         M_old      = M;
         iterations = 0;

   /*  B. Use a Newton-Raphson iteration to find M.  */

   do  {

      power    = (k + 1) / (2 * (k - 1));
      argument = ( (2 + (k - 1) * M * M) ) / (k + 1);

      f        = pow(argument, power) / M - A_ratio;

      power    = 1 / (k - 1) - 1 / 2;
      f_prime  = 2 * (M * M - 1) * pow(argument, power) / (M * M * (k + 1));

      M        = M_old - f / f_prime;
      if (M < minimum)  M = minimum;

      Error    = fabs((M - M_old) / M);

      M_old    = M;
      ++ iterations;

   }  while ((Error > Tolerance) && (iterations < 100));


      return(M);

}



double Mach_Area_Ratio(double A_ratio, double M_1, double k, int supersonic)  {

/*****************************************************************************/
/*                                                                           */
/*  Function:  Mach_Area_Ratio                                               */
/*                                                                           */
/*  Purpose:   Compute the Mach number corresponding to selected values     */
/*             of the area ratio, A / A(throat), of k (ratio of specific     */
/*             heats) and for a subsonic or supersonic solution.            */
/*                                                                           */
/*  Input:     a)  A_ratio    = desired value of A / A(throat).              */
/*             b)  k          = ratio of specific heats.                     */
/*             c)  M_1        = Mach number
/*             c)  supersonic = 1, find supersonic flow solution.            */
/*                 supersonic = 0, find subsonic flow solution.              */
/*                                                                           */
/*  Output:    a)  Mach number.                                              */
/*                                                                           */
/*  Version:   1, 12/17/2018.                                                */
/*                                                                           */
/*****************************************************************************/

   double M, M_old, power, argument, f, f_prime;
   double Error, Tolerance, minimum, c_1;

   int iterations;

      Tolerance = 1e-6;
      minimum   = 1e-6;

   /*  A. Use these seed values to start.  */

      if (supersonic == 1)    M = 5.0;
      if (supersonic == 0)    M = 0.5;

         M_old      = M;
         iterations = 0;

   /*  B. Use a Newton-Raphson iteration to find M.  */

      power    = (k + 1) / (2 * (k - 1));
      c_1      = M_1 / pow((2 + (k - 1) * M_1 * M_1), power);

   do  {

      f        = A_ratio - c_1 * pow((2 + (k - 1) * M * M), power) / M;

      f_prime  = c_1 * ( -2 * (k - 1) *
                 pow((2 + (k - 1) * M * M), -(k - 3) / (2 * (k - 1))) +
                 pow((2 + (k - 1) * M * M), power) / (M * M)  );

      M        = M_old - f / f_prime;

      if ((M < minimum) && (supersonic == 0))  M = minimum;
      if ((M > 1)       && (supersonic == 0))  M = 1;

      if ((M < 1)       && (supersonic == 1))  M = 1;
      if ((M > 500)     && (supersonic == 1))  M = 500;

      Error    = fabs((M - M_old) / M);

      M_old    = M;
      ++ iterations;

   }  while ((Error > Tolerance) && (iterations < 100));

      return(M);

}



/***************************************************************/
/*                                                             */
/*  Program:  UNLV-SNRE-t.c                                    */
/*                                                             */
/*  Purpose:  Model a nuclear thermal fission rocket engine.   */
/*            Compute temperature distributions in the bulk    */
/*            fluid, in the coating, and in the fuel as a      */
/*            function of "z" along the central axis of the    */
/*            reactor.                                         */
/*            Note:  This version has been modified to study   */
/*                   the performance of a variable area nozzle */
/*                   and computes the post-burn flow throught  */
/*                   the de Laval nozzle.                      */
/*  Input:    1)  Hydrogen properties.                         */
/*            2)  Input file:  input-snre-t.txt                */
/*  Output:   1)  thermodynamic-properties.dat                 */
/*            2)  output.dat                                   */
/*            3)  postburn.dat                                 */
/*  Version:  5, 6/2019, WGC KG                                */
/*            Updated 6/1/22, WGC and KG                       */
/*            Copyright (2022) Kimberly Gonzalez and           */
/*            William Culbreth.                                */
/*                                                                */
/******************************************************************/


int main()  {

      FILE   *fp1, *fp2;

      time_t t_start, t_end;

      char   input_line[256], title_string[32], value_string[32], tag_string[32];
      char   Fuel_Assembly_name[32], Output_file[32];
      char   * c_time_string;

      int    i, j;

      double T_LH2_tank, p_LH2_tank;
      double dm_dt, Power, a;
      double T_reactor_inlet, R_channel, D_channel, pitch;
      double D_reactor, R_reactor, L_reactor, extrapolation_length;
      double D_fuel_element, coating_thickness;

      double A_channel, A_total_cell, A_fuel_equivalent;
      double R_fuel, FE_length;
      double p_reactor_inlet;
      int    N_coolant_channels, N_fuel_elements;
      double A_fuel_element, A_fuel;
      double R_coating;
      int    N_z_increments;
      double T_increment, z, dz;
      double T, T_b, dq_total, H_e, dq_dz_center;
      int    oldsign, newsign;

      double T_bulk[NN], axial_position[NN];
      double dmdt_channel, V_channel[NN], M_channel[NN];

      double dq_dA, Nu_f, Pr_f, Re_f, h, M_f, dq_coolant_channel, F_R, T_fs, gamma;
      double T_wall[NN], T_fa[NN];
      double Volume_fuel, dq_dV[NN], T_coating[NN], T_fuel_max[NN];
      double pressure[NN];
      double dq_reactor, R_chamber, A_chamber, M_chamber;
      double T_o_chamber, p_o_chamber;
      double A_coolant_channel;
      double T_reactor_exit, p_reactor_exit, A_reactor_exit, M_reactor_exit;
      double k, dM;
      double c, T_chamber, p_chamber, T_chamber_stagnation, p_chamber_stagnation;
      double A_exit, A_ratio, A_ratio_exit_to_throat;

      double T_throat, p_throat, A_throat, M_throat;
      double Thrust, Thrust_momentum_term, Thrust_pressure_term;
      double T_exit, p_exit, V_exit, M_exit, a_exit;

      double burn_time, postburn_time, I_sp;
      double T_bulk_max, z_bulk_max, T_coating_max, z_coating_max;
      double T_fuel_center_max, z_fuel_center_max;
      double D_throat, D_exit, D_chamber;
      double Q_preheat, Q_latent_heat, Q_vapor_to_reactor, Power_effective;

      double dmdt_FP, A_throat_FP, Power_FP, Isp_FP, Thrust_FP;
      double T_bulk_max_FP, T_fuel_max_FP, T_coating_max_FP;
      double Impulse, t_increment, dmdt_increment, t;
      double p_turbopump, Power_turbopump;
      double Thrust_subsonic, I_sp_subsonic, M_exit_subsonic, T_exit_subsonic;
      double p_exit_subsonic, a_exit_subsonic, V_exit_subsonic;
      double Thrust_momentum_term_subsonic, Thrust_pressure_term_subsonic;
      double T_stagnation, p_stagnation;
      double t_max;
      double Impulse_SUM, dmdt_current, A_throat_current, Power_current;
      double Isp_current, Thrust_current;
      double T_fuel_max_allowable, Impulse_FP, T_bulk_max_current;
      double T_coating_max_current, T_fuel_max_current,Impulse_current;
```

```
   double t_postburn, I_sp_current;

   double M_H2_used, Integral_Impulse;
   double process_time;;

   clock_t start_time, end_time;



      start_time = clock();


/***********************************************************/
/*                                                         */
/*   A.  Obtain user input.                                */
/*                                                         */
/***********************************************************/

      fp1 = fopen("input-SNRE-t.txt", "r");

   while (fgets(input_line, sizeof(input_line), fp1))  {

       for(j = 0; j < 3; ++j)  {
          title_string[j] = input_line[j + 4];
       }
          title_string[3] = '\0';
          strcpy(tag_string, title_string);

       for(j = 0; j < strlen(input_line); ++j)  {
          value_string[j] = input_line[j + 69];
       }

       if( strcmp(tag_string, "A.1") == 0 )  dm_dt                     =
           atof(value_string);
       if( strcmp(tag_string, "A.2") == 0 )  Power                     =
           atof(value_string);
       if( strcmp(tag_string, "A.3") == 0 )  burn_time                 =
           atof(value_string);
       if( strcmp(tag_string, "A.4") == 0 )  postburn_time             =
           atof(value_string);
       if( strcmp(tag_string, "A.5") == 0 )  p_turbopump               =
           atof(value_string);

       if( strcmp(tag_string, "B.1") == 0 )  D_reactor                 =
           atof(value_string);
       if( strcmp(tag_string, "B.2") == 0 )  L_reactor                 =
           atof(value_string);
       if( strcmp(tag_string, "B.3") == 0 )  N_fuel_elements           =
           atoi(value_string);

       if( strcmp(tag_string, "C.1") == 0 )  T_reactor_inlet           =
           atof(value_string);
       if( strcmp(tag_string, "C.2") == 0 )  p_reactor_inlet           =
           atof(value_string);
       if( strcmp(tag_string, "C.3") == 0 )  FE_length                 =
           atof(value_string);

       if( strcmp(tag_string, "C.4") == 0 )  N_coolant_channels        =
           atoi(value_string);
       if( strcmp(tag_string, "C.5") == 0 )  D_fuel_element            =
           atof(value_string);
       if( strcmp(tag_string, "C.A") == 0 )  D_channel                 =
           atof(value_string);
       if( strcmp(tag_string, "C.B") == 0 )  pitch                     =
           atof(value_string);
       if( strcmp(tag_string, "C.C") == 0 )  coating_thickness         =
           atof(value_string);
       if( strcmp(tag_string, "C.D") == 0 )  extrapolation_length      =
           atof(value_string);
       if( strcmp(tag_string, "C.E") == 0 )  T_LH2_tank                =
           atof(value_string);
       if( strcmp(tag_string, "E.1") == 0 )  D_exit                     =
           atof(value_string);

       if( strcmp(tag_string, "G.1") == 0 )  N_z_increments             =
           atof(value_string);
       if( strcmp(tag_string, "G.2") == 0 )  T_increment                =
           atof(value_string);
       if( strcmp(tag_string, "G.3") == 0 )  dmdt_increment             =
           atof(value_string);
       if( strcmp(tag_string, "G.4") == 0 )  t_increment                =
           atof(value_string);
       if( strcmp(tag_string, "G.5") == 0 )  t_max                      =
           atof(value_string);
       if( strcmp(tag_string, "G.6") == 0 )  T_fuel_max_allowable       =
           atof(value_string);

   }

      fclose(fp1);


/************************************************************/
/*                                                          */
/*   B.  Define constants and functions.                    */
/*                                                          */
/************************************************************/

   /*********************************************************/
   /*                                                       */
   /*  B.1  Define the radius of a coolant channel.         */
   /*                                                       */
   /*********************************************************/

      R_channel = D_channel / 2.0;


   /*********************************************************/
   /*                                                       */
   /*  B.2  Define the thermodynamic properties of the      */
   /*       coating materials and the fuel.                 */
   /*                                                       */
   /*********************************************************/

      /*  k_fuel(T) in (W/m-K), (K).  */

      double k_fuel(double T)  {
         double value;
         value =   2.023631384512141e-5 * T * T
                 - 0.087137556636804 * T
                 + 132.357237443363;
         // value = 30;
         return(value);
      }


      /*  k_coating(T) in (W/m-K), (K).  */

      double k_coating(double T)  {
         double value;
         value =   (1.0 / 120.0) * T
                  + 20.0;
         // value = 27;
         return(value);
      }


   /*********************************************************/
   /*                                                       */
   /*  B.3  Friction Factor in a Circular Channel.          */
   /*                                                      */
   /********************************************************/

         /*  Petukhov friction factor.  */

      double f_friction(double Re)  {
         double value;
         if(Re <  2000)  return(64/Re);
         if(Re >= 2000)  {
           /*   value = pow((0.79 * log(Re) - 1.64), 2);  */
           value = 0.316 / pow(Re, 0.25);
           return(value);
         }
      }


   /*****************************************************************/
   /*                                                               */
   /*  B.4  Nusselt Number - Use the Gnielinski correlation version */
   /*       of the Dittus=Boelter equation for turbulent .          */
   /*                                                               */
   /*           0.5 <= Pr <= 2000, 3000 <= Re <= 5e6                */
   /*                                                               */
   /*       This function takes into account the change in Nu with  */
   /*       Re and with z/D.                                        */
   /*                                                               */
   /*                                                               */
   /*****************************************************************/

      double Nu_Gnielinski(double Re, double Pr, double z,
                   double T_bulk, double T_wall)  {

         double value, D, f;

         if (Re <= 2300)  {
            return(4.36);
         }

         if (Re >  2300)  {
            f = f_friction(Re);
            D = 2 * R_channel;

            if(z < 0.01)  {
                  /*  This avoids a divide by zero error in (D/z).  */
                  return(0);
               }
               else  {
                  value = (((f/8) * (Re - 1000) * Pr)
                          / ((1 + 12.7 * sqrt(f/8)) * (pow(Pr, 0.6667) - 1))
                          * (1 + pow((D/z), 0.6667)))
                          * pow((T_bulk / T_wall), 0.45);
                  return(value);
            }
         }
      }


   /********************************************************/
   /*                                                      */
   /*  B.5  Dittus-Boelter Relationship.                   */
   /*                                                      */
   /********************************************************/

      double Nu_Dittus_Boelter(double Re, double Pr)  {
         double value;

         value = 0.023 * pow(Re, 0.8) * pow(Pr, 0.4);

         return(value);
      }
   /********************************************************/
   /*                                                      */
   /*  B.6  Reynolds number calculated from dm/dt.         */
   /*                                                      */
   /********************************************************/

      double Reynolds_number(double T, double dm_dt,
                             double R_channel)        {
         double value;
         value = (4/PI) * (dm_dt / (2 * R_channel * dynamic_viscosity_H2(T)));
         return(value);
      }


/*----------------------------------------------------------------------------*/
/*                                                                            */
/*   C. Begin the Calculations.                                               */
/*                                                                            */
/*----------------------------------------------------------------------------*/

   /*-------------------------------------------------------------------------*/
   /*                                                                         */
   /*   C.1  Compute the Andreas-equivalent geometry for the fuel.            */
   /*                                                                         */
   /*-------------------------------------------------------------------------*/

         A_channel         = PI * R_channel * R_channel;      /*  (m^2).  */
         A_total_cell      = (sqrt(3)/2) * pitch * pitch;     /*  (m^2).  */
         A_fuel_equivalent = A_total_cell - A_channel;
         R_fuel            = sqrt(A_fuel_equivalent / PI + R_channel * R_channel);

         A_fuel_element    = 6 * pow((D_fuel_element / 2), 2) * tan(PI/6);
         A_fuel            = A_fuel_element - N_coolant_channels * PI * R_channel *
                             R_channel;
         H_e               = L_reactor + 2 * extrapolation_length;
                             /*  Extrapolated height of the reactor (m).  /*


         /*---------------------------------------------------------------------*/
         /*                                                                     */
         /*  C.1.a  Reduce the channel thickness by the thickness               */
         /*         of the coating.                                             */
         /*                                                                     */
         /*---------------------------------------------------------------------*/

         R_channel        -= coating_thickness;
         D_channel         = 2 * R_channel;

         R_coating         = R_channel + coating_thickness;


         /*---------------------------------------------------------------------*/
         /*                                                                     */
         /*  C.1.b  Print out key thermodynamic data.                           */
         /*                                                                     */
         /*---------------------------------------------------------------------*/

      /*

         fp2 = fopen("thermodynamic-properties.dat", "w");

         for(T = 0; T <= 3000; T += 10)  {
            fprintf(fp2, "%lf, ", T);
            fprintf(fp2, "%lf, ", ratio_of_specific_heats_H2(T));
            fprintf(fp2, "%lf, ", specific_heat_p_H2(T) / R_H2);
            fprintf(fp2, "%lf, ", specific_heat_v_H2(T) / R_H2);
            fprintf(fp2, "%lf, ", speed_of_sound_H2(T));
            fprintf(fp2, "%lf, ", Prandtl_number_H2(T));
            fprintf(fp2, "%.8e, ", dynamic_viscosity_H2(T));
            fprintf(fp2, "%lf, ", integral_CpdT(T));
            fprintf(fp2, "%lf, ", thermal_conductivity_H2(T));
            fprintf(fp2, "\n");

         }

         fclose(fp2);

      */


/*--------------------------------------------------------------------------*/
/*                                                                          */
/*   D.  Compute the Rocket Performance Under Full Reactor Power.           */
/*                                                                          */
/*--------------------------------------------------------------------------*/

   t                = 0.0;
   M_H2_used        = 0.0;
   Integral_Impulse = 0.0;


            printf("*************************************************\n");
            printf("*                                               *\n");
            printf("*  I.  Full Reactor Power Burn                  *\n");
            printf("*                                               *\n");
            printf("*************************************************\n");
            printf("                                                 \n");

   /*---------------------------------------------------------------------*/
   /*                                                                     */
   /*  D.1  How much heat is required to bring the hydrogen temperature  */
   /*       up from "T_LH2_tank" to "T_reactor_inlet"?  Adjust the        */
   /*       heat that goes into the hydrogen flowing through the          */
   /*       reactor, appropriately. Also, include the power required      */
   /*       by the turbopump since it constitutes a small power drain.    */
   /*                                                                     */
   /*---------------------------------------------------------------------*/

            Q_vapor_to_reactor = dm_dt * (integral_CpdT(T_reactor_inlet) -
                                 integral_CpdT(T_LH2_tank));

            Q_latent_heat      = dm_dt * latent_heat_H2();

            Q_preheat          = Q_latent_heat + Q_vapor_to_reactor;
            Power_turbopump    = dm_dt * (p_turbopump - 0.0) / density_H2(T_LH2_tank);

            printf("A.  Estimate of Small Power Losses\n");
            printf("    1.  Power_turbopump (MW)      = %f\n", Power_turbopump / 1e6);
            printf("    2.  p_turbopump (MPa)     = %f\n", p_turbopump / 1e6);
            printf("    3.  dm_dt (kg/s)          = %f\n", dm_dt);
            printf("    4.  T_LH2_tank (K)        = %f\n", T_LH2_tank);
            printf("    5.  density_H2 (kg/m^3)   = %f\n", density_H2(T_LH2_tank));
            printf("    6.  latent_heat_H2 (J/kg) = %f\n", latent_heat_H2());
            printf("    7.  Q_latent_heat (MW)    = %f\n", Q_latent_heat/1e6);
            printf("                                  \n");

            Power_effective    = Power - Q_preheat - Power_turbopump;


   /*---------------------------------------------------------*/
   /*                                                         */
   /*   D.2  This iteration finds the bulk hydrogen coolant   */
   /*        temperature along the reactor length (z=0 to     */
   /*        L_reactor).  This is based on a simple energy    */
   /*        balance.  Ignoring axial conduction along the    */
   /*        fuel elements, the heat generated by a slice of  */
   /*        the equivalent fuel of width "dz" must go into   */
   /*        the hydrogen resulting in an increase of "dT"    */
   /*        in the bulk temperature.                         */
   /*                                                         */
   /*---------------------------------------------------------*/
         dz           = L_reactor / N_z_increments;
         pressure[0]  = p_reactor_inlet;
         dq_dz_center = H_e * sin(PI * L_reactor / (2 * H_e));

      for(z = -L_reactor/2 + (dz/2), T_b = T_reactor_inlet, i = 0;
          z <= L_reactor/2; z += dz, ++i)                   {

            axial_position[i] = z;


         /*----------------------------------------------------------------------*/
         /*                                                                      */
         /*  D.2.a  First, compute the heat generated, "dq_total", by a          */
         /*         thickness of the reactor integrated from entrance            */
         /*         at -(L_reactor/2) to z.                                      */
         /*                                                                      */
         /*----------------------------------------------------------------------*/

            dq_total =   (Power / 2) *
                       (  1 + ( sin(PI * z / H_e) /
                       sin((PI * L_reactor) / (2 * H_e)) )  );


         /*----------------------------------------------------------------------*/
         /*                                                                      */
         /*  D.2.b  Next, compute the change in the bulk coolant temperature     */
         /*         corresponding to "dq_total" within the interval "z" from     */
         /*         the reactor entrance.                                        */
         /*                                                                      */
         /*----------------------------------------------------------------------*/

            T       = T_b;
            oldsign = sgn(dm_dt * (integral_CpdT(T)
                                 - integral_CpdT(T_reactor_inlet))
                      - dq_total);

            for(T = T + T_increment; T < 6000; T += T_increment)  {
               newsign = sgn(dm_dt * (integral_CpdT(T)
                                    - integral_CpdT(T_reactor_inlet))
                         - dq_total);
               if (newsign != oldsign)  {
                  T_b = T;
                  break;
               }
            }

            T_bulk[i] = T_b;


         /*----------------------------------------------------------------------*/
         /*                                                                      */
         /*  D.2.c  OK, now compute the velocity through each coolant channel.   */
         /*                                                                      */
         /*----------------------------------------------------------------------*/

            dmdt_channel = dm_dt / (N_coolant_channels * N_fuel_elements);
            V_channel[i] = dmdt_channel / (A_channel * density_H2(T_b));
            M_channel[i] = V_channel[i] / speed_of_sound_H2(T_b);


         /*---------------------------------------------------------------------*/
         /*                                                                     */
         /*  D.2.d  Calculate the wall temperature of the coolant channel.      */
         /*                                                                     */
         /*     Re_f  = Reynolds number of the bulk fluid at "z".               */
         /*     Pr_f  = Prandtl number of the bulk fluid at "z".                */
         /*     Nu_f  = Nusselt number calculated at "z".                       */
         /*     dq_reactor = total reactor power produced between z             */
         /*                  and z+dz.                                          */
         /*     dq_coolant_channel  = total power power produced in single      */
```

```
      /*                  coolant channel from z to z+dz.                    */
      /*     dq_dA = power produced in single coolant channel per            */
      /*             unit surface area facing the coolant.                   */
      /*     M_f   = Mach number in the bulk fluid at "z".                   */
      /*     gamma = ratio of specific heats at "z".                         */
      /*     T_fs  = temperature of the fluid at the channel surf.           */
      /*     T_fa  = temperature at wall adjusted for Mach number.           */
      /*                                                                     */
      /*---------------------------------------------------------------------*/

         dq_reactor         = (Power/2) * (sin(PI * (z + dz)/H_e) - sin(PI * z / H_e))
                               / sin(PI * L_reactor / (2 * H_e));
         dq_coolant_channel = dq_reactor / (N_coolant_channels * N_fuel_elements);
         dq_dA              = dq_coolant_channel / (2 * PI * R_channel * dz);

         Re_f       = Reynolds_number(T_b, dmdt_channel, R_channel);
         Pr_f       = Prandtl_number_H2(T_b);
         M_f        = M_channel[i];
         F_R        = pow(Prandtl_number_H2(T_b), 0.3333);
         gamma      = ratio_of_specific_heats_H2(T_b);
         T_fs       = T_b * (1 + (gamma - 1) * M_f * M_f / 2);
         T_fa[i]    = F_R * (T_fs - T_b) + T_b;


      /*---------------------------------------------------------------------*/
      /*                                                                     */
      /*  D.2.e  Pressure drop to this value of z.                           */
      /*                                                                     */
      /*---------------------------------------------------------------------*/

         pressure[i+1] = pressure[i] - f_friction(Re_f) * (dz / D_channel)
                           * V_channel[i] * V_channel[i] / 2.0;

         Nu_f          = Nu_Dittus_Boelter(Re_f, Pr_f);
         h             = Nu_f * thermal_conductivity_H2(T_b) / D_channel;
         T_wall[i]     = dq_dA / h + T_fa[i];


      /*---------------------------------------------------------------------*/
      /*                                                                     */
      /*  D.2.f  Next, find the temperature in the coating and in the fuel.  */
      /*                                                                     */
      /*---------------------------------------------------------------------*/

         Volume_fuel   = N_coolant_channels * N_fuel_elements * A_fuel
                         * L_reactor;
         dq_dV[i]      = - dq_reactor / Volume_fuel;
         T_coating[i]  = ((R_fuel * R_fuel - R_coating * R_coating) * dq_dV[i]
                         / (2 * k_coating(T_wall[i])))
                         * log(R_channel / R_coating) + T_wall[i];
         T_fuel_max[i] = (dq_dV[i] / (4 * k_fuel(T_coating[i])))
                         * (R_fuel * R_fuel - R_coating * R_coating
                         + 2 * R_fuel * R_fuel * log(R_coating / R_fuel))
                         + T_coating[i];

   }


/*---------------------------------------------------------------------------*/
/*                                                                           */
/*   E.  DeLaval Nozzle Calculations.                                        */
/*                                                                           */
/*       At this point, all temperatures and pressures have been computed  */
/*       in the reactor.  Compute the properties in the DeLaval nozzle,    */
/*       including the thrust.                                               */
/*                                                                           */
/*---------------------------------------------------------------------------*/

   /*---------------------------------------------------------------------*/
   /*                                                                     */
   /*   E.1  Compute the maximum fuel, bulk and coating temperatures.     */
   /*                                                                     */
   /*---------------------------------------------------------------------*/

         T_bulk_max        = T_bulk[0];
         T_coating_max     = T_coating[0];
         T_fuel_center_max = T_fuel_max[0];

      for(i = 0; i < N_z_increments; ++i)  {

         if (T_bulk_max < T_bulk[i])  {
            T_bulk_max = T_bulk[i];
            z_bulk_max = axial_position[i];
         }

         if (T_coating_max < T_coating[i])  {
            T_coating_max = T_coating[i];
            z_coating_max = axial_position[i];
         }

         if (T_fuel_center_max < T_fuel_max[i])  {
            T_fuel_center_max = T_fuel_max[i];
            z_fuel_center_max = axial_position[i];
         }

      }

         printf("\n");


   /*---------------------------------------------------------------------*/
   /*                                                                     */
   /*  E.2  Compute the stagnation values and specific impulse            */
   /*       in the lower reactor chamber.                                 */
   /*                                                                     */
   /*     1.  The H2 expands as it exits the coolant channels             */
   /*         and enters the chamber.                                     */
   /*     2.  Compute p, T, and M in the chamber.                         */
   /*     3.  Compute stagnation properties in the chamber.               */
   /*     4.  Compute the properties at the throat.                       */
   /*     5.  Compute exit Mach number and velocity.                      */
   /*     6.  Compute specific impulse and thrust.                        */
   /*                                                                     */
   /*                                                                     */
   /*---------------------------------------------------------------------*/

         T_reactor_exit    = T_bulk[N_z_increments - 1];
         p_reactor_exit    = pressure[N_z_increments - 1];

         k                 = ratio_of_specific_heats_H2(T_reactor_exit);

         A_coolant_channel = PI * R_channel * R_channel;
         A_reactor_exit    = A_coolant_channel * N_coolant_channels
                              * N_fuel_elements;
         M_reactor_exit    = M_channel[N_z_increments - 1];
         T_stagnation      = T_reactor_exit * (1 + ((k - 1) / 2) *
                             M_reactor_exit * M_reactor_exit);
         p_stagnation      = p_reactor_exit * pow(  (1 + ((k - 1) / 2) *
                             M_reactor_exit * M_reactor_exit), k / (k - 1)  );

            /*  Properties in the lower reactor chamber.  */

         D_chamber         = D_reactor;
         A_chamber         = (PI / 4) * D_chamber * D_chamber;

         printf("B.  Reactor exit to the lower reactor chamber.  \n");
         printf("    1.  T_reactor_exit (K)       = %f\n", T_reactor_exit);
         printf("    2.  p_reactor_exit (MPa)     = %f\n", p_reactor_exit / 1e6);
         printf("    3.  A_coolant_channel (mm^2) = %f\n", A_coolant_channel * 1e6);
         printf("    4.  A_reactor_exit (m^2)     = %f\n", A_reactor_exit);
         printf("    5.  A_chamber (m^2)          = %f\n", A_chamber);
         printf("    6.  D_chamber (mm)           = %f\n", D_chamber * 1e3);
         printf("    7.  M_reactor_exit           = %f\n", M_reactor_exit);
         printf("    8.  T_stagnation (K)         = %f\n", T_stagnation);
         printf("    9.  p_stagnation (MPa)       = %f\n", p_stagnation/1e6);
         printf("   10.  k                        = %f\n", k);
         printf("                                     \n");


      /*---------------------------------------------------------------------*/
      /*                                                                     */
      /*   E.3  Compute the Mach number in the lower reactor                 */
      /*        chamber by iteration.                                        */
      /*                                                                     */
      /*---------------------------------------------------------------------*/


         A_ratio   = A_chamber / A_reactor_exit;
         k         = ratio_of_specific_heats_H2(T_reactor_exit);
         M_chamber = Mach_Area_Ratio(A_ratio, M_reactor_exit, k, 0);

         c         = (k - 1) / 2;
         T_chamber = T_reactor_exit * (1 + c * M_chamber * M_chamber)
                                    / (1 + c * M_reactor_exit * M_reactor_exit);
         p_chamber = p_reactor_exit * pow( (1 + c * M_chamber * M_chamber)
                                      / (1 + c * M_reactor_exit * M_reactor_exit),
                                      (k / (k - 1)) );
         T_chamber_stagnation = T_chamber * (1 + c * M_chamber * M_chamber);
         p_chamber_stagnation = p_chamber * pow(1 + c * M_chamber * M_chamber,
                                             k / (k - 1) );

         printf("C.  Values in the DeLaval Nozzle Upper Chamber  \n");
         printf("    1.  Chamber Mach number                 = %f\n", M_chamber);
         printf("    2.  Area ratio (chamber / reactor exit) = %f\n", A_ratio);
         printf("    3.  k                                   = %f\n", k);
         printf("    4.  T_chamber (K)                       = %f\n", T_chamber);
         printf("    5.  p_chamber (MPa)                     = %f\n", p_chamber / 1e6);
         printf("    6.  M_chamber                           = %f\n", M_chamber);
         printf("    7.  T_chamber_stagnation (K)            = %f\n",
                         T_chamber_stagnation);
         printf("    8.  p_chamber_stagnation (MPa)          = %f\n",
                         p_chamber_stagnation / 1e6);
         printf("                                                \n");


      /*---------------------------------------------------------------------*/
      /*                                                                     */
      /*   E.4  Next, calculate what happens as we move from the reactor     */
      /*        chamber to the throat.                                       */
      /*                                                                     */
      /*---------------------------------------------------------------------*/

          k        = ratio_of_specific_heats_H2(T_chamber);
          A_throat = dm_dt / (p_chamber_stagnation *
                     sqrt(   (k / (R_H2 * T_chamber_stagnation)) *
                     pow(  2 / (k + 1), (k - 1) / (k + 1) )  )    );
          T_throat = T_chamber * (2 + (k - 1) * M_chamber * M_chamber) /
                                    (2 + (k - 1) )  ;
          p_throat = p_chamber * pow( (T_throat / T_chamber), k / (k - 1) );
          M_throat = 1;
          D_throat = sqrt( (4/PI) * A_throat );

          printf("D.  Throat Values.\n");
          printf("    1.  A_throat (mm^2) = %.8e\n", A_throat * 1e6);
          printf("    2.  p_throat (MPa)  = %.8e\n", p_throat / 1e6);
          printf("    3.  T_throa1 (K)    = %f\n",   T_throat);
          printf("    4.  M_throat        = %f\n",   M_throat);
          printf("    5.  k               = %f\n",   k);
          printf("    6.  D_throat (mm)   = %f\n",   D_throat * 1e3);
          printf("    7.  dm/dt (kg/s)    = %f\n",   dm_dt);
          printf("    8.  R (H2) (J/kg-K) = %f\n",   R_H2);
          printf("    \n");
```

```
/*---------------------------------------------------------------------*/
/*                                                                     */
/*   E.5  Calculate values for the DeLaval Nozzle exit.                */
/*                                                                     */
/*---------------------------------------------------------------------*/

      k                       = ratio_of_specific_heats_H2(T_reactor_exit);
      A_exit                  = (PI/4) * D_exit * D_exit;
                               // (m^2).
      A_ratio_exit_to_throat = A_exit / A_throat;

   /*----------------------------------------------------------------*/
   /*                                                                */
   /*   E.5.a  Compute values at the exit for the supersonic flow    */
   /*          solution.  This is valid if the throat area is the    */
   /*          correct diameter to choke the flow.                   */
   /*                                                                */
   /*----------------------------------------------------------------*/

      M_exit = Mach_Area_Ratio_to_Throat(A_ratio_exit_to_throat, k, 1);

      T_exit = T_throat * (k + 1) / ( 2 + (k - 1) * M_exit * M_exit);   // (K).
      p_exit = p_throat * pow( (T_exit / T_throat), (k / (k - 1)) );    // (Pa).
      a_exit = sqrt(k * R_H2 * T_exit);                                 // (m/s).
      V_exit = M_exit * a_exit;                                         // (m/s).

      Thrust_momentum_term = dm_dt * V_exit;                            // (N).
      Thrust_pressure_term = p_exit * A_exit;                           // (N).

      Thrust = dm_dt * V_exit + p_exit * A_exit;                        // (N).

      I_sp   = Thrust / (dm_dt * 9.81);                                 // (s).


      printf("E.  Exit Values.\n");
      printf("    1.  k                         = %f\n", k);
      printf("    2.  M_exit                    = %f\n", M_exit);
      printf("    3.  A_exit (m^2)              = %f\n", A_exit);
      printf("    4.  T_exit (K)                = %f\n", T_exit);
      printf("    5.  p_exit (MPa)              = %f\n", p_exit / 1e6);
      printf("    6.  a_exit (m/s)              = %f\n", a_exit);
      printf("    7.  V_exit (m/s)              = %f\n", V_exit);
      printf("    8.  Thrust, momentum term (N) = %f\n", Thrust_momentum_term);
      printf("    9.  Thrust, pressure term (N) = %f\n", Thrust_pressure_term);
      printf("   10.  Total Thrust (N)          = %f\n", Thrust);
      printf("        Total Thrust (klbf)       = %f\n", Thrust * 0.224809 /
                      1000);
      printf("   11.  Specific Impulse (s)      = %f\n", I_sp);
      printf("   12.  D_exit (mm)               = %f\n", D_exit * 1e3);
      printf("                                      \n");


   /*----------------------------------------------------------------*/
   /*                                                                */
   /*   E.5.b  Compute values at the exit for the subsonic flow      */
   /*          solution.  This is valid if the throat area is too    */
   /*          large to choke the flow.                              */
   /*                                                                */
   /*----------------------------------------------------------------*/

      M_exit_subsonic = Mach_Area_Ratio_to_Throat(A_ratio_exit_to_throat, k, 0);

      T_exit_subsonic = T_throat * (k + 1) / ( 2 + (k - 1) *
                        M_exit_subsonic * M_exit_subsonic);           // (K).
      p_exit_subsonic = p_throat * pow( (T_exit_subsonic / T_throat),
                        (k / (k - 1)) );                              // (Pa).
      a_exit_subsonic = sqrt(k * R_H2 * T_exit_subsonic);             // (m/s).
      V_exit_subsonic = M_exit_subsonic * a_exit_subsonic;            // (m/s).
```

```
               Thrust_momentum_term_subsonic = dm_dt * V_exit_subsonic;        // (N).
               Thrust_pressure_term_subsonic = p_exit_subsonic * A_exit;       // (N).

               Thrust_subsonic = dm_dt * V_exit_subsonic + p_exit_subsonic * A_exit;
                                                                               // (N).

               I_sp_subsonic   = Thrust_subsonic / (dm_dt * 9.81);             // (s).


               printf("F. Exit Values for Subsonic Exit.\n");
               printf("   1.  T (K)                      = %f\n", T_exit_subsonic);
               printf("   2.  p (MPa)                    = %f\n", p_exit_subsonic/1e6);
               printf("   3.  Thrust, subsonic (N)       = %f\n", Thrust_subsonic);
               printf("   4.  Mach number                = %f\n", M_exit_subsonic);
               printf("   5.  I_sp_subsonic (s)          = %f\n", I_sp_subsonic);
               printf("   6.  Speed of sound (m/s)       = %f\n", a_exit_subsonic);
               printf("   7.  Exhaust velocity (m/s)     = %f\n", V_exit_subsonic);
               printf("   8.  Thrust due to momentum (N) = %f\n",
                              Thrust_momentum_term_subsonic);
               printf("   9.  Thrust due to pressure (N) = %f\n",
                              Thrust_pressure_term_subsonic);
               printf("                                      \n\n");


      /*----------------------------------------------------------------------*/
      /*                                                                      */
      /*   E.6  Wrap up the full power calculations and define constants      */
      /*        to normalize changes during the postburn session.             */
      /*                                                                      */
      /*----------------------------------------------------------------------*/

         A_throat_FP      = A_throat;
         Isp_FP           = I_sp;
         Power_FP         = Power;
         dmdt_FP          = dm_dt;
         T_bulk_max_FP    = T_bulk_max;
         T_coating_max_FP = T_coating_max;
         T_fuel_max_FP    = T_fuel_center_max;
         Thrust_FP        = Thrust;
         Impulse_FP       = Thrust * burn_time;

         printf("*********************************************************\n");
         printf("*                                                       *\n");
         printf("*  F.  End of Burn.                                     *\n");
         printf("*                                                       *\n");
         printf("*********************************************************\n");
         printf("                                                         \n");
         printf("    1.  t_burntime (s):      %f\n", burn_time               );
         printf("    2.  Impulse (MN-s):      %f\n", Impulse_FP/1e6          );
         printf("                                                         \n");

         end_time     = clock();
         process_time = ((double) (end_time - start_time)) / CLOCKS_PER_SEC;


/*----------------------------------------------------------------------------*/
/*                                                                            */
/*   F.  Print out results.                                                   */
/*                                                                            */
/*----------------------------------------------------------------------------*/

   printf("****************************************************************\n");
   printf("*                                                              *\n");
   printf("*                  Program Output SNRE                         *\n");
   printf("*                                                              *\n");
   printf("****************************************************************\n");
   printf("                                                                \n");
   printf("A. Input Values                                                 \n");
   printf("                                                                \n");
   printf("   1. General Reactor Parameters                                \n");
   printf("      a. Mass flowrate (kg/s), (lbm/s):                       %f, %f\n",
                   dm_dt, dm_dt*2.2);
   printf("      b. Core Reactor Power (MW):                             %f\n",
                    Power_FP/1e6);
   printf("      c. Coolant reactor inlet temperature (K):               %f\n",
                    T_reactor_inlet);
   printf("      d. Pressure at the reactor inlet (MPa):                 %f\n",
                    p_reactor_inlet / 1e6);
   printf("      e. Area ratio of exit to throat:                        %f\n",
                    A_ratio_exit_to_throat);
   printf("      f. Burntime (s):                                        %f\n",
                    burn_time);
   printf("      g. Postburn time (s):                                   %f\n",
                    postburn_time);
   printf("      h. Temperature of the hydrogen tank (K):                %f\n",
                    T_LH2_tank);
   printf("                                                                \n");
   printf("   2. Reactor Geometry                                          \n");
   printf("      a. Reactor mixture extrapolation length (m):            %f\n",
                    extrapolation_length);
   printf("      b. Reactor diameter (m):                                %f\n",
                    D_reactor);
   printf("                                                                \n");
   printf("   3. Fuel Element Geometry                                     \n");
   printf("      a. Coolant channel diameter (mm):                       %f\n",
                    D_channel*1e3);
   printf("      b. Coolant channel pitch (mm):                          %f\n",
                    pitch*1e3);
   printf("      c. Fuel element length (m):                             %f\n",
                    L_reactor);
   printf("      d. Fuel element width, flat-to-flat (mm):               %f\n",
                    D_fuel_element*1e3);
   printf("      e. Coating thickness (microns):                         %f\n",
                    coating_thickness*1e6);
   printf("      f. Number of coolant channels:                          %d\n",
                    N_coolant_channels);
   printf("      g. Number of fuel assemblies:                           %d\n",
                    N_fuel_elements);
   printf("                                                                \n");
   printf("   4. Program Parameters                                        \n");
   printf("      a.  Number of axial divisions in the reactor:           %d\n",
                     N_z_increments);
   printf("      b.  Temperature increment for the coolant (K):          %f\n",
                     T_increment);
   printf("      c.  Maximum time for the post-burn simulation (s):      %f\n",
                     t_max);
   printf("      d.  Maximum allowable temperature in fuel (K):          %f\n",
                     T_fuel_max_allowable);
   printf("                                                                \n");
   printf("B. Computed Values                                              \n");
   printf("                                                                \n");
   printf("   1. Andreas equivalent geometry                               \n");
   printf("      a. Coolant channel area (mm^2):                         %f\n",
                    A_channel * 1e6);
   printf("      b. Total unit cell area (mm^2):                         %f\n",
                    A_total_cell * 1e6);
   printf("      c. Equivalent fuel element area (mm^2):                 %f\n"
                    A_fuel_equivalent * 1e6);
   printf("      d. Equivalent fuel radius (mm):                         %f\n",
                    R_fuel * 1e3);
   printf("      e. Area of the fuel element (mm^2):                     %f\n",
                    A_fuel_element * 1e6);

   printf("      f. Cross-sectional area of the fuel (mm^2):             %f\n",
                    A_fuel * 1e6);
   printf("      g. Channel radius (modified by coating) (mm):           %f\n",
                    R_channel * 1e3);
   printf("      h. Outer radius of the channel coating (mm):            %f\n",
                    R_coating * 1e3);
   printf("      i. Extrapolated height of the reactor (m):              %f\n", H_e);
   printf("                                                                \n");
   printf("   2. Power Produced                                            \n");
   printf("      a. (dq/dz) at origin, 0, (W/m):                         %f\n",
                     dq_dz_center);
   printf("      b. Maximum fuel T(K), z(m), z/Reactor Length:       %f, %f, %f\n",
          T_fuel_center_max, z_fuel_center_max, (z_fuel_center_max + L_reactor/2
          / L_reactor);
   printf("      c. Maximum coatingT(K), z(m), z/Reactor Length:     %f, %f, %f\n",
          T_coating_max, z_coating_max, (z_coating_max + L_reactor/2) / L_reactor);
   printf("      d. Maximum bulk T(K), z(m), z/Reactor Length:       %f, %f, %f\n",
          T_bulk_max, z_bulk_max, (z_bulk_max + L_reactor/2) / L_reactor);
   printf("      e. Heat lost preheating H2 (MW), (%):                   %f, %f\n",
          Q_preheat/1e6, Q_preheat * 100/Power);
   printf("      f. Reactor power (MW):                                  %f\n",
                    Power_FP/1e6);
   printf("      g. Effective reactor power (MW):                        %f\n",
                    Power_effective/1e6);
   printf("      h. Latent heat from tank to reactor (MW):               %f\n",
                    Q_latent_heat/1e6);
   printf("      i. Heat to vapor from tank to reactor (MW):             %f\n",
                    Q_vapor_to_reactor/1e6);
   printf("      j. Power to the turbopump (MW):                         %f\n",
                    Power_turbopump/1e6);
   printf("      k. Total preheat (Latent + vapor heat (MW):             %f\n",
                    Q_preheat/1e6);
   printf("                                                                \n");
   printf("   3. Reactor Coolant Channel Entrance.                         \n");
   printf("      a. Mass flowrate in single channel (kg/s):              %f\n",
                    dmdt_channel);
   printf("      b. Area of a single coolant channel (m^2):              %f\n",
                    A_channel);
   printf("                                                                \n");
   printf("   4. Reactor Coolant Channel Exit.                             \n");
   printf("      a. T (K):                                               %f\n",
                    T_reactor_exit);
   printf("      b. p (MPa),:                                            %f\n",
                    p_reactor_exit/1e6);
   printf("                                                                \n");
   printf("   5. Reactor Lower Chamber Values.                             \n");
   printf("      a. M:                                                   %f\n",
                    M_chamber);
   printf("      b. T (K):                                               %f\n",
                    T_chamber);
   printf("      c. p (MPa):                                             %f\n",
                    p_chamber/1e6);
   printf("      d. Area (m^2):                                          %f\n",
                    A_chamber);
   printf("      e. Stagnation T (K):                                    %f\n",
                    T_chamber_stagnation);
   printf("      f. Stagnation p (MPa):                                  %f\n",
                    p_chamber_stagnation/1e6);
   printf("                                                                \n");
   printf("   6.  Throat Values.                                           \n");
   printf("      a. M:                                                   %f\n", M_throat);
   printf("      b. T (K):                                               %f\n", T_throat);
   printf("      c. p (MPa):                                             %f\n",
                    p_throat/1e6);
   printf("      d. Area (m^2):                                          %f\n", A_throat);
   printf("                                                                \n");
   printf("   7.  Exit Area Values.                                        \n");
   printf("      a. M:                                                   %f\n", M_exit);
   printf("      b. T (K):                                               %f\n", T_exit);
   printf("      c. p (MPa):                                             %f\n",
                    p_exit/1e6);
   printf("      d. Area (m^2):                                          %f\n", A_exit);
   printf("      e. Exit velocity (m/s):                                 %f\n", V_exit);
   printf("      f. Thrust (N, lbf):                                     %f, %f\n",
                    Thrust, Thrust * 0.2248);
   printf("      g. Thrust (precent momentum, percent pressure):         %f, %f\n",
                    Thrust_momentum_term * 100 / Thrust, Thrust_pressure_term *
                    100 / Thrust);
   printf("      h. Total Impulse during burn (N-s):                     %f\n",
                    Thrust * burn_time);
```

```
   printf("      i. Hydrogen propellant consumed during burn (kg):       %f\n",
                    dm_dt * burn_time);
   printf("      j. Burn time (s):                                       %f\n",
                    burn_time);
   printf("      k. Processing time (s):                                %lf\n",
                    process_time);
   printf("                                                              \n");


   /*---------------------------------------------------------------------*/
   /*                                                                     */
   /*  F.1  Write out the temperature distribution in the fuel, bulk H2,  */
   /*       and surface as functions of "z".                              */
   /*                                                                     */
   /*---------------------------------------------------------------------*/

        fp1 = fopen("output.dat", "w");

     for(i = 0; i < N_z_increments; ++i)  {
        fprintf(fp1, "%f,  %f,  %f,  %f\n",
                axial_position[i] / L_reactor, T_bulk[i], T_wall[i], T_fuel_max[i]);
     }

        fclose(fp1);




/*-----------------------------------------------------------------------------*/
/*                                                                             */
/*   G.  Begin the PostBurn Iteration!!!                                       */
/*                                                                             */
/*       This section computes the rocket performance after the full burn     */
/*       has been completed and variable cross-sectional nozzle is used       */
/*       along with hydrogen gas that removes decay heat.                      */
/*                                                                             */
/*-----------------------------------------------------------------------------*/

       M_H2_used        = dm_dt * burn_time;
       Integral_Impulse = Thrust * burn_time;


         fp1 = fopen("postburn.dat", "w");
         fprintf(fp1, "Full Reactor Power (MW):     %f\n", Power_FP / 1e6);
         fprintf(fp1, "I_sp (s) at full power:      %f\n", Isp_FP);
         fprintf(fp1, "dm_dt (kg/s) at full power:  %f\n", dmdt_FP);
         fprintf(fp1, "Burn Time (s):               %f\n", burn_time);
         fprintf(fp1, "                               \n");
         fprintf(fp1, "A*_ratio, dm/dt_ratio, power_ratio, I_sp(s), Thrust_ratio,
                       Impulse_ratio\n");

      for(t = t_increment; t <= t_max; t += t_increment)  {

         Power = Power_FP * decay_heat_ratio(burn_time, t);

         printf("\n\n");
         printf("*********************************************************\n");
         printf("*  Time After Burn = %f (s), Power (MW) = %f\n", t, Power/1e6);
         printf("*********************************************************\n");
         printf("\n");

         /*-------------------------------------------------------------*/
         /*                                                             */
         /*  G.0.a  Compute the mass flowrate that will remove this heat. */
         /*                                                             */
         /*-------------------------------------------------------------*/

            dm_dt = Power / (integral_CpdT(T_fuel_max_allowable) -
                    integral_CpdT(T_reactor_inlet));
```

```
/*---------------------------------------------------------------------*/
/*                                                                     */
/*  G.1  How much heat is required to bring the hydrogen temperature   */
/*        up from "T_LH2_tank" to "T_reactor_inlet"?  Adjust the       */
/*       heat that goes into the hydrogen flowing through the          */
/*       reactor, appropriately. Also, include the power required      */
/*       by the turbopump since it constitutes a small power drain.    */
/*                                                                     */
/*---------------------------------------------------------------------*/

      Q_vapor_to_reactor = dm_dt * (integral_CpdT(T_reactor_inlet) -
                           integral_CpdT(T_LH2_tank));

      Q_latent_heat      = dm_dt * latent_heat_H2();

      Q_preheat          = Q_latent_heat + Q_vapor_to_reactor;
      Power_turbopump    = dm_dt * (p_turbopump - 0.0) / density_H2(T_LH2_tank);

      printf("A.  Estimate of Small Power Losses\n");
      printf("    1.  Power_turbopump (MW)  = %f\n", Power_turbopump / 1e6);
      printf("    2.  p_turbopump (MPa)     = %f\n", p_turbopump / 1e6);
      printf("    3.  dm_dt (kg/s)          = %f\n", dm_dt);
      printf("    4.  T_LH2_tank (K)        = %f\n", T_LH2_tank);
      printf("    5.  density_H2 (kg/m^3)   = %f\n", density_H2(T_LH2_tank));
      printf("    6.  latent_heat_H2 (J/kg) = %f\n", latent_heat_H2());
      printf("    7.  Q_latent_heat (MW)    = %f\n", Q_latent_heat/1e6);
      printf("                                  \n");

      Power_effective    = Power - Q_preheat - Power_turbopump;


/*---------------------------------------------------------*/
/*                                                         */
/*   G.2  This iteration finds the bulk hydrogen coolant   */
/*        temperature along the reactor length (z=0 to     */
/*        L_reactor).  This is based on a simple energy    */
/*        balance.  Ignoring axial conduction along the    */
/*        fuel elements, the heat generated by a slice of  */
/*        the equivalent fuel of width "dz" must go into   */
/*        the hydrogen resulting in an increase of "dT"    */
/*        in the bulk temperature.                         */
/*                                                         */
/*---------------------------------------------------------*/

   dz           = L_reactor / N_z_increments;
   pressure[0]  = p_reactor_inlet;
   dq_dz_center = H_e * sin(PI * L_reactor / (2 * H_e));

for(z = -L_reactor/2 + (dz/2), T_b = T_reactor_inlet, i = 0;
    z <= L_reactor/2; z += dz, ++i)                    {

      axial_position[i] = z;


   /*----------------------------------------------------------------------*/
   /*                                                                      */
   /*  G.2.a  First, compute the heat generated, "dq_total", by a          */
   /*         thickness of the reactor integrated from entrance            */
   /*         at -(L_reactor/2) to z.                                      */
   /*                                                                      */
   /*----------------------------------------------------------------------*/

      dq_total =   (Power / 2) *
                 (  1 + ( sin(PI * z / H_e) /
                 sin((PI * L_reactor) / (2 * H_e)) )  );


   /*----------------------------------------------------------------------*/
   /*                                                                      */
   /*  G.2.b  Next, compute the change in the bulk coolant temperature     */
   /*         corresponding to "dq_total" within the interval "z" from     */
   /*         the reactor entrance.                                        */
   /*                                                                      */
   /*----------------------------------------------------------------------*/

      T       = T_b;
      oldsign = sgn(dm_dt * (integral_CpdT(T)
                           - integral_CpdT(T_reactor_inlet))
                - dq_total);

      for(T = T + T_increment; T < 6000; T += T_increment)  {
         newsign = sgn(dm_dt * (integral_CpdT(T)
                              - integral_CpdT(T_reactor_inlet))
                   - dq_total);
         if (newsign != oldsign)  {
            T_b = T;
            break;
         }
      }

      T_bulk[i] = T_b;


   /*----------------------------------------------------------------------*/
   /*                                                                      */
   /*  G.2.c  OK, now compute the velocity through each coolant channel.   */
   /*                                                                      */
   /*----------------------------------------------------------------------*/

      dmdt_channel = dm_dt / (N_coolant_channels * N_fuel_elements);
      V_channel[i] = dmdt_channel / (A_channel * density_H2(T_b));
      M_channel[i] = V_channel[i] / speed_of_sound_H2(T_b);


   /*---------------------------------------------------------------------*/
   /*                                                                     */
   /*  G.2.d  Calculate the wall temperature of the coolant channel.      */
   /*                                                                     */
   /*     Re_f  = Reynolds number of the bulk fluid at "z".               */
   /*     Pr_f  = Prandtl number of the bulk fluid at "z".                */
   /*     Nu_f  = Nusselt number calculated at "z".                       */
   /*     dq_reactor = total reactor power produced between z             */
   /*                  and z+dz.                                          */
   /*     dq_coolant_channel  = total power power produced in single      */
   /*                  coolant channel from z to z+dz.                    */
   /*     dq_dA = power produced in single coolant channel per            */
   /*             unit surface area facing the coolant.                   */
   /*     M_f   = Mach number in the bulk fluid at "z".                   */
   /*     gamma = ratio of specific heats at "z".                         */
   /*     T_fs  = temperature of the fluid at the channel surf.           */
   /*     T_fa  = temperature at wall adjusted for Mach number.           */
   /*                                                                     */
   /*---------------------------------------------------------------------*/

      dq_reactor         = (Power/2) * (sin(PI * (z + dz)/H_e) - sin(PI * z / H_e))
                            / sin(PI * L_reactor / (2 * H_e));
      dq_coolant_channel = dq_reactor / (N_coolant_channels * N_fuel_elements);
      dq_dA              = dq_coolant_channel / (2 * PI * R_channel * dz);

      Re_f               = Reynolds_number(T_b, dmdt_channel, R_channel);
      Pr_f               = Prandtl_number_H2(T_b);
      M_f                = M_channel[i];
      F_R                = pow(Prandtl_number_H2(T_b), 0.3333);
      gamma              = ratio_of_specific_heats_H2(T_b);
      T_fs               = T_b * (1 + (gamma - 1) * M_f * M_f / 2);
      T_fa[i]            = F_R * (T_fs - T_b) + T_b;


   /*---------------------------------------------------------------------*/
   /*                                                                     */
   /*  G.2.e  Pressure drop to this value of z.                           */
   /*                                                                     */
```

```
   /*---------------------------------------------------------------------*/

         pressure[i+1] = pressure[i] - f_friction(Re_f) * (dz / D_channel)
                         * V_channel[i] * V_channel[i] / 2.0;

         Nu_f          = Nu_Dittus_Boelter(Re_f, Pr_f);
         h             = Nu_f * thermal_conductivity_H2(T_b) / D_channel;
         T_wall[i]     = dq_dA / h + T_fa[i];


   /*---------------------------------------------------------------------*/
   /*                                                                     */
   /*  G.2.f  Next, find the temperature in the coating and in the fuel.  */
   /*                                                                     */
   /*---------------------------------------------------------------------*/

         Volume_fuel   = N_coolant_channels * N_fuel_elements * A_fuel
                         * L_reactor;
         dq_dV[i]      = - dq_reactor / Volume_fuel;
         T_coating[i]  = ((R_fuel * R_fuel - R_coating * R_coating) * dq_dV[i]
                         / (2 * k_coating(T_wall[i])))
                         * log(R_channel / R_coating) + T_wall[i];
         T_fuel_max[i] = (dq_dV[i] / (4 * k_fuel(T_coating[i])))
                         * (R_fuel * R_fuel - R_coating * R_coating
                         + 2 * R_fuel * R_fuel * log(R_coating / R_fuel))
                         + T_coating[i];

}



/*-------------------------------------------------------------------------*/
/*                                                                         */
/*   G.3  DeLaval Nozzle Calculations.                                     */
/*                                                                         */
/*       At this point, all temperatures and pressures have been computed  */
/*       in the reactor.  Compute the properties in the DeLaval nozzle,    */
/*       including the thrust.                                             */
/*                                                                         */
/*-------------------------------------------------------------------------*/

   /*---------------------------------------------------------------------*/
   /*                                                                     */
   /*   G.3.a  Compute the maximum fuel, bulk and coating temperatures.   */
   /*                                                                     */
   /*---------------------------------------------------------------------*/

      T_bulk_max        = T_bulk[0];
      T_coating_max     = T_coating[0];
      T_fuel_center_max = T_fuel_max[0];

   for(i = 0; i < N_z_increments; ++i)  {

      if (T_bulk_max < T_bulk[i])  {
         T_bulk_max = T_bulk[i];
         z_bulk_max = axial_position[i];
      }

      if (T_coating_max < T_coating[i])  {
         T_coating_max = T_coating[i];
         z_coating_max = axial_position[i];
      }

      if (T_fuel_center_max < T_fuel_max[i])  {
         T_fuel_center_max = T_fuel_max[i];
         z_fuel_center_max = axial_position[i];
      }

   }

      printf("\n");
```

```
/*---------------------------------------------------------------------*/
/*                                                                     */
/*  G.4  Compute the stagnation values and specific impulse            */
/*       in the lower reactor chamber.                                 */
/*                                                                     */
/*     1.  The H2 expands as it exits the coolant channels             */
/*         and enters the chamber.                                     */
/*     2.  Compute p, T, and M in the chamber.                         */
/*     3.  Compute stagnation properties in the chamber.               */
/*     4.  Compute the properties at the throat.                       */
/*     5.  Compute exit Mach number and velocity.                      */
/*     6.  Compute specific impulse and thrust.                        */
/*                                                                     */
/*                                                                     */
/*---------------------------------------------------------------------*/

      T_reactor_exit    = T_bulk[N_z_increments - 1];
      p_reactor_exit    = pressure[N_z_increments - 1];

      k                 = ratio_of_specific_heats_H2(T_reactor_exit);

      A_coolant_channel = PI * R_channel * R_channel;
      A_reactor_exit    = A_coolant_channel * N_coolant_channels
                           * N_fuel_elements;
      M_reactor_exit    = M_channel[N_z_increments - 1];
      T_stagnation      = T_reactor_exit * (1 + ((k - 1) / 2) *
                          M_reactor_exit * M_reactor_exit);
      p_stagnation      = p_reactor_exit * pow(  (1 + ((k - 1) / 2) *
                          M_reactor_exit * M_reactor_exit), k / (k - 1)  );

         /*  Properties in the lower reactor chamber.  */

      D_chamber         = D_reactor;
      A_chamber         = (PI / 4) * D_chamber * D_chamber;

      printf("B.  Reactor exit to the lower reactor chamber.  \n");
      printf("    1.  T_reactor_exit (K)       = %f\n", T_reactor_exit);
      printf("    2.  p_reactor_exit (MPa)     = %f\n", p_reactor_exit / 1e6);
      printf("    3.  A_coolant_channel (mm^2) = %f\n", A_coolant_channel * 1e6);
      printf("    4.  A_reactor_exit (m^2)     = %f\n", A_reactor_exit);
      printf("    5.  A_chamber (m^2)          = %f\n", A_chamber);
      printf("    6.  D_chamber (mm)           = %f\n", D_chamber * 1e3);
      printf("    7.  M_reactor_exit           = %f\n", M_reactor_exit);
      printf("    8.  T_stagnation (K)         = %f\n", T_stagnation);
      printf("    9.  p_stagnation (MPa)       = %f\n", p_stagnation/1e6);
      printf("   10.  k                        = %f\n", k);
      printf("                                     \n");


/*---------------------------------------------------------------------*/
/*                                                                     */
/*   G.5  Compute the Mach number in the lower reactor                 */
/*        chamber by iteration.                                        */
/*                                                                     */
/*---------------------------------------------------------------------*/


      A_ratio   = A_chamber / A_reactor_exit;
      k         = ratio_of_specific_heats_H2(T_reactor_exit);
      M_chamber = Mach_Area_Ratio(A_ratio, M_reactor_exit, k, 0);

      c         = (k - 1) / 2;
      T_chamber = T_reactor_exit * (1 + c * M_chamber * M_chamber)
                                 / (1 + c * M_reactor_exit * M_reactor_exit);
      p_chamber = p_reactor_exit * pow( (1 + c * M_chamber * M_chamber)
                                   / (1 + c * M_reactor_exit * M_reactor_exit),
                                   (k / (k - 1)) );
      T_chamber_stagnation = T_chamber * (1 + c * M_chamber * M_chamber);
      p_chamber_stagnation = p_chamber * pow(1 + c * M_chamber * M_chamber,
                                             k / (k - 1) );

      printf("C.  Values in the DeLaval Nozzle Upper Chamber  \n");
      printf("    1.  Chamber Mach number                 = %f\n", M_chamber);
      printf("    2.  Area ratio (chamber / reactor exit) = %f\n", A_ratio);
      printf("    3.  k                                   = %f\n", k);
      printf("    4.  T_chamber (K)                       = %f\n", T_chamber);
      printf("    5.  p_chamber (MPa)                     = %f\n", p_chamber / 1e6);
      printf("    6.  M_chamber                           = %f\n", M_chamber);
      printf("    7.  T_chamber_stagnation (K)            = %f\n",
                      T_chamber_stagnation);
      printf("    8.  p_chamber_stagnation (MPa)          = %f\n",
                      p_chamber_stagnation / 1e6);
      printf("                                                \n");


/*----------------------------------------------------------------------*/
/*                                                                      */
/*   G.6  Next, calculate what happens from the reactor chamber to     */
/*        the throat.                                                   */
/*                                                                      */
/*----------------------------------------------------------------------*/

       k        = ratio_of_specific_heats_H2(T_chamber);
       A_throat = dm_dt / (p_chamber_stagnation *
                  sqrt(   (k / (R_H2 * T_chamber_stagnation)) *
                  pow(  2 / (k + 1), (k - 1) / (k + 1) )  )   );
       T_throat = T_chamber * (2 + (k - 1) * M_chamber * M_chamber) /
                              (2 + (k - 1) )  ;
       p_throat = p_chamber * pow( (T_throat / T_chamber), k / (k - 1) );
       M_throat = 1;
       D_throat = sqrt( (4/PI) * A_throat );

       printf("D.  Throat Values.\n");
       printf("    1.  A_throat (mm^2) = %.8e\n", A_throat * 1e6);
       printf("    2.  p_throat (MPa)  = %.8e\n", p_throat / 1e6);
       printf("    3.  T_throa1 (K)    = %f\n",   T_throat);
       printf("    4.  M_throat        = %f\n",   M_throat);
       printf("    5.  k               = %f\n",   k);
       printf("    6.  D_throat (mm)   = %f\n",   D_throat * 1e3);
       printf("    7.  dm/dt (kg/s)    = %f\n",   dm_dt);
       printf("    8.  R (H2) (J/kg-K) = %f\n",   R_H2);
       printf("    \n");


/*----------------------------------------------------------------------*/
/*                                                                      */
/*   G.7  Calculate values for the DeLaval Nozzle exit.                 */
/*                                                                      */
/*----------------------------------------------------------------------*/

         k                      = ratio_of_specific_heats_H2(T_reactor_exit);
         A_exit                 = (PI/4) * D_exit * D_exit;
                                   // (m^2).
         A_ratio_exit_to_throat = A_exit / A_throat;

   /*------------------------------------------------------------------*/
   /*                                                                  */
   /*   G.7.a  Compute values at the exit for the supersonic flow     */
   /*          solution.  This is valid if the throat area is the     */
   /*          correct diameter to choke the flow.                    */
   /*                                                                  */
   /*------------------------------------------------------------------*/

         M_exit = Mach_Area_Ratio_to_Throat(A_ratio_exit_to_throat, k, 1);

         T_exit = T_throat * (k + 1) / ( 2 + (k - 1) * M_exit * M_exit);  // (K).
         p_exit = p_throat * pow( (T_exit / T_throat), (k / (k - 1)) );   // (Pa).
         a_exit = sqrt(k * R_H2 * T_exit);                                // (m/s).
         V_exit = M_exit * a_exit;                                        // (m/s).
         Thrust_momentum_term = dm_dt * V_exit;                           // (N).
         Thrust_pressure_term = p_exit * A_exit;                          // (N).

         Thrust = dm_dt * V_exit + p_exit * A_exit;                       // (N).

         I_sp   = Thrust / (dm_dt * 9.81);                                // (s).


         printf("G.6.  Exit Values.\n");
         printf("    1.  k                        = %f\n", k);
         printf("    2.  M_exit                   = %f\n", M_exit);
         printf("    3.  A_exit (m^2)             = %f\n", A_exit);
         printf("    4.  T_exit (K)               = %f\n", T_exit);
         printf("    5.  p_exit (MPa)             = %f\n", p_exit / 1e6);
         printf("    6.  a_exit (m/s)             = %f\n", a_exit);
         printf("    7.  V_exit (m/s)             = %f\n", V_exit);
         printf("    8.  Thrust, momentum term (N) = %f\n", Thrust_momentum_term);
         printf("    9.  Thrust, pressure term (N) = %f\n", Thrust_pressure_term);
         printf("   10.  Total Thrust (N)          = %f\n", Thrust);
         printf("        Total Thrust (klbf)       = %f\n", Thrust * 0.224809 /
                         1000);
         printf("   11.  Specific Impulse (s)      = %f\n", I_sp);
         printf("   12.  D_exit (mm)               = %f\n", D_exit * 1e3);
         printf("                                      \n");


   /*----------------------------------------------------------------*/
   /*                                                                */
   /*   G.7.b  Compute values at the exit for the subsonic flow      */
   /*          solution.  This is valid if the throat area is too    */
   /*          large to choke the flow.                              */
   /*                                                                */
   /*----------------------------------------------------------------*/

         M_exit_subsonic = Mach_Area_Ratio_to_Throat(A_ratio_exit_to_throat, k, 0);

         T_exit_subsonic = T_throat * (k + 1) / ( 2 + (k - 1) *
                           M_exit_subsonic * M_exit_subsonic);            // (K).
         p_exit_subsonic = p_throat * pow( (T_exit_subsonic / T_throat),
                         (k / (k - 1)) );                                   // (Pa).
         a_exit_subsonic = sqrt(k * R_H2 * T_exit_subsonic);              // (m/s).
         V_exit_subsonic = M_exit_subsonic * a_exit_subsonic;             // (m/s).

         Thrust_momentum_term_subsonic = dm_dt * V_exit_subsonic;         // (N).
         Thrust_pressure_term_subsonic = p_exit_subsonic * A_exit;        // (N).

         Thrust_subsonic = dm_dt * V_exit_subsonic + p_exit_subsonic * A_exit;
                                                                          // (N).

         I_sp_subsonic   = Thrust_subsonic / (dm_dt * 9.81);              // (s).


         printf("G.6. Exit Values for Subsonic Exit.\n");
         printf("   1.  T (K)                      = %f\n", T_exit_subsonic);
         printf("   2.  p (MPa)                    = %f\n", p_exit_subsonic/1e6);
         printf("   3.  Thrust, subsonic (N)       = %f\n", Thrust_subsonic);
         printf("   4.  Mach number                = %f\n", M_exit_subsonic);
         printf("   5.  I_sp_subsonic (s)          = %f\n", I_sp_subsonic);
         printf("   6.  Speed of sound (m/s)       = %f\n", a_exit_subsonic);
         printf("   7.  Exhaust velocity (m/s)     = %f\n", V_exit_subsonic);
         printf("   8.  Thrust due to momentum (N) = %f\n",
                        Thrust_momentum_term_subsonic);
         printf("   9.  Thrust due to pressure (N) = %f\n",
                        Thrust_pressure_term_subsonic);
         printf("                                     \n\n");


/*----------------------------------------------------------------------*/
/*                                                                      */
/*   G.8  Wrap up the partial power calculations and define constants   */
/*        to normalize changes during the postburn session.             */
/*                                                                     */
/*---------------------------------------------------------------------*/

   A_throat_current      = A_throat;
   Isp_current           = I_sp;
   Power_current         = Power;
   dmdt_current          = dm_dt;
   T_bulk_max_current    = T_bulk_max;
   T_coating_max_current = T_coating_max;
   T_fuel_max_current    = T_fuel_center_max;
   Thrust_current        = Thrust;
   Impulse_current       = Thrust * t_increment;
   Impulse_SUM          += Impulse_current;
   Integral_Impulse     += Impulse_current;
   M_H2_used            += dm_dt * t_increment;

   /*----------------------------------------------------------------*/
   /*                                                                */
   /*  G.8.a  Compute the wall clock time that has expired since the */
   /*         program started.                                       */
   /*                                                                */
   /*----------------------------------------------------------------*/

      end_time     = clock();
      process_time = ((double) (end_time - start_time)) / CLOCKS_PER_SEC;


/*---------------------------------------------------------------------*/
/*                                                                     */
/*   G.9  Print out the results for each post-burn iteration.          */
/*                                                                     */
/*---------------------------------------------------------------------*/

   printf("                                                           \n");
   printf("G.7  POST-BURN:  Final values at time %f (s).              \n", t);
   printf("                                                           \n");
   printf("   1. Performance Parameters                               \n");
   printf("      a. Decay heat power output (MW):                   %f\n",
                    Power / 1e6);
   printf("      b. Current specific impulse, Isp (s):              %f\n",
                 Isp_current);
   printf("                                                           \n");
   printf("   2. Temperatures                                         \n");
   printf("      a. Maximum H2 temperature at reactor exit (K):     %f\n",
                 T_bulk_max_current);
   printf("      b. Maximum fuel coating temperature (K):           %f\n",
                 T_coating_max_current);
   printf("      c. Maximum fuel temperature (K):                   %f\n",
                 T_fuel_max_current);
   printf("                                                           \n");
   printf("   3.Throat Area                                           \n");
   printf("      a. Throat area in the variable area nozzle (m^2):  %f\n",
                 A_throat_current);
   printf("      b. Throat area / full-burn throat area:            %f\n",
                 A_throat_current / A_throat_FP);
   printf("                                                           \n");
   printf("   4. Thrust                                               \n");
   printf("      a. Current Thrust (N):                             %f\n",
                 Thrust_current);
   printf("      b. Thrust / Thrust full burn:                      %f\n",
                 Thrust_current / Thrust_FP);
   printf("                                                           \n");
   printf("   5. Hydrogen Propellant Usage                            \n");
   printf("      a. Current mass flowrate of H2, dm/dt (kg/s):      %f\n", dm_dt);
   printf("      b. Mass of hydrogen propellant used (kg):          %f\n",
                 M_H2_used);
   printf("                                                           \n");
   printf("   6. Specific Impulse                                     \n");
   printf("      a. Integral Impulse (GN-s):                        %f\n",
                 Integral_Impulse/1e9);
   printf("      b. Current Impulse (N-s):                          %f\n",
                         Impulse_current);
            printf("      c. Impulse postburn sum / Impulse from full burn:  %f\n",
                             Impulse_SUM / Impulse_FP);
            printf("                                                           \n");
            printf("   7. Processing Parameters                                \n");
            printf("      a. Total processing time (s):                       %f\n",
                             process_time);


            fprintf(fp1, "%f  %f  %f  %f  %f  %f  %f  %f  %f\n", t, A_throat_current /
                    A_throat_FP,
                    dm_dt / dmdt_FP, Power_current / Power_FP, Isp_current,
                    Thrust_current / Thrust_FP, Impulse_SUM / Impulse_FP), Integral_Impulse,
                    M_H2_used;



/*

            printf("*********************************************************\n");
            printf("*                                                       *\n");
            printf("*  G.9.  End of this time iteration.                    *\n");
            printf("*                                                       *\n");
            printf("*********************************************************\n");
            printf("                                                         \n");
            printf("    A.  Time                                             \n");
            printf("        1.  time (s):                                  %f\n", t);
            printf("        2.  burn time (s):                             %f\n", burn_time);
            printf("        3.  delay time [t - burn time], (s):           %f\n",
                            t - burn_time);
            printf("                                                         \n");
            printf("    B.  Calculations                                     \n");
            printf("        1.  Impulse (MN-s):                            %f\n",
                            Impulse_FP/1e6);
            printf("                                                         \n");
*/


         }     /*  This terminates the post-burn time iteration loop.  */

            fclose(fp1);
}
```

REFERENCES

Benensky, Kelsa M.. “Summary of Historical Solid Core Nuclear Thermal Propulsion Fuels.” (2013).

R. E. Bilstein & W. R. Lucas. (1980). Stages to Saturn: A Technological History of the Apollo/Saturn Launch Vehicle. Washington, DC: U.S. National Aeronautics and Space Administration. ISBN 0-16-048909-1. OCLC 36332191.

S. K. Borowski, D. R. McCurdy & T. W. Packard, “Nuclear Thermal Propulsion (NTP): A Proven Growth Technology for Human NEO/Mars Exploration Missions,” NASA Report, (2010).

M. W. Chase, Jr., “NIST-JANAF Thermochemical Tables,” 4th edition, *J. Phys. Chem. Ref. Data*, Monograph 9, (1998).

W. Culbreth & K. Gonzalez, “Capture of Decay Heat to Produce Thrust in a Nuclear Propulsion Rocket,” ANS Transactions, ANS Winter Annual Conference, Washington, D.C., (2019).

G. Dantzig. (May 1987). "Origins of the simplex method" (PDF). In Nash, Stephen G. (ed.). A History of Scientific Computing. Association for Computing Machinery. pp. 141–151. doi:10.1145/87252.88081. ISBN 978-0-201-50814-7.

Y. Demyanenko, A. Dmitrenko; A. Ivanov; V. Pershin; et al. (July 2005). Ground Test Demonstrator Engine Boost Turbopumps Design and Development. *41st AIAA/ASME/SAE/ASEE Joint Propulsion Conference & Exhibit*. Tucson, Arizona: American Institute of Aeronautics and Astronautics, Inc. AIAA 2005-3945.

J. J. Duderstadt, & L. J. Hamilton. (1976). *Nuclear Reactor Analysis,* John Wiley & Sons.

M. M. El-Wakil. (1962). *Nuclear Power Engineering*, McGraw-Hill, ISBN 07-019300-2.

Emrich, William Jr. (2016). *Principles of Nuclear Rocket Propulsion*. Oxford, UK: Elsevier Butterworth-Heinemann.

Fallon, Brandon, Rodrigez, Mariana, & Salas-Sanchez, Edgar. (1999). “Project VTRN: Variable Throat Rocket Nozzle”. UNLV Senior Design Report, University of Nevada, Las Vegas, Department of Mechanical Engineering.

J. L. Finseth. (1991). *Rover Nuclear Rocket Engine Program, Overview of Rover Engine Tests, Final Report*. NASA CR-184270, Feb., (1991).

O. G. B. Gaballa. (2012). Processing development of 4TaC-HfC and related carbides and borides for extreme environments. Ph.D. Dissertation, 124 pages, Iowa State University, 2012.

K. Gonzalez. (2021). Variable Area NTP Rocket Propellant Savings for Missions to Mars. American Astronautical Society, 2021 Wernher Von Braun Memorial Symposium, October 12-14, 2021.

K. Gonzalez & W. Culbreth. (2018). Relationship of Thermal Stresses and Bulk Coolant Temperature on the Corrosion of Nuclear Fission Rocket Fuel. 11th Wernher Von Braun Memorial Symposium, Oct. 23-25, 2018.

Grey, J., Nelson, S.T., & Williams, P.M., “Liquid-Core Reactor Concept Examined for Nuclear Rocket”, Jun 1, 1965, AIAA Paper 64-385, Journal of Spacecraft and Rockets, vol 2, May-June 1965, 384-391.

E. Howell . (2021). US Military Picks 3 Companies to Test Nuclear Propulsion in Cislunar Space. Space.com. Published April 20, 2021.

H. E. Khalifa, C. P. Deck, O. Gutierrez, G. M. Jacobsen, and C. A. Beck, "Fabrication and characterization of joined silicon carbide cylindrical components for nuclear applications," Journal of Nuclear Materials, Vol 457, pp. 227-240, Feb. 2015.

J. Khatry, F. Aydogan, M. Ilyas, and M. Houts. (2018). Design of a passive safety system for a nuclear thermal rocket: *Annals of Nuclear Energy*, **111**, 536-553, (2018).

Daniel R. Koenig, "Experience Gained from the Space Nuclear Rocket Program (Rover)," LA-10062-H, Los Alamos National Laboratory, 57 pages, May 1986.

Korea Atomic Energy Research Institute, Chart of the Nuclides, https://atom.kaeri.re.kr/.

S. Labib and J. King, “Design and analysis of a single stage to orbit nuclear thermal rocket reactor engine,” *Nuclear Engineering and Design*, 287, 36-47, (2015).

LaMarsh, J. R., & Baratta, A. J. (2001). *Introduction to Nuclear Engineering. 3rd ed.* Prentice Hall.

Leachman, J. W., Jacobsen, R. T., Penoncello, S. G., & Lemmon, E. W. (2009). “Fundamental Equations of State for Parahydrogen, Normal Hydrogen, and Orthohydrogen”. M. Phys. Chem. Ref. Data. Vol. 38. No 3. 0047-2689/2009/38(3)/721/28.

T. E. Llewellyn, “Variable Thrust Rocket Engine,” United States Patent Office, U.S. 3478965, Issued on Nov. 18, (1969).

J. Mick. (2009). Russia is Developing Nuclear Fission Spaceship to Reach the Red Planet. DailyTech.com, October 29, 2009.

N-Division. (1969). *Pewee I Reactor Test Report.* Rep. no. LA-4217-MS. Los Alamos, NM: Los Alamos National Laboratory.

Nam, Seung Hyun, Venneri, Paolo, Kim, Yonghee, Lee, Jeong Ik, Chang, Soon Heung, & Jeong, Yong Hoon. (2015). Innovative Concept for an Ultra-Small Nuclear Thermal Rocket Utilizing a New Moderated Reactor. Nuclear Engineering & Technology, vol 47, 678-799, Elsevier.

NASA. (1992). *Rover/NERVA-Derived Near-Term Nuclear Propulsion, FY92 Final Review at NASA-LeRC*: N93-26913, NP-TIM92, October 22, (1992).

Pelaccio, Dennis G. and El-Genk, Mohamed S., “A Review of Nuclear Thermal Propulsion Carbide Fuel Corrosion and Key Issues,” NASA-CR-197533, November 1, 1994.

I. Pioro. (2016). *Handbook of Generation IV Nuclear Reactors: A Guidebook*, ISBN 978081001493, Elsevier Science & Technology.

R. B. Pond and J. E. Matos, Nuclear Mass Inventory, Photon Dose Rate and Thermal Decay Heat of Spent Research Reactor Fuel Assemblies (Rev. 1): ANL/RERTR/TM-26, December, (1996).

D. C. Prince, Jr., “Variable Area Nozzle,” United States Patent Office, U.S. 3138921, Issued on June 30, (1964).

RIA Novosti. (2020). Russian Space Agency Confirms Plans to Launch Nuclear-Powered Space Tug by 2020. SpaceDaily.com, January 29, 2020.

F. E. Roach and O. E. Deumlet, “Variable Throat Rocket Nozzle,” United States Patent Office, U.S. 2552497, Issued May 8, (1951).

R. G. Ragsdale, “Performance Potential of a Radiant-Heat-Transfer Liquid-Core Nuclear Rocket Engine”, NASA TN D-4148, October 1967.

Rockwell International (1992), “Rover/NERVA-Derived Near-Term Nuclear Propulsion, FY92 Final Review at NASA-LeRC”. N93-26913, NP-TIM-2, Rockwell International.

R. P. Schmitt, “The design of a molten fueled, vortex reactor, for use as a rocket propulsion device,” M.S. Thesis, 1968, Missouri S&T, Curtis Lews Wilson Library. 67 pages.

W. A. Sanders & H. B. Probst, (1969). High-Temperature Mechanical Propertiesof Pol crystalline Hafnium Carbide and Hafnium Carbide Containing 13-Volume-Percent Hafnium Diboride, Lewis Research Center, NASA TN D-5008.

B. G. Schnitzler. (2012). Small Reactor Designs Suitable for Direct Nuclear Thermal Propulsion: Interim Report. January 2012, Idaho National Laboratory, INL/EXT-12-24776, 27 pages.

Sheth, A. & Leibowitz, L. (1975). "Equation of State and Transport Properties of Uranium and Plutonium Carbides in the Liquid Region. Argonne National Laboratory, ANL-AFP-11.

Storn, R.; Price, K. (1997). "Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces". Journal of Global Optimization. **11** (4): 341–359. doi:10.1023/A:1008202821328. S2CID 5297867.

G. P. Sutton. (1963). *Rocket Propulsion Elements, 3rd edition*. New York: John Wiley & Sons. pp. 5-25.

H. Twardy, "On a variable-thrust hydrogen-oxygen rocket engine," *Acta Astronautica*, **2**, pp. 627-647, (1975).

C. W. Watson. (1994). "Nuclear Rockets: High-Performance Propulsion for Mars. LA-12784-MS, UC-743. Los Alamos National Laboratory.

CURRICULUM VITAE

Graduate College

University of Nevada, Las Vegas

Kimberly Gonzalez

Contact Information

Email: kimgonz02@gmail.com

Education

Bachelor of Science in Engineering – Mechanical Engineering, 2018

University of Nevada, Las Vegas

Dissertation Title

Nuclear Thermal Propulsion Based on Molten Uranium Carbide Fuel

Thesis Examination Committee

Chairperson, Dr. William Culbreth

Committee Member, Dr. Alexander Barzilov

Committee Member, Dr. Yi-tung Chen

Graduate Faculty Representative, Dr. Monika Neda