2016

Modelling of Continuous Rod Withdraw and Pulse Transients in a TRIGA mk. II Reactor at The University of Texas at Austin using a Coupled Neutronics Thermal-Hydraulics Model

G. Kline

Table of Contents

List of Figures

List of Tables

Introduction

The recent relicensing process at The University of Texas at Austin called for reactor response analysis to transient reactivity events in the core. This motivation led to the creation of a transient analysis model and solution program, named the $\{\mathbb{R}^0, \mathbb{R}^2\}$ model.

The following outlines the model and the analysis of two specific events: a pulse and a continuous rod withdraw. Both simulations use a zero-dimensional neutronics model based on the In-hour equation found in an Idaho National Laboratory paper[1] and Akcasu's dynamics[2]. and equations from the 2015 UT LOCA model. Simnad provides the fuel specific properties such as material and geometric, while the UT TRIGA was used for the site-specific quantities including fuel loading, burnup, and rod worth; Weaver provides the neutronics data.

The neutronics portion takes into account the three major fissile isotopes' effects, as well as decay heat fractions and the current UT poison loading. MATLAB was chosen as the programming language for its ease, user input accessibility, and effective ODE solution methods for stiff and non-stiff equation sets. Consideration is taken to incorporate as many state dependent variables as possible to improve accuracy; however, the program is currently designed for events on the order of hours or less, where fuel loading and fission product poison concentration can be considered constant.

The verification process involved historical records for the pulse, and actual reactor operations for the rod withdraw portion. The whole program revolves around an accurate and easy user interface.

Prior Work

General Atomics

General Atomics (GA) did a number of studies around the early 1960's on the feasibility and behavior of a research reactor.[3] [4] The codes General Atomics used to develop the kinetic behavior (GAM-II, GATHER-II,GAZE-II,GAMBLE, and DTF-IV) are unavailable for use. The kinetics presented in these papers are well defined for two specific TRIGA configurations, neither of which are physically exacting enough to conditions at The University of Texas at Austin (UT) to determine valid local criteria.

Simnad and Hawley provide the basis for the 1150°C safety Limit for high Zr:H ratio fuels. This stems from the phase change occurring at ~2000°C.[5], [6] It is also established that the limiting factor in reactivity insertions is the cladding itself.[5] In order to limit this cladding temperature, which is difficult to measure, the fuel temperature, which is already being measured, is limited to 940°C in Hawley's paper[5].

Argonne National Laboratory performed pulsing analysis for GA and found a number of temperatures relative to added energy levels for various fuels.[7] This newer report, from 2008, also recommends a temperature limit of 830°C for 20% LEU fuel. This is based on the pressure of Hydrogen in the fuel matrix above 874°C causing microscopic expansion of cracks in the fuel.[7]

Argonne also performed a thermal-hydraulic analysis of the TRIGA coolant channel.[8] Here another proprietary code, STAT, was used to analyze a limiting channel. The nature of the TRIGA fuel geometry and core configuration shows independence of a channel on the neighboring channels and a need only to analyze the limiting channel. Additionally, a number of codes and correlations are compared for

critical heat flux calculations that are not considered directly here, but through a site-specific peaking factor from UT SCALE.

UT TRIGA

Previous to the creation of the reactivity event modelling, a loss of coolant accident model (LOCA) was created for the limiting site-specific core.[9], [10] This model incorporated a one-dimensional, radial, finite element analysis. A sub-model involved a one-dimensional fluid flow analysis of the fuel element coolant space. From this, a number of items were gained including: a thermal hydraulic geometric model, code configuration, buoyancy driven flow analysis, solution control for large system linear ODE solvers. Aspects from this model were taken, and improved upon in the thermal-hydraulic portion discussed below.

The Model

The basis of the model is to take zero-dimensional point neutronics combined with a simplified form of a two-dimensional energy balance and solve for limiting values using site specific configurations in a *state-determined system* model[11]. This will be referred to subsequently as the $\{\mathbb{R}^0, \mathbb{R}^2\}$ model, implying the vector spaces of the neutronics and thermal-hydraulics respectively. The theoretical basis is split between neutronics and thermal-hydraulics. The site-specific criteria define components necessary for accurate verification and validation (VnV), while the programming section outlines the code structure and ease of use.

List of Symbols

Table 1. List of Symbols

$n(t)$	*Total Neutron Density (n^0/m^3)*	$\mathbb{R}^n$	*Real Number Vector Space of Dimension, n*
$C_i(t)$	*Delayed Neutron Precursor Density of the i^{th} Group (n^0/m^3)*	$S(t)$	*Neutronics Independent Sources (n/s)*
λ_i	*Decay Constant for the i^{th} Group (s^{-1})*	Λ	*Neutron Generation Time (s)*
$\rho(t)$	*Event Reactivity* $(\frac{\Delta k}{k})$	β_i^{mix}	*Delayed Neutron Fraction of the Fuel Mixture for the i^{th} Group* $(\frac{\Delta k}{k})$
$\alpha_{F,M}$	*Temperature Feedback for the Fuel and Moderator* $(\frac{\Delta k/k}{C})$	υ	*Neutron Velocity (m/s)*
$T_{F,M}(t)$	*Time Dependent Temperature of Fuel or Moderator (C)*	$\phi(t)$	*Core Flux (n^0/m^2s)*
$T_{F,M,0}(t)$	*Initial Temperature of Fuel of Moderator (C)*	k	*Boltzmann Constant (J/K)*
β_{eff}^{mix}	*Total Core Effective Delayed Neutron Fraction for the Current Fuel Mixture* $(\frac{\Delta k}{k})$	m_n	*Mass of a Neutron (kg)*

K_{Beta}	*Delayed Neutron Fraction Proportionality Constant*	$\sigma_{f,a,s}$	*Microscopic cross section for fission, absorption, and scattering. (m^2)*
$K_{sp,j}$	*Spontaneous Fission Factor (n^0/s)*	N_j	*Number Density of the j^{th} Item ($atoms/m^3$)*
V_{core}	*Volume of the Core (m^3)*	m_j	*Mass of Material (kg)*
$s(t)$	*Installed Neutron Source Neutron Density (n^0/m^3s)*	$S(t)$	*Total Source Density (n^0/m^3s)*
$\rho_{State}(t)$	*Current Reactivity Including Feedback* ($\frac{\Delta k}{k}$)	$\bar{s}(t)$	*Calibration Sheet Neutron Production (n^0/s)*
$k_{eff}(t)$	*Effective Multiplication Factor*	$\rho_{M,F}(t)$	*Feedback Reactivity* ($\frac{\Delta k}{k}$)
l	*Mean Neutron Lifetime (s)*	$\Lambda(t)$	*Neutron Generation Time (s)*
H_{ij}	*Delayed Power of Group j of Isotope i (W)*	E_{ij}	*Delayed Power Fraction (MeV/fission-s)*
P_i	*Total Power of Isotope i (W)*	$\tilde{\lambda}_{ij}$	*Decay Constant of Group j of Isotope i (s)*
Q	*Energy per Fission (MeV)*	$P_{eff}(t)$	*Effective Power (W)*
$P_{inst}(t)$	*Instantaneous Power (W)*	$P_d^i(t)$	*Delayed Power (W)*
E_f	*Recoverable Energy from Fission (W)*	$\rho_{trans}(t)$	*Event Reactivity as a Result of State* ($\frac{\Delta k}{k}$)
RR	*Reactivity Addition Rate* ($\frac{\Delta k/_k}{s}$)	CRW	*Maximum Event Reactivity* ($\frac{\Delta k}{k}$)
$\dot{m}$	*Mass Flow Rate (kg/s)*	$\tilde{\rho}_i$	*Density (kg/m^3)*
A_i	*Flow Area (m^2)*	w_i	*Fluid Velocity (m/s)*
R_i	*Radius of i (m)*	$\check{A} - \check{D}$	*Density Correlation Constants*
dR_i	*Width of Pin Radial Element (m)*	dz_i	*Conductive Path Distance of i (m)*
h_i	*Height of Component i (m)*	dz_cond_i	*Conductive Path Radially of i (m)*
c_p	*Specific Heat Capacity (J/ kg K)*	$\tilde{\alpha}$	*Thermal Diffusivity (m^2/s)*
k_i	*Thermal Conductivity of i (W/ m K)*	g, g'	*Gravity and Density Compensated Reduced Gravity (m/s^2)*
$\dot{E}_{st}, \dot{E}_{in}, \dot{E}_{out}, \dot{E}_{gen}$	*Change in Energy: Stored, In, Out, Generated (W)*	q_{cond}	*Heat Transfer Rate (W)*
$\hat{h}_k$	*Specific Enthalpy of k (J/kg)*	z_k	*Height of k (m)*
$U_{wt\%}$	*Weight Percent of Uranium (kg/kg)*	$\hat{h}_0$	*Reference Enthalpy (j/kg)*

$\tilde{\beta}$	*Thermal Expansion Coefficient (1/K)*	μ	*Dynamic Viscosity (N s/m²)*
D	*Diameter (m)*	ν	*Kinematic Viscosity (m²/s)*
F_{Drag}	*Drag Force (N)*	F_g	*Gravity Resistive Force (N)*
C_D	*Coefficient of Drag (unit less)*	Δp	*Pressure Loss from Friction (Pa)*
f_D	*Darcy Friction Factor (unit less)*	D_H	*Hydraulic Diameter (m)*
ϵ	*Surface Roughness (m)*	P_{wet}	*Wetted Parameter (m)*
h_{cone}	*Cone Height (m)*	η_{fin}	*Fin Efficiency (unit less)*
I_i	*Modified Bessel Function of the i^{th} Kind*	L_{fin}	*Fin Height (m)*

Theory

Neutronics

The neutronics portion uses a zero-dimensional, single neutron velocity solution to the in-hour equation set presented by Johnson, et al. of Idaho National Laboratory.[1] The coupled ordinary linear ODE set presented by Johnson presents itself as a usable representation of Akcasu's point kinetic solution to the integrodifferential forms.[2] Namely, it provides a form free of a shape function and energy distribution, and the layout allows ease of transition to numerical solutions. The generic terms in this set are then made TRIGA specific using GA papers and UT specific using local parameters from UT SCALE, UT TRACE analyses and UT SAR.[12][13][9]

In-Hour Equations

The In-hour equation lies at the heart of the $\{\mathbb{R}^0, \mathbb{R}^2\}$ model. The choice of point kinetics reduces the order of the geometric vector space. The use of the single neutron velocity removes the velocity vector space and need to track lethargy.[2] The overall breakdown can be found below.

Neutron Density and Six DNP Group

The model is designed to simulate transients occurring over a very short time frame. These transients involve reactivity addition from rod withdraw only. Given the time frame, items such as burnup and fission product poison buildup are not considered. Additionally, the time dependence of β from change in isotope concentration is ignored. Values are dependent only on current UT core conditions. Thus, the equations of state for neutron population can be based on reactivity balance, feedback effects, sources, and DNP population only.[1] While Johnson and Akcasu both provide methodologies to track fuel burnup and poisons, this is considered beyond the scope for the model's initial phase. The $\{\mathbb{R}^0, \mathbb{R}^2\}$ model's basis equation is[1]:

$$\frac{dn(t)}{dt} = \frac{\rho(t) + \alpha_F\left(T_F(t) - T_{F,0}\right) + \alpha_M\left(T_M(t) - T_{M,0}\right) - \beta_{eff}^{mix}}{\Lambda(\mathrm{t})} n(t) + \sum_{i=1}^{6} \lambda_i C_i(t) + S(t) \qquad 1$$

$$\frac{dC_i(t)}{dt} = \frac{\beta_i^{mix}}{\Lambda(\mathrm{t})} n(t) - \lambda_i C_i(t) \ , \qquad i = 1 \ldots 6 \qquad 2$$

The governing equation for neutron population is relative to the prompt effects, delayed precursor concentration and external sources. While the use of a single velocity model would allow for a flux term instead of a neutron density, it would make the solution technique more difficult for future work. A single velocity model was chosen because it simplifies calculations involving flux by removing the need to track velocity and diffusion vector spaces in $\mathbb{R}^3$ or lethargy vector space in $\mathbb{R}^2$.[2] Any terms involving flux have the velocities accounted for with:

$$\phi(t) = \upsilon n(t) \tag{3}$$

$$\upsilon = \sqrt{\frac{2kT_F\{K\}}{m_n}} \tag{4}$$

Here, the fuel temperature, in Kelvin, is used to determine the velocity of the neutrons using the Maxwellian distribution assumption and equation for neutron mean temperature. While the neutron temperature is better represented by the bound hydrogen energy levels, due to their Einstein oscillator tendency[3], this was thought to impinge too much on the kinetic effects covered under the prompt temperature coefficient of the fuel. The dynamics are tracked using neutron density, the velocity term only comes into play in the density initial condition and the output calculation of peak power, so this assumption is considered appropriate.

The DNP concentrations are accounted for in (2) while the external sources, discussed below, include the core's external AmBe source and spontaneous fission effects of the fuel materials. The prompt neutron reactivity balance includes the event reactivity, feedback effects and the effective delayed neutron fraction.

The Delayed Neutron Fraction (β) and Effective Delayed Neutron Fraction (β_{eff})

The mixture delayed neutron fraction of a core is dependent upon the fuel materials used and their relative concentrations. In this model, the effects of ^{235}U, ^{238}U, and ^{239}Pu are all accounted for. Their concentrations are determined from SCALE. Here, the energy levels and decay constants of each fuel's 6 groups is combined using a weighted average.[1], [2], [14] The model accounts for the ith group's mixture delayed neutron fractions and decay constants using:

$$\beta_i^{mix} = \frac{\sum_{j=1}^{3} \upsilon_j \sigma_f^j N_j(t) \beta_j^i}{\sum_{j=1}^{3} \upsilon_j \sigma_f^j N_j(t)} \tag{5}$$

$$\lambda_i^{mix} = \frac{\sum_{j=1}^{3} \upsilon_j \sigma_f^j N_j(t) \lambda_j^i}{\sum_{j=1}^{3} \upsilon_j \sigma_f^j N_j(t)} \tag{6}$$

The effective delayed neutron fraction is dependent on a number of factors including birth energy distributions, core buckling, diffusion constants and neutron age.[2], [15] These values are not readily available for most cores as they are geometrically specific and thus site-specific and difficult to measure and verify. This lead to an alternative method for calculating the effective delayed neutron fraction. According to GA-7882, the TRIGA core with ^{235}U has an effective delayed neutron fraction of *.007.*[12] Taking into account the delayed neutron fraction of ^{235}U, a proportionality constant can be found:

$$K_{Beta} = \frac{\beta_{eff}^{235}}{\beta^{235}} \quad (7)$$

The total delayed neutron fraction for the mixture is given by:[14]

$$\beta^{mix} = \sum_{i=1}^{6} \beta_i^{mix} \quad (8)$$

This yields the mixture effective delayed neutron fraction using:

$$\beta_{eff}^{mix} = K_{Beta}\beta^{mix} \quad (9)$$

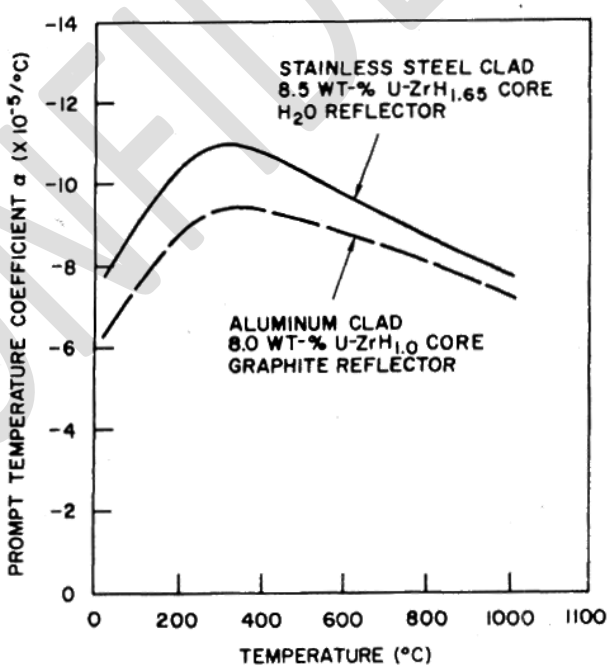
Both Weaver and Akcasu imply an effective delayed neutron fraction gain of 20-30%. This method slightly under estimates that at ~7% gain; however, this is considered an acceptable approximation. The basis for the current equation is: accepted research, and the real value's computational complexity does not fit the premise of the initial model.

Feedback

A majority of the prompt feedback in the $\{\mathbb{R}^0, \mathbb{R}^2\}$ model is accounted for in the fuel and water temperature coefficient terms.

α_F of Fuel

The fuel prompt temperature coefficient is a weighted average of multiple effects including: Einstein oscillator effect, Doppler broadening, Cell effect, and leakage.[3], [12] With a reactivity based point source model, it is desirable to lump these effects into a temperature coefficient term. GA-7882 provides a temperature dependent temperature coefficient for TRIGA fuel.[12]



1. Prompt Temperature Coefficient for TRIGA fuel vs Temperature from GA-7882

From this curve, a temperature dependent polynomial fitted curve was made.

$$\alpha_F(T,t) = 1.4990e^{-16}T_F(t)^4 - 5.3734e^{-13}T_F(t)^3 + 6.5947e^{-10}T_F(t)^2 - 2.8159e^{-7}T_F(t) + 6.9571e^{-5} \quad (10)$$

The value is temperature and time dependent; the equation is time dependent based on the time dependency of the fuel temperature. This time dependence is only notational; in reality the coefficient is based only on fuel temperature alone. Each time iteration, the value is recalculated based on the current state. The reactivity effects of the fuel are accounted for using the current temperature relative to the initial temperature:

$$\rho_F(t) = \alpha_F(T,t)\left[T_F(t) - T_{F,0}\right] \quad (11)$$

α_M of Water

The dominant feedback in a TRIGA reactor is fuel temperature feedback. Local modelling analysis can confirm this.[16] The effect of the water from a neutronics standpoint is only a minor consideration using the locally found temperature coefficient, and using a constant value for it. The reactivity effects are then similar to the fuel, but without state dependency of the coefficient:

$$\rho_M(t) = \alpha_M\left(T_M(t) - T_{M,0}\right) \quad (12)$$

The water tends to have a slightly positive effect.[16]

Xenon and poisons

While poison buildup, especially Xenon, would have an effect on the dynamics, the core is considered cold clean critical initially. With a timescale of seconds and minutes, this effect is considered negligible given the half-lives of Xenon and Iodine being on the order of hours.

Burnup

Burnup would account for a long term feedback effect. However, the intent is to model events on the order of seconds and minutes, so these effects are neglected. Core age does play a role in the initial condition vector and fissile material concentrations, but not the dynamic model.

Neutron Generation Time

According to Weaver,[14] the neutron lifetime is related to the current effective neutron multiplication factor and the mean energy lifetime[14]. The mean neutron lifetime is taken directly from the UT SAR.[13] The effective multiplication factor is found from the current state of event reactivity and feedback.[14]

$$\rho_{State}(t) = \rho(t) + \rho_F(t) + \rho_M(t) \quad (13)$$

$$k_{eff}(t) = \frac{1}{1 - \rho_{State}(t)} \quad (14)$$

$$\Lambda(t) = \frac{l}{k_{eff}(t)} \quad (15)$$

This dependency on the feedback state ensures the generation time is changing relative to current core conditions.[14]

External Sources

The external sources considered are the added AmBe source and the spontaneous fission of ^{235}U, ^{238}U, and ^{239}Pu.

$$S(t) = s(t) + \sum_{j=1}^{3} \frac{K_{sp,j} m_j}{V_{core}} \quad 16$$

The source density is determined from its calibration sheet neutron production rate and the volume of the core:

$$s(t) = \frac{\bar{s}(t)}{V_{core}} \quad 17$$

While source neutrons are not evenly distributed in higher dimensional models, the $\mathbb{R}^0$ assumption negates that geometric effect. While the spontaneous fission neutron production is from the factors found in [1] and the local material properties of mass and volume.

ANSI Decay Heat and Effective Power

Decay Heat

Decay heat in the core comes from radioactive decay of nuclides born at unstable energies.[1] ANS standards provide the information for the decay heat fractions of ^{235}U, ^{238}U, and ^{239}Pu.[17] This allows a state dependent calculation of the heat (power) associated with the radioactive decay of fission products. This heat has a role in the energy generation term of the fuel, but is not normally accounted for in the neutronics equations of state (1), (2). This requires "delayed power 'concentrations'"[1] and tracking their changes with state changes:

$$\frac{dH_{ij}}{dt} = \frac{E_{ij} P_i}{Q} - \tilde{\lambda}_{ij} H_{ij} \ , \qquad i = 1 \dots 3 \; j = 1 \dots 23 \quad 18$$

This leads to a term tracking the power of each nuclide fraction of each of the isotopes in the core. This lends to calculating the 'effective power' of the core.[1], [17]

Effective Power, Instantaneous Power, and Delayed Power

Three different forms of energy need to be tracked for each cycle. Instantaneous power is the recoverable energy from fission at the current state, mostly due to kinetic energy of the fission products.[1] Delayed power is the fraction of energy produced from each isotope that is delayed from the current state. Effective power takes into account the instantaneous power, the state's current power from the delayed power concentrations and compensating for this state's power produced delayed:

$$P_{eff}(t) = P_{inst}(t) - \sum_{i=1}^{3} P_d^i(t) + \sum_{i=1}^{3} \sum_{j=1}^{23} H_{ij}(t) \quad 19$$

$$P_{inst}(t) = \sum_{i=1}^{3} \phi(t) E_f N_i \sigma_f^i V_{core} \quad 20$$

$$P_d^i(t) = \frac{{P_{inst}}^i(t)}{Q} \sum_{j=1}^{23} \frac{E_{ij}}{\tilde{\lambda}_{ij}} \quad 21$$

The effective power plays an important role in the thermal hydraulic model. The instantaneous power equation is used during the output phase to determine the peak power from the stored neutron density.

Input Reactivity

The input reactivity term is relative to the chosen model.

$$\rho(t) = \rho(t_0) + \rho_{trans}(t) \quad 22$$

$$\frac{d\rho_{trans}(t)}{dt} = 0 \; \{Pulse, Rod\ Withdraw\ \rho_{trans}(t) \geq CRW\}; \quad 23$$

$$\frac{d\rho_{trans}(t)}{dt} = RR \; \{Rod\ Withdraw, \rho_{trans}(t) < CRW\} \quad 24$$

The reactivity addition rate is considered constant with the current version. Realistically, this value would vary along the differential rod worth curve. Additionally, rod withdraw events are limited to the maximum allowed reactivity, as chosen by the user. A pulse is considered an instantaneous addition of reactivity, and thus the event reactivity is constant. Thus, pulses use a step insertion and rod withdraws use ramp insertion, whose slope is based on the reactivity addition rate.

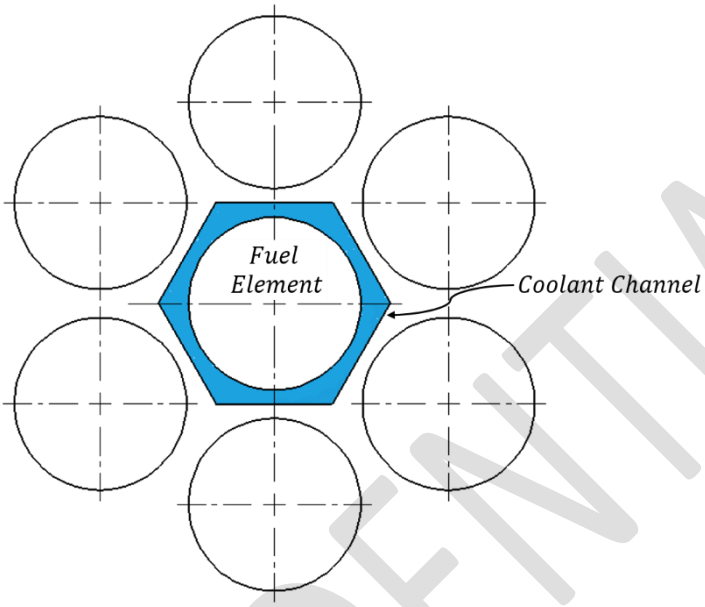
Time Dependence and State Considerations

The model lends itself to time dependence on items such as source strength, poison effects, and delayed neutron fraction. These items are driven by fissile material concentrations and are neglected. Tracking of the neutron density stems from the point kinetics model.[1], [2] The decision to use a single velocity neutron spectrum is a result of initial model phasing and computational access.

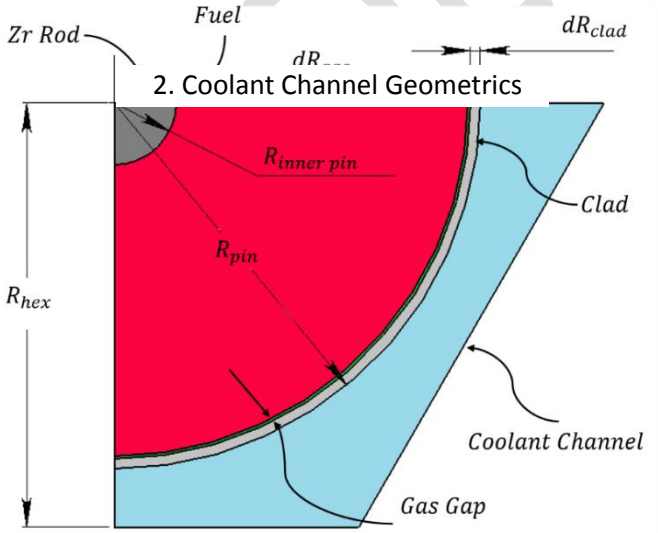
Thermal Hydraulics

The $\{\mathbb{R}^0, \mathbb{R}^2\}$ model uses simplified two-dimensional thermal hydraulic interactions. The generic regions of fuel, graphite, gas, cladding and coolant channels are discretized. The state dependent mass, momentum, and energy balances are solved to find the temperatures of these regions. The coolant and fuel temperatures then feed directly into the neutronics equations.

The geometry is based on the UT TRIGA hexagonally arranged core with simplified entry and exit criteria. The fuel pin used accounts for the inner Zirconium pin, fuel meat, gas gap, cladding and coolant channel. The edges of the hexagonal coolant channel bisect the distance between the related pin, and create a single pin representative unit. This is based on the assumption the flow between pins is symmetric.

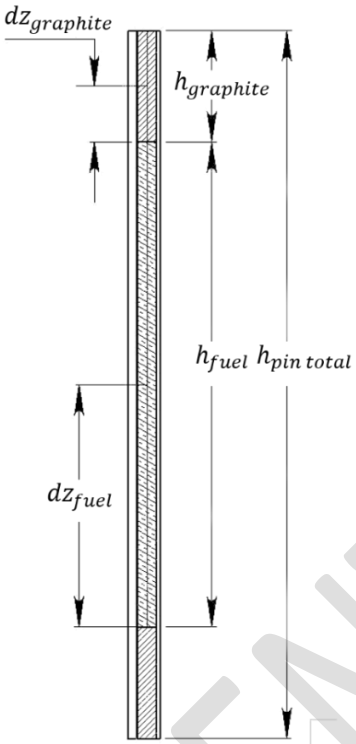


2. Coolant Channel Geometrics



3. Radial Section View of Fuel Element and Coolant Channel

The axial portion of the pin model, shown below, accounts for fuel and graphite regions.



4. Axial Fuel Pin Heights

Mass Balance and Buoyancy Driven Flow

The mass balance assumption allows simplification in the solution technique of the energy balance. Mass flow rate is considered constant:[18]–[22]

$$\dot{m} = \tilde{\rho}_i A_i w_i = \tilde{\rho}_{i+1} A_{i+1} w_{i+1} \quad (25)$$

This allows the known or limiting region mass flow rate to be found and used for the entire channel. This simplification is necessary to maintain the linearity of the system of equations and to optimize them to non-super computer solution methods.

Geometry

The mass flow rate area is based on the surface normal to the surface of the fuel pin. The core configuration is hexagonal, leaving a hexagonal shaped coolant channel with a circular portion occupied by the pin. The area of the coolant hexagon is given by:

$$A_{hex} = \frac{\sqrt{3}}{2}(2R_{hex})^2 - \pi R_{pin}^2 \quad (26)$$

This gives the area for a single pin along the axial region of constant diameter. With the constant mass flow assumption, this is the only required area for coolant fluid flux.

Density

The TRIGA reactor uses buoyancy driven flow causing coolant density to become a state dependent function. A temperature based water density correlation was found to be:[23]

$$\tilde{\rho}(T,t) = \frac{\check{A}}{\check{B}^{\left(1+\left(1-\frac{T_M(t)}{\check{C}}\right)^{\check{D}}\right)}} \quad (27)$$

Its time dependency stems from the need to calculate it each iteration.

Specific Heat Capacity

Specific Heat capacity is also a temperature dependent function, making is a state dependent function. A curve fit was created from a table of specific heat values:[24]

$$c_p(T,t) = 3.16744e^{-10}{T_M}^4 - 1.05772e^{-7}{T_M}^3 + 2.3533e^{-5}{T_M}^2 - 1.4767e^{-3}T_M + 4.20617 \quad (28)$$

This function return needs to be multiplied by 1000 to give the proper units of (J/kg K).

Thermal Diffusivity

Thermal diffusivity relates the ability of a material to conduct thermal energy versus store thermal energy. It contains the state dependent variables of density, specific heat capacity, and thermal conductivity. This makes it state dependent.

$$\tilde{\alpha}(T,t) = \frac{k(T,t)}{\tilde{\rho}(T,t)c_p(T,t)} \quad (29)$$

The temperature dependence of the thermal conductivity is found with the state dependent equation:

$$k(T,t) = -1.488445 + 4.12292\frac{T_w}{298.15} - 1.63866\left(\frac{T_w}{298.15}\right)^2 \quad (30)$$

Mass Flow Rate

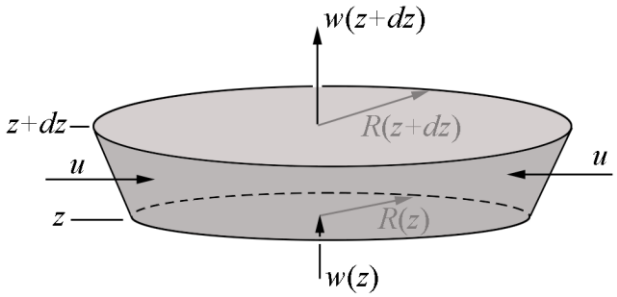
Buoyancy driven flow adds a number of complexities to a finite element, constant volume model greater than $\mathbb{R}^0$ as the equations are coupled and elliptic partial differential equations.[19], [25], [26] The flow itself is driven by density changes, which in this case, occur from heat transfer from the fuel pin. This density change is found by comparing element inlet and outlet temperatures. The temperature rise is associated with the total energy transferred into the volume from the fuel element. To find the total energy transferred, either a coolant velocity or volume residence time needs to be known. Both of which are related to the density change itself.

Channel Characteristics

In an effort to simplify these equations and find a velocity relationship, a number of assumptions are made. The first of which relates to the coolant channel characteristics. It is assumed that the space between the pins results in a symmetrical flow of constant radial velocity, allowing only half the gap between pins to be considered. This leads to the hexagonal channel shape. Additionally, the velocity in the radial direction, $\vec{u}_{Radial}$, is considered to be 0. Also, the flow is considered to be fully developed with a uniform radial velocity profile.

Plume Model

The next simplification was to find a model outlining a buoyant driven velocity relationship that is solvable. The velocity profile of buoyant plumes is a topic of study in the geological field.[27], [28]



5. Momentum Balance in a Thermal Plume[27]

Above shows the momentum balance of a thermal plume. It takes into account height, axial and radial velocities and momentum flux. Derivation of the momentum equations leads to the velocity differential:

$$\frac{d}{dz}(R^2w^2) = R^2g' \tag{31}$$

However, in the case of the pin coolant channel with the $\vec{u}_{Radial} = 0$ assumption, the radius is constant not dependent on the height. This leads to:

$$dw = \sqrt{g'dz} \tag{32}$$

The reduced gravity term in (31) is a state dependent variable defined by:[29]

$$g'(t) = g\frac{\rho_i(t) - \rho_{i+1}(t)}{\rho_i(t)} \tag{33}$$

Boussinesq Assumption

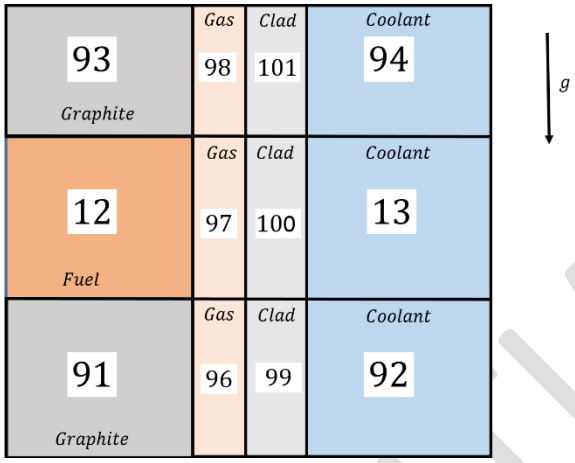
The next assumption is the Boussinesq assumption, used to find a relationship for outlet density. Here the density differences are neglected, except those related to gravity, [19] and changes in density are due to temperature alone.[25] This makes the equation for outlet density:

$$\tilde{\rho}_{i+1}(T,t) = \tilde{\rho}_i(T,t)[1 - \tilde{\alpha}(T,t)(T_{i+1} - T_i)] \tag{34}$$

Substituting (31) and (30) into (29) gives a differential velocity term related to a temperature difference:

$$\frac{dw}{dt} \cong \sqrt{\tilde{\alpha}(T,t)[T_{i+1} - T_i]dz} \tag{35}$$

Using the constant mass flow rate assumption, solving for one mass flow in a known region leads to the required mass flow in the energy balance for all axial, convective regions. The coolant channel is segmented axially into three regions, correlated with the fuel and graphite regions on the pin.



6. Region Distribution in Thermal Hydraulics[9], [62]

Velocity is calculated using the densities of regions 13 and 94. The area is taken from the boundary between these two regions and the density used is from region 13. The assumption is that region 94 will tend to be less dense than region 13. This leads to the mass flow rate equation:

$$\dot{m}(t) = \rho_{13} A_{13-94} w_{13-94}(t) \tag{36}$$

The final assumption revolves around the solution technique. Fully transient conditions would tend to satisfy a momentum balance, but not necessarily a mass balance. The computational instability of the momentum balance equation led to using a pseudo steady state solution at each iteration. This is the same method used in RELAP5 and is considered acceptable.[8]

Momentum Balance Expansion of Thermal Hydraulics

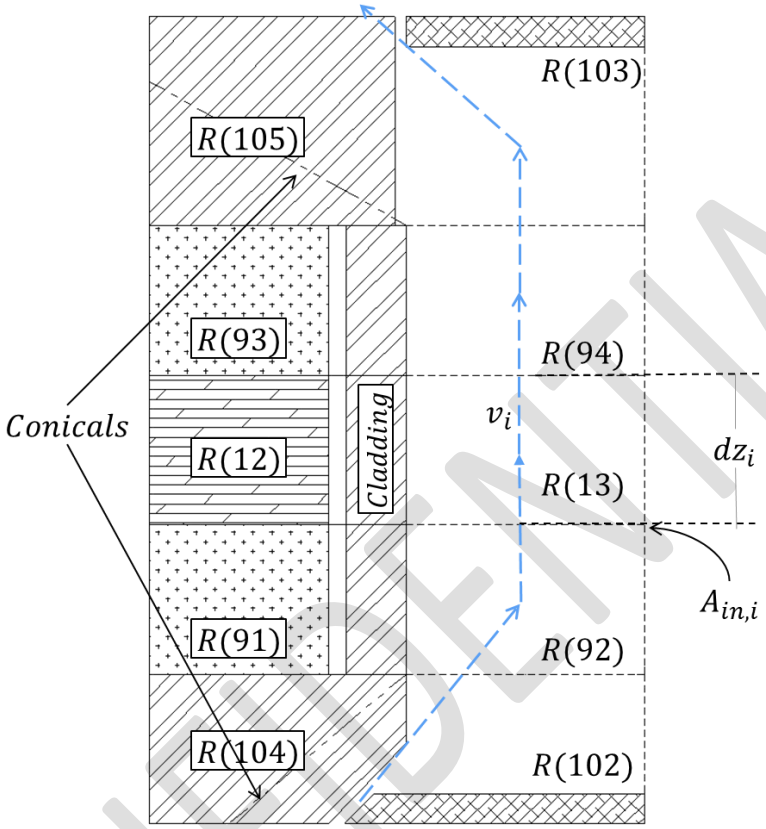
The momentum balance expansion takes the simplified geometries and assumptions of the mass balance and develops them further. The regions are expanded to incorporate the effects of the fuel pin components above and below the graphite; a one-dimensional flow vector, vice a constant value, represents fluid velocity. On the fuel element ends, the fins are taken into consideration both in flow development and heat transfer, and the effects of drag are accounted for. Conservation of momentum is obtained using the following equation:

$$\rho_{out} A_{out} w_{out}^2 - \rho_{in} A_{in} w_{in}^2 = F_{Drag} + F_g \tag{37}$$

Where resistive forces of gravity and drag offset the change in inlet and outlet conditions.[20], [21] Much like the mass balance, a pseudo steady state condition is assumed, to allow removal of the transient term. This equation represents a better characterization of the flow, but contributes to the extreme stiffness of the ODE set.

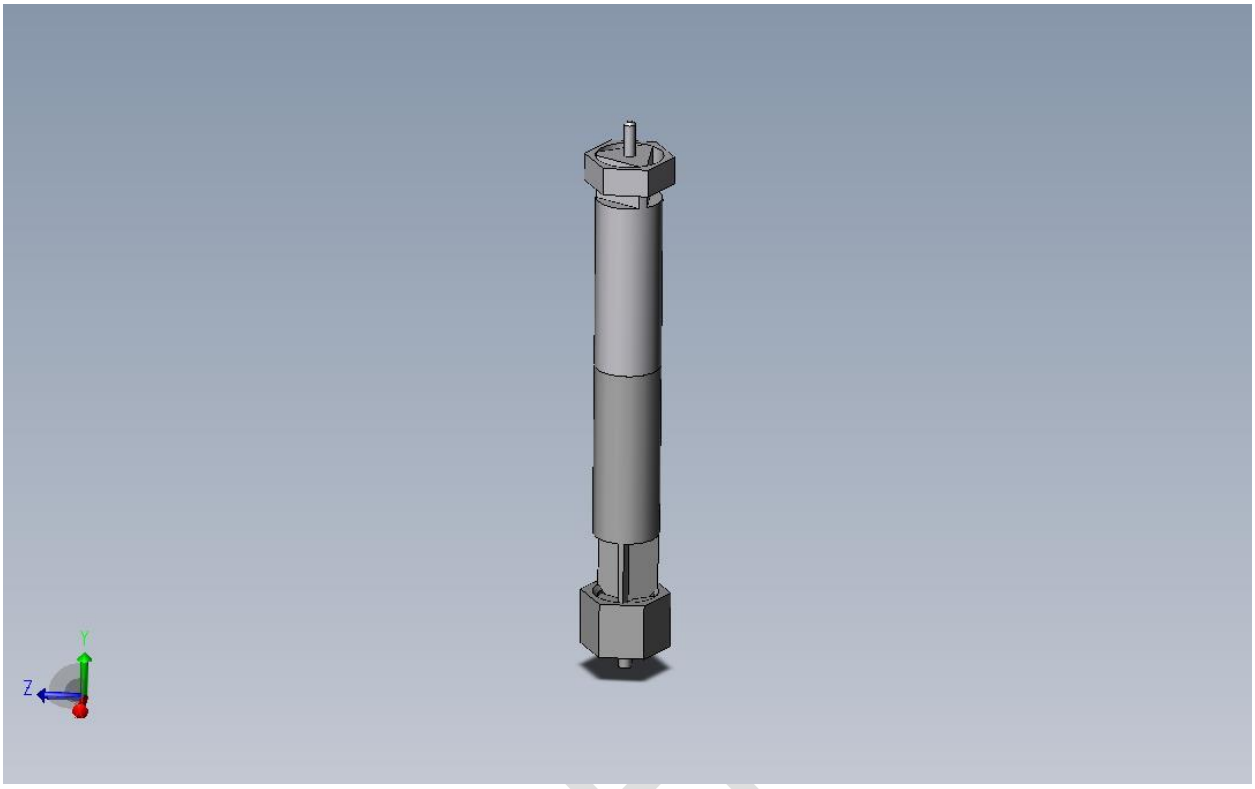
Regional Expansion

The regions being tracked are expanded to compensate for the new equations and relationships. The fuel element had two added regions of all stainless steel, while the coolant channel has two additional volumes added.



Each region has an inlet and outlet areas, as well as relative differential height for the drag calculations. There are five distinct velocities to be considered as well as five different density regions.

The areas of regions 92, 13, and 94 are constant along the constant radius cylinder. The inlet and outlet area are calculated from solid models based on fuel element drawings.



7. Fuel Pin Solid Model from 10000 Series Used for Inlet and Outlet Flow Area Characteristics (Note: this top represents the simpler 2000 Series for Initial Geometric Design. The Model Uses the Geometry in Figure 10 for Actual Values.)

Using the solid part and creating a part from the void, the area and parameter values are measured and used in the iterative mass flow calculator.

Flow Development

With the expansion into the momentum balance, the flow is considered differently than the mass balance. In the mass balance, the plume model provides a differential velocity between the fuel area coolant and the upper graphite coolant. Flow, and thus mass flow, is then considered constant. In the momentum balance, the flow between these two regions, 13 and 94, is developed in the same way; however, each region's specific velocity and mass flow rate are calculated using a sub-function. This provides better representation in the energy balance.

The major defining factor is the drag force, defined by:

$$F_{Drag} = \frac{1}{2}\rho w^2 C_D A_S \qquad 38$$

The drag force is divided into the effects of the cylinder walls and the cone features. The fins are considered triangular extended surfaces, whose heat transfer effects are considered, but not drag features.[30]–[39] They were modeled in Solidworks and used to find surface area for heat transfer.[32] Models were made for both the top and bottom fin assemblies.

8. Lower Fuel Element Fin Geometry

Cylindrical Drag Force

The drag force along the cylindrical portions of the fuel pin is calculated from the Darcy friction equation. The effects of flow axial along a rough cylinder are not well categorized directly from a drag coefficient. Most analyses revolve around cross-flow.[18], [20], [21], [40], [41] However, the drag coefficient can also be found from a pressure loss using:[40]

$$C_D = \frac{\Delta p}{\frac{1}{2}\rho w^2} \tag{39}$$

Where the differential pressure can be found with:[40]

$$\Delta p = \frac{1}{2} f_D \frac{dz}{D_H} \rho w^2 \tag{40}$$

And the friction factor is found using the Swamee–Jain equation with:

$$f_D = \left[-2 \log \left(\frac{\epsilon}{3.7 D_H} + \frac{5.74}{\left(\frac{w\rho}{\nu}\right)^{.9}} \right) \right]^{-2} \tag{41}$$

For the state dependence, the friction factor's Reynolds number is found using the coolant regions temperature dependent density, viscosity and velocity. The roughness for stainless steel is based on values from White, while the hydraulic diameters are found with 42.[40]

The Darcy drag force is found for each region's cylindrical sections. The conical portions are considered to have a higher geometric effect and the change in pressure from the walls can be neglected.

Characterization of Fuel Element Extremities

The fuel element ends are characterized by considering the effects of: the heat transfer from the fin, the drag on the conical sections and cylindrical sections, and the heat transfer from the region (considered one unit for mass and temperature) to the coolant.

Each loop the ODE solver recalculates the velocity distribution using a sub-function. First, the area and parameter values from the solid models are used to find the hydraulic diameters:

$$D_H = \frac{4A_{flow}}{P_{wet}} \tag{42}$$

This plays a role in both the momentum balance and the Darcy friction factor.[20], [31]

Cone Section

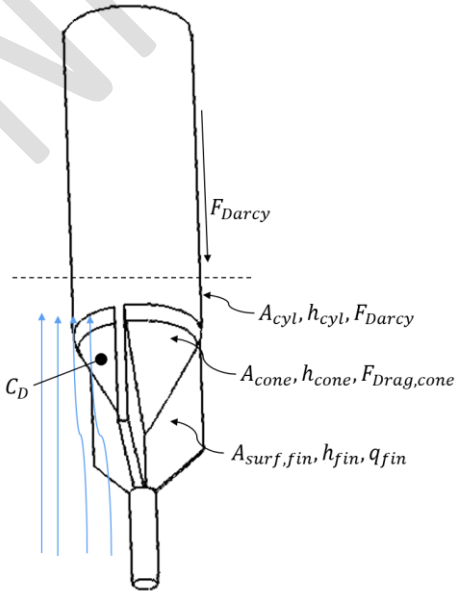
The conical section plays two roles: it has a drag factor for the resistive force and provides a surface area for heat transfer to the coolant.

The surface area of a cone is found with:

$$A_{s,cone} = \pi r_{pin}\left(h_{cone}^2 + r_{pin}^2\right) \tag{43}$$

Since the lateral portion is the only portion exposed to the water, the flat portion is ignored. Additionally, for simplicity, the portions occupied by the thin fins are not neglected.

The coefficient of drag for a cone was found using rocketry correlations, and White[40] specifically: $C_{D,cone} = .8$ [42]–[44] Where the drag coefficient is based on the angle of attack of the cone.



9. Lower Fuel Element Geometry

The heat transfer coefficient from the cone is found using the Rayleigh number from the fin calculations and finding the Nusselt number using:[35], [45]

$$Nu_{cone} = 2.0963 + .669 Gr_{fin} Pr_i \quad (44)$$

The heat transfer from the cone uses Newton's law of cooling 61.

Fin Sections

The fin sections are considered triangular extended surfaces. Their heat transfer effect on the region of the cone and cylinder are considered; and they are used to develop that region's flow properties, such as the Rayleigh number. Their energy contribution to the region is found with:[30], [36], [39], [46]

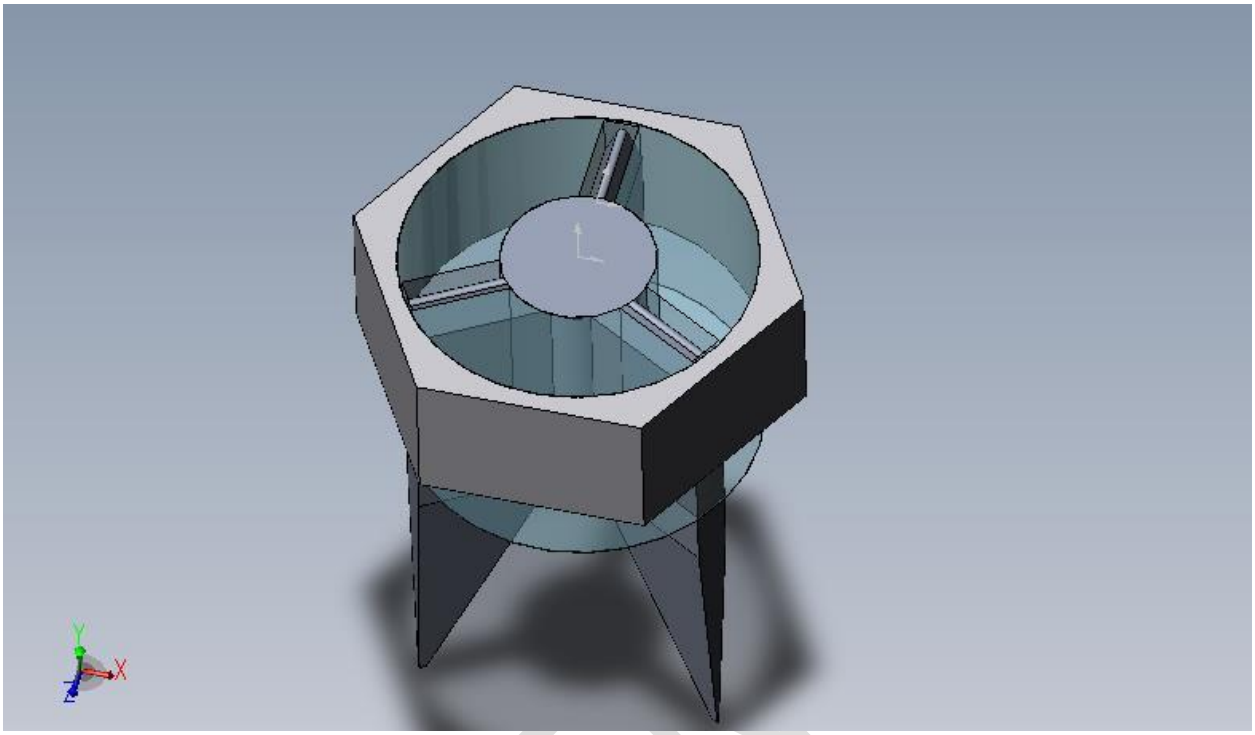
$$q_{fin} = \eta_{fin} h_{fin} A_{S,fin} (T_{SS,i} - T_{coolant,i}) \quad (45)$$

To find the heat transfer coefficient first Rayleigh number must be found from the Grashof and Prandtl numbers. The Prandtl number is found with 63 and the Grashof number is found with 62 where, D in this case is replaced with the height of the fin in the radial direction. 45 contains an efficiency factor whose derivation is below.

Geometry

The fin geometry was developed using solid modelling software in Solidworks. A replica fuel element of the 10,000 series type and a micrometer was used to develop the upper and lower fins. This allowed for a surface area and parameter values using grid plate interfaces built from the UT TRIGA original drawings.

The lower fuel element extension below the fins sits below the lower grid plate and is neglected. The upper fin geometry was simplified to be solely an array triangular section and it disregards the radially flat portions of the fins.



10. Fuel Element Upper Portion for Flow Analysis and Fin Properties

The portion used to grasp the fuel with the tool is also ignored, as its effects are considered negligible.

Efficiency

The efficiency of the fin plays a key role in the effect it has on the transient temperature of the extended fuel element regions. It is found based on the flow conditions in the surrounding area.[21], [31]

While the efficiency factor varies, the computational demands were excessive enough to validate a static analysis and calculation of an effective constant value. A temperature of 30°C was used for the coolant and a value of 450°C was used for the fuel temperature. These represent median values for the respective regions based off of the mass balance analysis.

The Rayleigh number was calculated with both a 1°C and 800°C ΔT. It was seen that the flow remains near laminar Rayleigh numbers up to ~650°C ΔT so the laminar flow correlation was used:

$$Nu_{fin} = .59 * Ra_{fin}^{.25} \quad 46$$

Where the Rayleigh number is found using equation 64, but with the Grashof and Prandtl numbers for water at 30°C.

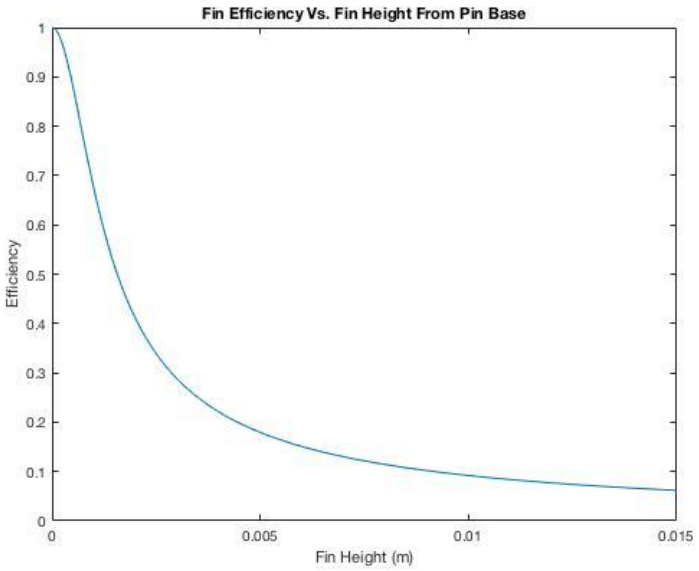
The fin efficiency was found using the equation from Bergman that ratios modified Bessel functions of the first and zero order:[21]

$$\eta_{fin} = \frac{1}{mL_{fin}} \frac{I_1(2mL_{fin})}{I_0(2mL_{fin})} \quad 47$$

Here, *m*, is a factor found using:

$$m = \sqrt{\frac{2h_{fin}}{k_{SS}t}} \tag{48}$$

The heat transfer coefficient is found using 46, but the length L representing the fin extension from base surface varies over the length of the fin. An array in MATLAB was created with 1000 points from the minimum to maximum heights as measured from the Solidworks models. This gave an output efficiency curve:



It is evident from the shape of the curve that for a majority of the heights the efficiency is between 10% and 20%. Since the fin shape tends toward mostly heights greater than .0075m, the median efficiencies were used for the upper and lower fins. 45 can now be used to find the energy lost from the stainless region by the extended surface.

Flow calculations

With the development of velocity based on the 13-94, the momentum equation is solved to find the unknown velocity for each region.

$$w_{in} = \sqrt{\frac{\rho_{out}A_{out}w_{out}^2 - F_{Drag,cyl} - F_{Drag,cone} - F_g}{\rho_{in}A_{in}}} \tag{49}$$

Which leads to that surface's mass flow rate using equation 36. The sub-function outputs each of the control volume's boundary mass flow in to the energy equation using (36).

Energy Balance

The energy balance is a fundamental aspect of the $\{\mathbb{R}^0, \mathbb{R}^2\}$ model. The basis lies in a constant volume, finite element analysis (FEA) method using nano-scale time steps. The UT LOCA showed using nanosecond time steps leads to acceptable Fourier numbers.[9] While the mass flow rates involved are the pseudo steady-state values found from (36), the temperatures are considered purely transient. The effects of radiation are neglected due to the water filled channel.[20], [21] The energy balance is the transient version of the first law of thermodynamics:

$$\dot{E}_{st} = \dot{E}_{in} - \dot{E}_{out} + \dot{E}_{gen} \rightarrow \quad (50)$$

$$\rho \cancel{V} c_p \frac{dT}{dt} = q_{cond} + q_{conv} + q_{gen} + \dot{m} \sum_k (\hat{h}_k + g z_k) \rightarrow \quad (51)$$

$$\frac{dT}{dt} = \frac{1}{m(t) c_p(t)} \left[q_{cond} + q_{conv} + q_{gen} + \dot{m} \sum_k (\hat{h}_k + g z_k) \right] \quad (52)$$

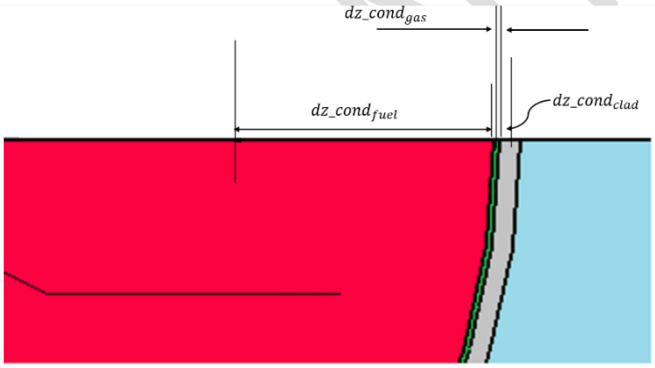
In the above balance, considerations are made for state dependent mass (based on temperature dependent density), specific heat and energy transfer. The water is considered to be entirely in the liquid phase at near STP with a small temperature change across the liquid region, allowing changes in specific energy to be approximated by changes in specific enthalpy.

Conductive Pathways

Geometry

The conduction equations used are standard mixed material conduction for square geometry. The differential radii of the gas and cladding regions are considered small enough to allow this.

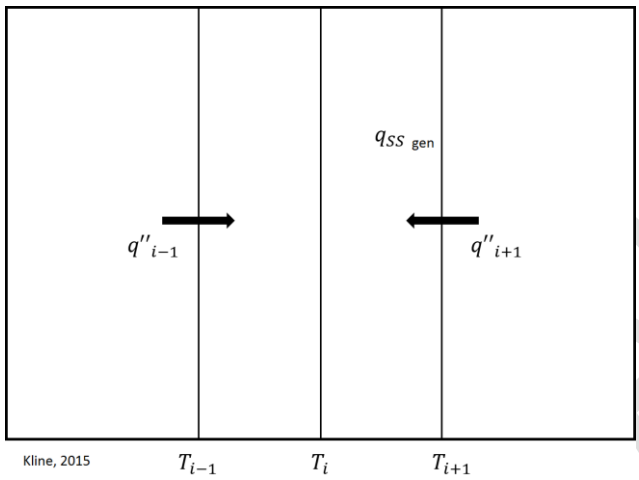
Thus the conduction path between material A and B is defined as:



11. Radial Conduction Terms

$$q_{cond,A}(t) = \frac{(T_B - T_A) A_{AB}}{\frac{dz_A}{k_A} + \frac{dz_B}{k_B}} \quad (53)$$

Here the differential heights are the distance from the geometric center to the boundary along the dimension of conduction. In FEA, all the conductive energies are considered inward so changes in temperatures of the regions will naturally work out the proper energy transfer.[9], [21]



12. Conductive Layout for Finite Element Analysis

Heat Capacity

The specific heat capacity of the fuel element materials is considered to have a transient temperature dependence as well. The graphite specific heat is taken from a material properties chart and curve fit:[24]

$$c_{p,graph}(T,t) = .10795e^{8}{T_G}^{-3} - .61257e^{5}{T_G}^{-2} + .30795e^{-4}T_G + .44391 \quad (54)$$

This value has to be adjusted for (J/ kg K) by multiplying by 4183.995. The heat capacity of the fuel, like all values used for TRIGA fuel is taken from Simnad.[6]

$$c_{p,vol}\left\{\frac{J}{m^3K}\right\} = 2.04 + 4.17e-3 \cdot T_F(t) \quad (55)$$

This gives the volumetric heat capacity but, the specific mass based version is required. Simnad again provides the equation for density:[6]

$$\rho_{Fuel} = \frac{1}{\left(\frac{U_{wt\%}}{\rho_U}\right) + \frac{(1-U_{wt\%})}{\rho_{Zr}}} \quad (56)$$

This allows conversion to mass based using:

$$c_{p,fuel}\left\{\frac{J}{kg*K}\right\} = \frac{\rho_{Fuel}}{c_{p,vol}} \quad (57)$$

The specific heat capacities of the gas and clad were taken from Pacific Northwest National Laboratory and Makeitfrom materials websites.[47], [48]

Gas

The fundamentals of the gas film layer between the fuel and the cladding are taken mostly from Fenech.[49] The heat transfer coefficient is taken from the Kansas State SAR.[50] Using Fenech's correlation, the thermal conductivity can be found:

$$k_{gas} = \frac{\hat{h}_{gas}}{dR_{gas}} \tag{58}$$

The extreme complexities of the gas gap are simplified by assuming a constant consistency of Hydrogen, and a conductive behavior vice internal circulatory flow.

Work Terms

The work terms arise in the energy balance in the regions containing coolant flow. Constant volume, pseudo steady-state analysis involves both an enthalpy balance and potential energy change.

$$\dot{m} \sum_{k} \left(\hat{h}_k(T,t) + g z_k \right) \tag{59}$$

The enthalpy is dependent on both specific heat capacity and deviation in temperature from a reference temperature. The reference enthalpy is calculated at a reference temperature of 0°C with a value of 9007 (J/kg).

$$\hat{h}_k(T,t) = c_{p,k}(T,t) T_k(t) + \hat{h}_0 \tag{60}$$

The potential energy term is calculated using the differential pin axial height.

Heat Transfer Coefficient

The convection effects of the energy balance are considered using Newton's Law of Cooling[20] and the heat transfer coefficient relationship:

$$q_{conv,k}(T,t) = \tilde{h}_k A_{surf} \left(T_{clad,k} - T_k \right) \tag{61}$$

This requires finding an appropriate heat transfer coefficient. The heat transfer coefficient is considered constant over the length of each axial region.

Rayleigh Number

The Rayleigh number is a product of the Grashof and Prandtl numbers. The Grashof relates buoyancy and viscosity within a fluid:[20], [21]

$$Gr_{L,k} = \frac{g \cdot \tilde{\beta} \cdot \left(T_{clad,k} - T_k \right) \cdot {D_{pin}}^3}{\mu^2} \tag{62}$$

While the Prandtl number relates momentum and thermal diffusivities:[21]

$$Pr = \frac{\nu}{\alpha} \tag{63}$$

This leads to the Rayleigh number:[20]

$$Ra_{L,k} = Gr_{L,k}Pr = \frac{g\tilde{\beta}\left(T_{clad,k} - T_k\right){dz_k}^3}{\alpha\nu}Pr \quad 64$$

This number applies to each vertical coolant section individually, as the temperature of the channel and cladding varies axially.

Nusselt Number

The Nusselt number is the essential correlation in the convective phase. It relates convective and conductive heat transfer and is used to find the heat transfer coefficient. This model uses a vertical cylinder correlation relating the Prandtl and Rayleigh numbers for external free convection.[41]

$$\overline{Nu_{L,k}} = \left\{.825 + \frac{.387Ra_{L,k}^{1/6}}{\left[1 + \left(\frac{.492}{Pr}\right)^{9/16}\right]^{8/27}}\right\}^2 \quad 65$$

This correlation was chosen as it best represents the physical situation.

Heat Transfer Coefficient

The heat transfer coefficient can now be calculated using the Nusselt number correlation:[20]

$$\overline{Nu_{L,k}} = \frac{\tilde{h}_k dz_k}{k_{water}} \rightarrow \tilde{h}_k = \frac{\overline{Nu_{L,k}}k_{water}}{dz_k} \quad 66$$

With the convective relationship found, the region specific energy balances can be built

Fuel Energy Balance

The fuel energy balance is the only region with heat generation. The other terms are conductive, as the cladding is the only convective boundary:

$$\frac{dT_F}{dt} = \frac{1}{m_F(t)c_{p,F}(t)}\left[P_{eff}(t) + \frac{\left(T_{graph,1}(t) - T_F(t)\right)A_{F,G}}{\left(\frac{dz_{graphite}}{k_{graphite}} + \frac{dz_{fuel}}{k_{fuel}}\right)} + \frac{\left(T_{graph,2}(t) - T_F(t)\right)A_{F,G}}{\left(\frac{dz_{graphite}}{k_{graphite}} + \frac{dz_{fuel}}{k_{fuel}}\right)} + \frac{\left(T_{gas,2}(t) - T_F(t)\right)A_{S-F,G}}{\left(\frac{dz\_cond_{gas}}{k_{gas}} + \frac{dz\_cond_{fuel}}{k_{fuel}}\right)}\right] \quad 67$$

The effective power is the value discussed in the neutronics decay heat calculations. The areas represent the surface areas along the conduction vector. The mass is calculated from the Simnad density equation (56). The remaining values are state variables.

Graphite Energy Balance

The graphite sections have similar energy balances. No energy is created internally and the border along the grid plate areas is considered insulated. This leaves conduction between the fuel and gas only:

$$\frac{dT_{graphite}}{dt} = \frac{1}{m_{graph}(t)c_{p,graph}(t)}\left[\frac{\left(T_F(t) - T_{graph,1}(t)\right)A_{F,G}}{\left(\frac{dz_{graphite}}{k_{graphite}} + \frac{dz_{fuel}}{k_{fuel}}\right)} + \frac{\left(T_{gas}(t) - T_{graph,1}(t)\right)A_{S-Graph,G}}{\left(\frac{dz\_cond_{gas}}{k_{gas}} + \frac{dz\_cond_{fuel}}{k_{graphite}}\right)}\right] \quad (68)$$

Gas energy Balance

The gas is considered to interact via conduction and not convection. Its energy balance is:

$$\frac{dT_{gas,2}}{dt} = \frac{1}{m_{gas}(t)c_{p,gas}(t)}\left[\frac{\left(T_{gas,1}(t) - T_{gas,2}(t)\right)A_{G-G}}{\left(\frac{dz_{graphite}}{k_{graphite}} + \frac{dz_{fuel}}{k_{graphite}}\right)} + \frac{\left(T_{gas,3}(t) - T_{gas,2}(t)\right)A_{G-G}}{\left(\frac{dz_{graphite}}{k_{graphite}} + \frac{dz_{fuel}}{k_{graphite}}\right)} + \frac{\left(T_{clad,2}(t) - T_{gas,2}(t)\right)A_{S-Graph,C}}{\left(\frac{dz\_cond_{gas}}{k_{gas}} + \frac{dz\_cond_{clad}}{k_{clad}}\right)} + \frac{\left(T_{gas,2}(t) - T_F(t)\right)A_{S-F,G}}{\left(\frac{dz\_cond_{gas}}{k_{gas}} + \frac{dz\_cond_{fuel}}{k_{fuel}}\right)}\right] \quad (69)$$

The axial region will determine either fuel or graphite interaction.

Clad Energy Balance

The cladding interacts conductively with the gas and clad, and convectively with the coolant channel. The energy balance is:

$$\frac{dT_{clad,2}}{dt} = \frac{1}{m_{clad}(t)c_{p,cald}(t)}\left[\frac{\left(T_{clad,1}(t) - T_{clad,2}(t)\right)A_{C-C}}{\left(\frac{dz_{graphite}}{k_{clad}} + \frac{dz_{fuel}}{k_{clad}}\right)} + \frac{\left(T_{clad,3}(t) - T_{clad,2}(t)\right)A_{C-C}}{\left(\frac{dz_{graphite}}{k_{clad}} + \frac{dz_{fuel}}{k_{clad}}\right)} + \frac{\left(T_{gas,2}(t) - T_{clad,2}(t)\right)A_{S-Graph,C}}{\left(\frac{dz\_cond_{gas}}{k_{gas}} + \frac{dz\_cond_{clad}}{k_{clad}}\right)} - \tilde{h}_2(t)A_{s-C,w}\left(T_{clad,2}(t) - T_{13}(t)\right)\right] \quad (70)$$

This relationship is for the mid cladding; the other regions will have graphite interaction.

Water Channel Energy Balance

The water control volume balances the energy from the cladding surface temperatures and the mass balance. The mass of the water is found with the state's density and the volume of the coolant channel in that region.

$$\frac{dT_{coolant,2}}{dt} = \frac{1}{m_{coolant}(t)c_{p,coolant}(t)}\left[\tilde{h}_2(t)A_{s-C,w}\left(T_{clad,2}(t) - T_{13}(t)\right) + \dot{m}\left(\hat{h}_{in}(t) + gz_{in}\right) - \dot{m}\left(\hat{h}_{out}(t) + gz_{out}\right)\right] \quad (71)$$

The inlet and outlet enthalpies are state dependent variables calculated using the previous time step's regional temperatures.

The energy balance considers the effects of heat generation through the effective power calculation, from neutronics. Mass flow rate model leads to the fluid energy transport, while conduction completes the surface interaction of heat transfer around the control volume. The transient equations of the energy balance are added to the linear system of neutronics to complete the $\{\mathbb{R}^0, \mathbb{R}^2\}$ model.

Output

The reactor instantaneous power is not recorded during the solution process, it is found afterwards from the neutron population vector using (20) and (3). This array is used to find the maximum power and time of occurrence.

The fuel temperature vector represents a core average temperature. To find a limiting temperature, a peaking factor formula must be used. This formula is power dependent:

$$PF(P_{inst}) = 2.433e^{-19}{P_{inst}}^3 - 5.3307e^{-13}{P_{inst}}^2 + 4.1352e^{-7}P_{inst} + 1.0094 \quad (72)$$

This creates a vector the same length as the power and temperature vectors. The peaking factor and temperature vectors are multiplied elementally to find the transient peak temperature:

$$\vec{T}_{peak,n} = \vec{T}_{Fuel,m}PF_m \quad (73)$$

The peak temperature vector is then searched for a maximum value and associated time.

UT TRIGA Specifics

The $\{\mathbb{R}^0, \mathbb{R}^2\}$ input values are made specific for the University of Texas at Austin TRIGA reactor. Below are lists of the variables, their values, and the source.

Fuel Specifications

The fuel burnup was calculated using ORIGIN and SCALE to find the burnup factor from the original inventory and the number density of ^{239}Pu and other trace elements. The gas layer is considered to be Hydrogen and the cladding is 304 stainless steel. UT uses 8.5wt%, 19.7% enriched Uranium with a Zr:H ratio of 1.6.

Table 2. University of Texas Fuel Specific Model Parameters

Parameter	Value	Source
Outer Pin Radius	.018771 (m)	UT SAR[13]
Inner Zr Rod Radius	.003175 (m)	UT SAR[13]
Cladding Thickness	.000508 (m)	UT SAR[13]
Gas Thickness	1.3105e^{-4} (m)	Fenech[49]
Active Fuel Region	.381 (m)	UT SAR[13]
Graphite Height	.087376 (m)	UT SAR[13]
Total Pin Height	.73152 (m)	UT SAR[13]
Burn Factor ^{235}U	.922896237	ORIGIN
Burn Factor ^{238}U	.863175801	ORIGIN
$U_{wt\%}$	8.5 (%kg/kg)	UT SAR[13]
Enrichment	19.7 (%)	UT SAR[13]
Mass of ^{239}Pu	.406333 (kg)	ORIGIN
K_{Beta}	1.077	GA Paper, UT SAR[12], [13]
l^*	51.9e^{-6} (s)	ORIGIN, SCALE
AmBe Source Strength	7.4e^{10} (n^0/s)	Calibration Certificate
k_{fuel}	17.573 (W/m K)	Simnad[6]
$k_{graphite}$	112.4 (W/m K)	Engineering Toolbox[24]
$c_{p,gas}$	14.53e^3 (J/kg K)	PNNL[51]
$c_{p,clad}$	500 (J/kg K)	Makeitfrom[48]
h_{gas}	2.84e^3 (W/m^2 K)	KSU SAR[50]

Core Geometrics

The parameters below are related to the core configuration at The University of Texas at Austin.

Table 3. University of Texas Core Geometry Model Specifics

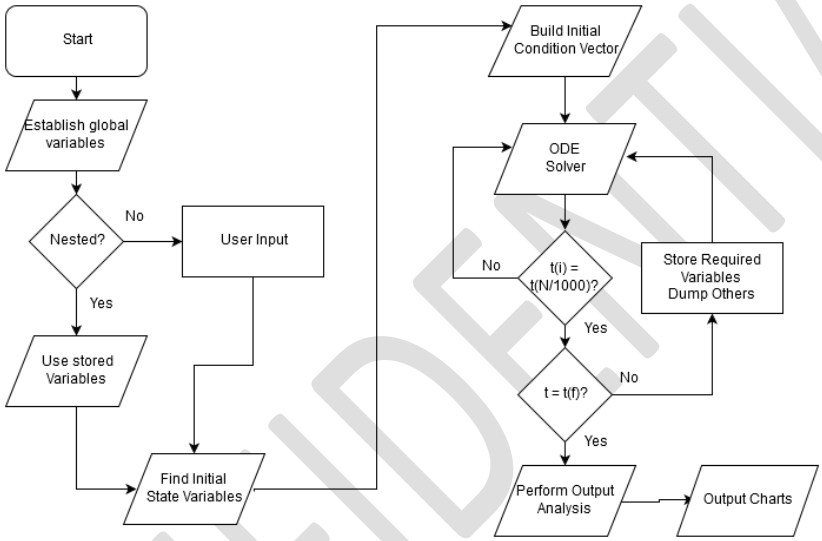
Parameter	Value	Source
Number of Pins*	113.7 (#)	UT SAR[13]
Inner Coolant Hex Radius	.0217678 (m)	UT Core Drawing
α_M	3.8952e^{-6} ($\frac{\Delta k/k}{C}$)	Lab Report[16]
Gravity	9.8066 (m/s^2)	UT SAR[13]
Kinematic Viscosity of Water	.279e^{-6} (m^2/s)	Engineering Toolbox[24]
Expansion Coefficient	.207e^{-3} (1/K)	Engineering Toolbox[24]

* The number of pins is a non-integer to account for the diameter variation between the fuel pins and the fuel followed control rods (FFCRs)

Programming

The $\{\mathbb{R}^0, \mathbb{R}^2\}$ model creates a set of 101 coupled linear ordinary differential equations and a number of state variables, requiring recalculation at each time step. These equations tend to be stiff due to rapid neutronics change relative to slower temperature changes, and require attention in solving.[2] It was decided that MATLAB would provide an appropriate environment due to its performance in handling linear systems.

The program is a set of triple nested code. A governing program is used to allow parallel computing of multiple events. This program calls the main function. The main function accounts for user input, site-



13. Block Diagram of UT TRIGA Solver

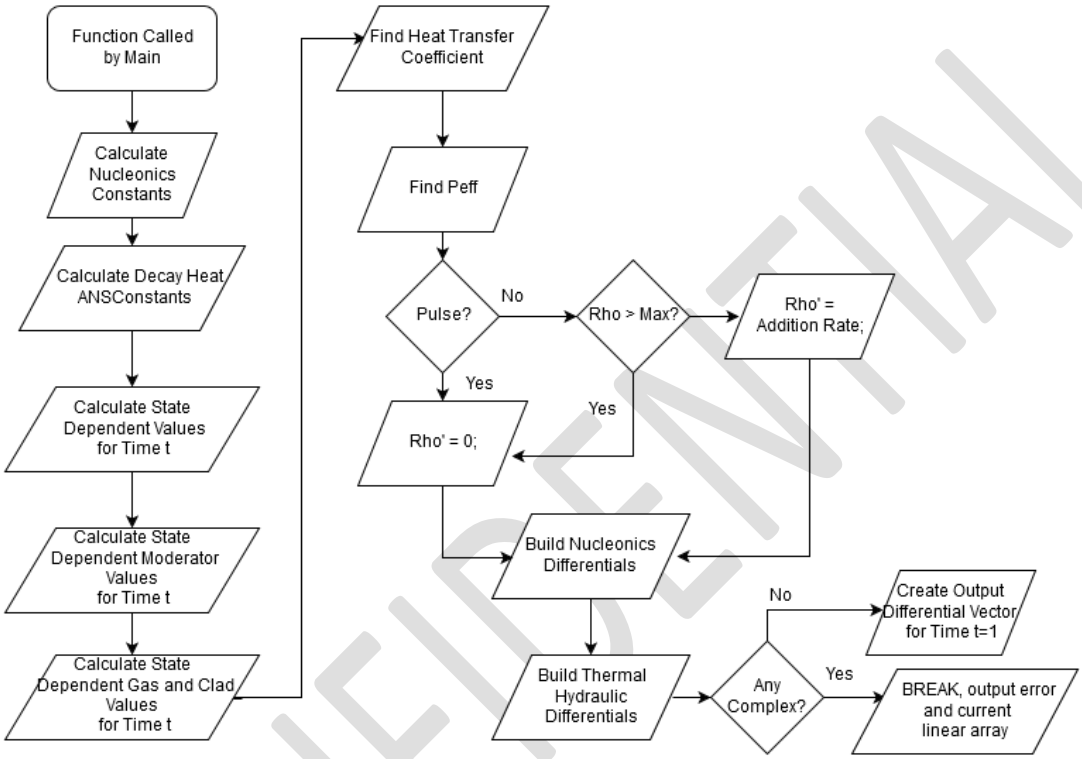
specific variables, calls the ODE, and creates outputs. The large ODE set and small time scale creates a memory burden. This is handled through creative use of a 'for' loop and ODE solver call. A basic flow diagram is below.

The program can be used in two ways, one as a called function from a governing program, the other prompts the user in the MATLAB command line. The program takes the initial conditions given and calculates the neutronics and thermal hydraulics based on a cold clean critical core for the input power.

During the ODE solution process a large amount of memory is used due to the system size and small time step. To increase performance, the ODE solver is nested. Beginning from time 0, it runs through a temporary final time dictated by the subdivision of the event final time. The column vectors of interest are stored in memory, the final row vector of time t(i) is stored as the initial condition of t(i+1) in the next loop, and the rest of the information, such as decay heat fractions, is deleted. This information is unnecessary for post run analysis but vital to an accurate solution. This method reduces the size of the read/write array the ode uses. Increases in performance include a 1/32nd reduction in memory usage and 1/5th reduction in solution time, while still tracking power and temperature for each time step.

ODE solver

ODE113 was the chosen ode solver. Mathworks recommends usage of ODE113 based on increased accuracy and performance.[52] The only issue of concern is the stiffness of the equations. ODE113 is a variable order method that uses information from previous iterations in calculation of the current. This makes it efficient and accurate. It uses a variable time step to ensure these meet the error requirements.[53] It also has the highest accuracy according to Mathworks. This method is used to solve the ODE function called by the main program:



14. ODE Functional Block Diagram

The function begins by finding the current constants based on the global variables and neutronics constants. The previous loops temperature data is used to find the state of moderator based parameters. The current heat transfer coefficient is found followed by the current powers from (19-21).

If the event is a pulse, the change in reactivity is 0, as a pulse adds all of the reactivity instantaneously. If the event is a rod withdraw, the current event reactivity is compared to the maximum reactivity limit set by user input. If it is equal or above this, the change Is 0, otherwise the change is based on a reactivity addition rate.

The differential vector is built from the differential equations. It is then checked for complex values. The elements being solved for are real and physical values and should not contain complex numbers.[2] This is a method of checking for calculation execution errors in code and physical representation. If there are no errors in the array it is passed to the next time step.

Output Variable Tracking and Handling

Upon completion of the solver, the code finds the maximum power and temperature as discussed above. It then creates a reduced array for output to plots. The solution arrays are on the order of $5e^8$ in length. To reduce the size but keep fidelity, a subset of the arrays is chosen using 10,000 points each. The program outputs: Power, Temperature, Reactivity, maximum times for power and temperature, and the reduced arrays.

If the main function is nested in a solver script, as is the case for VnV, the output values are stored in a cell structure and the next event is processed.

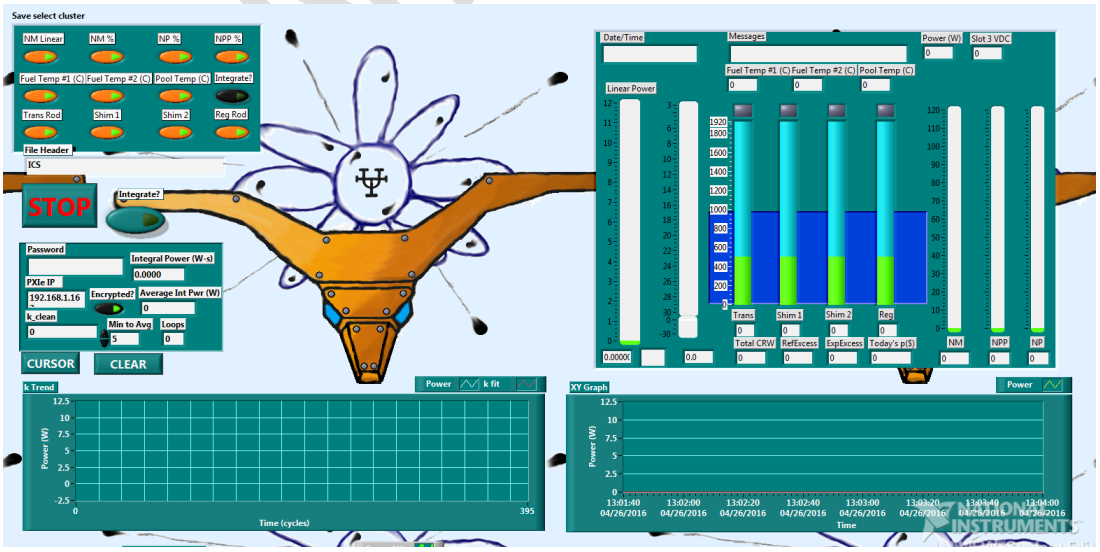
Verification and Validation

Verification and validation of the model involves a number of steps including: verification of equations, ODE stability analysis, ODE drift analysis, comparison with pulse maintenance, reactor operation rod withdraw validation.

Event Validation

The University of Texas at Austin has an annual pulse comparison maintenance, SURV-7, that makes pulse output comparison very easy. The inherent 'reactivity added' value from the QNX pulse record window is not calculated using 8.5wt% Uranium of Zr:H of 1.6. In order to find a better approximation of the added reactivity, the Fuchs-Hansen model was solved based on the QNX recorded peak powers.[54]

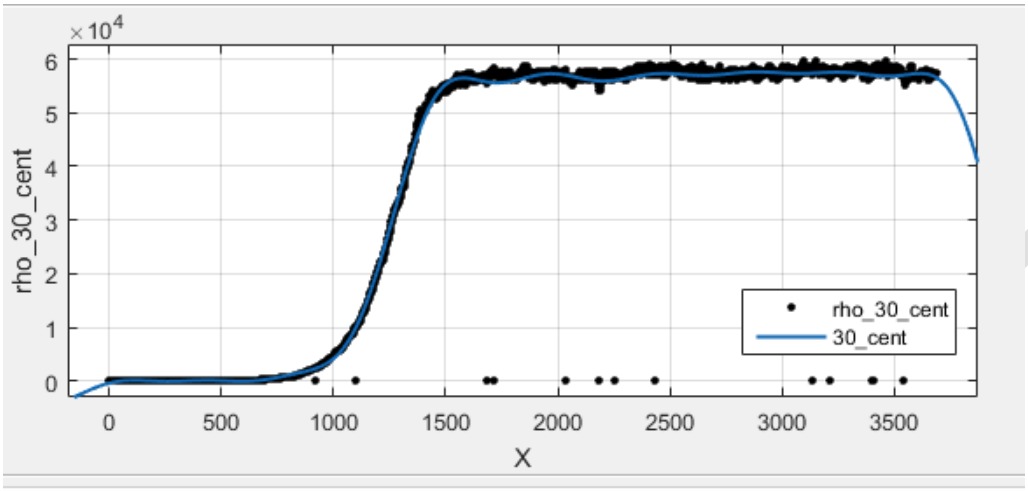
Rod withdraw validation is based on the UT TRIGA reactor and set reactivity additions. The transient rod was brought to a height nearing the flatter portion of the differential rod worth curve (~370 units). This creates the most constant reactivity addition rate possible, as the rod drive speed is constant and the rod worth is at its flattest. The reactor was left for at least 20min at 50W. The rod was withdrawn to add the desired amount of total reactivity: $.20, $.30, $.40, $.50. It was the left to sit for a minimum of 10 min without any external interaction. All of the values were recorded using a local UT ICS program.[55]



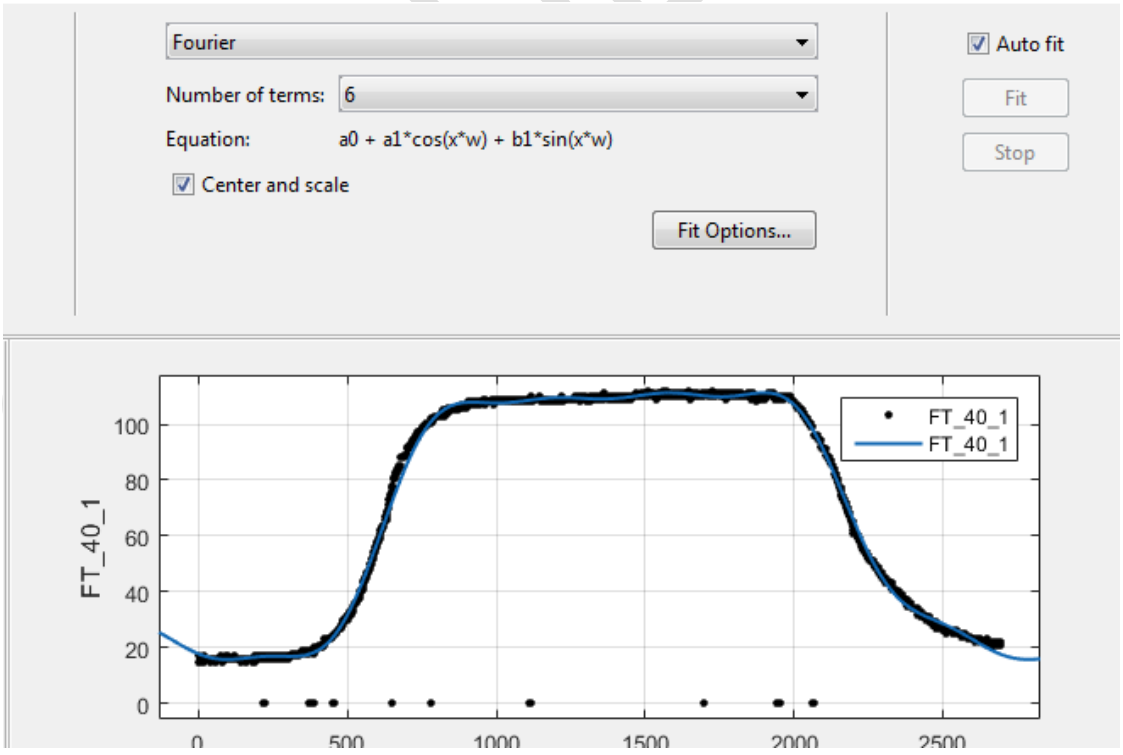
15. ICS Logging Program with LabVIEW and PXIe

The program continuously predicts, k_{eff}, allowing DNP equilibrium to be ensured. Additionally, it logs all outputs of the QNX screen at the same frequency the QNX program itself updates, allowing very little data loss and increased fidelity.

The data was analyzed using MATLAB's curve fitting tool to find the maximum powers and temperatures.



16. Curve Fitting $.30 Reactivty Addition Recorded Data

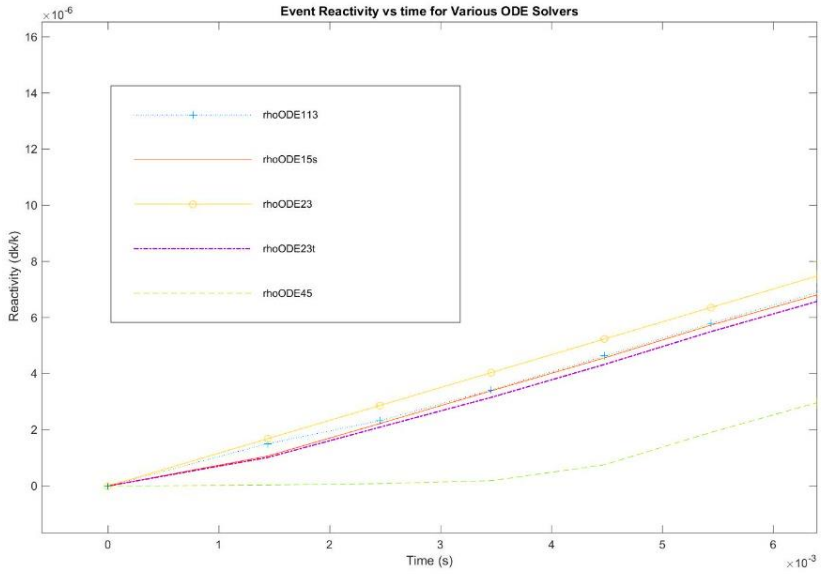


17. Curve Fitting Thermocouple Data

ODE Validation

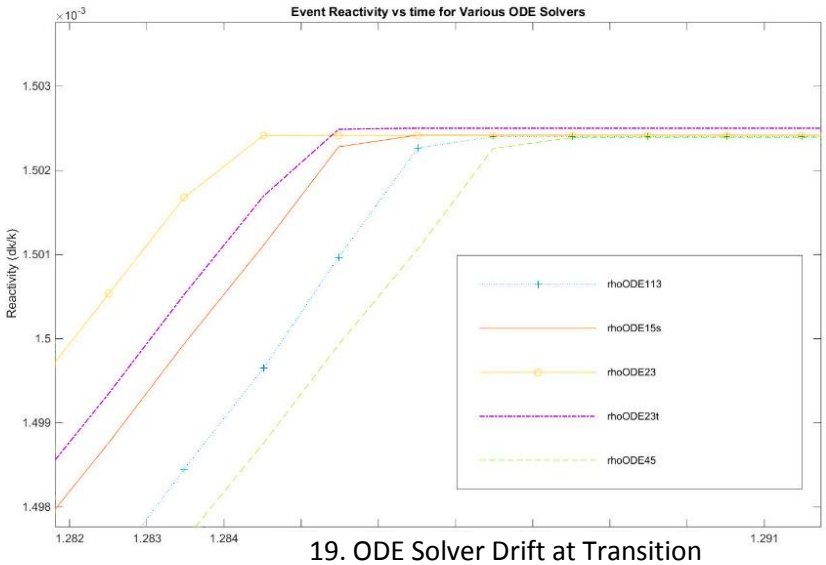
ODE Solver Drift

The ODE equation set's size and relationships led to mild stiffness and solver drift possibilities. The initial few seconds of the incident are some of the worst, as a prompt jump event is combined with a sluggish thermal system, as the reactor is below the point of adding heat. The rod withdraw event has a linear reactivity addition. If the ODE solver is drifting considerably, the first initial seconds' reactivity curve will be nonlinear, as shown below.



18. ODE Solver Drift in the Initial Model Time

This was the basis for choosing ODE 113, as the other options led to very nonlinear initial seconds. ODE45 was initially chosen, until initial drift, above, was noticed. The reactivity added from a rod



19. ODE Solver Drift at Transition

withdraw should be linear in this case, since the reactivity addition rate is constant. ODE23 and 15 provided excessive deviation in the previous $\{\mathbb{R}^0, \mathbb{R}^0\}$ model.

The additional benefit of ODE113 is the efficiency with high error tolerances. The default settings for ODE solvers is $1e^{-3}$ (~.1%) of the local and absolute values. This means the higher decimal places are neglected. With an even like a prompt jump or neutronics right at the point of deviation from criticality, this can be very error prone. The settings were increased to a local and absolute error of $1e^{-9}$, which decreases step size and increases solution time. This makes the choice of ODE113 even more justifiable.

ODE Perturbation Analysis

In addition to the ODE selections, the model was verified using uncertainty analysis. The factors affecting differential temperature were varied based on partial derivative error analysis:

$$y = f(\boldsymbol{x}, \boldsymbol{\theta}, t)\ where: \boldsymbol{x}\ \rightarrow variables, \boldsymbol{\theta} \rightarrow constants \tag{74}$$

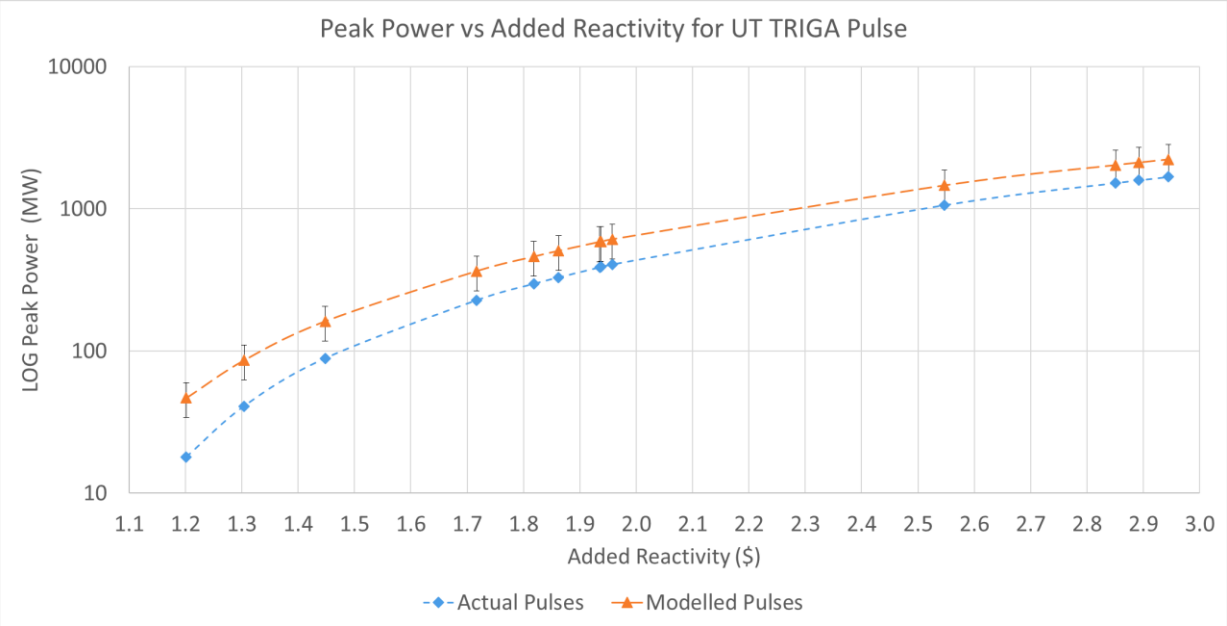
$$\delta y = \sqrt{\sum \left(\frac{\partial y}{\partial x_i}\delta x_i\right)^2 + \sum \left(\frac{\partial y}{\partial \theta_j}\delta \theta_j\right)^2} \tag{75}$$

In this method, the partial derivatives of the main function with respect to all variables and constants are found. In the model, constants are values like geometric dimensions. These have a fixed error that is state-independent. Variables are items that are state-dependent, such as Rayleigh number, or thermal conductivity. This leads to a maximum and minimum equation for the time step's change. The ODE set was rerun with this new value set and error bars on the model based on perturbations were found. This is expressed below by the use of error bars.

Results

Pulses

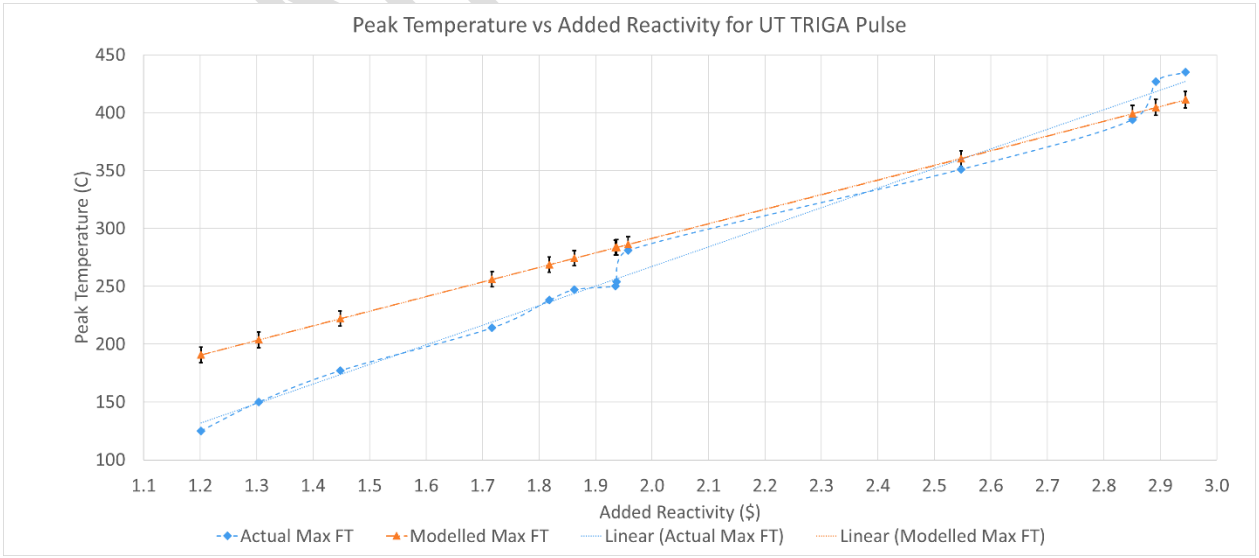
The $\{\mathbb{R}^0, \mathbb{R}^2\}$ model was run in pulse mode for all of the reactivities of the last 18 pulses. Comparisons were made between both peak power and peak temperature. The input reactivity was taken from the



21. Peak Power Vs. Added Reactivity for Modelled and Actual Pulses at UT TRIGA

actual peak powers of the pulses and solved using the Fuchs-Hansen Model

$$P_{max} = \frac{(\rho')^2}{2l\gamma} \tag{76}$$



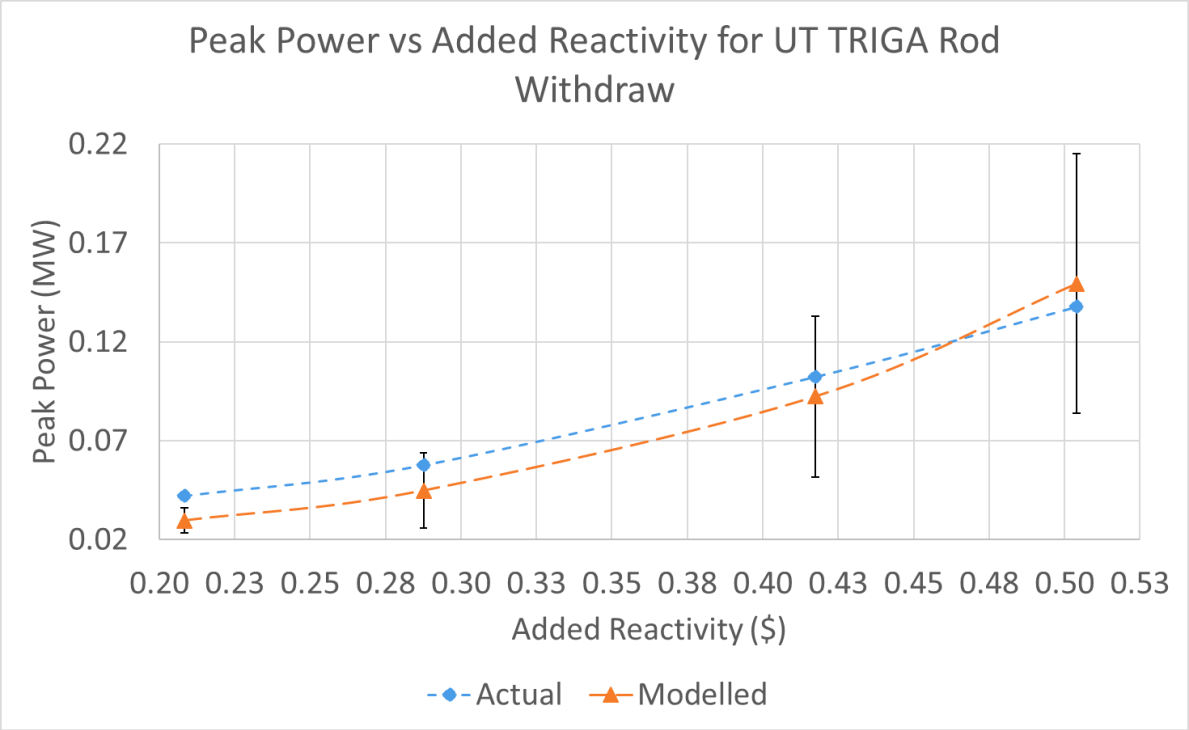
20. Peak Temperature Vs. Added Reactivity of Modelled and Actual Pulses at UT TRIGA

The error between the model and reality tends to decrease as the added reactivity increases. This can be seen by the convergence of the two curves on a log scale.

Errors in actual measured temperature include a new IFE installed on 2016, which has a higher peaking factor due to lower fission product poison inventory and minimal burnup.

Rod Withdraw

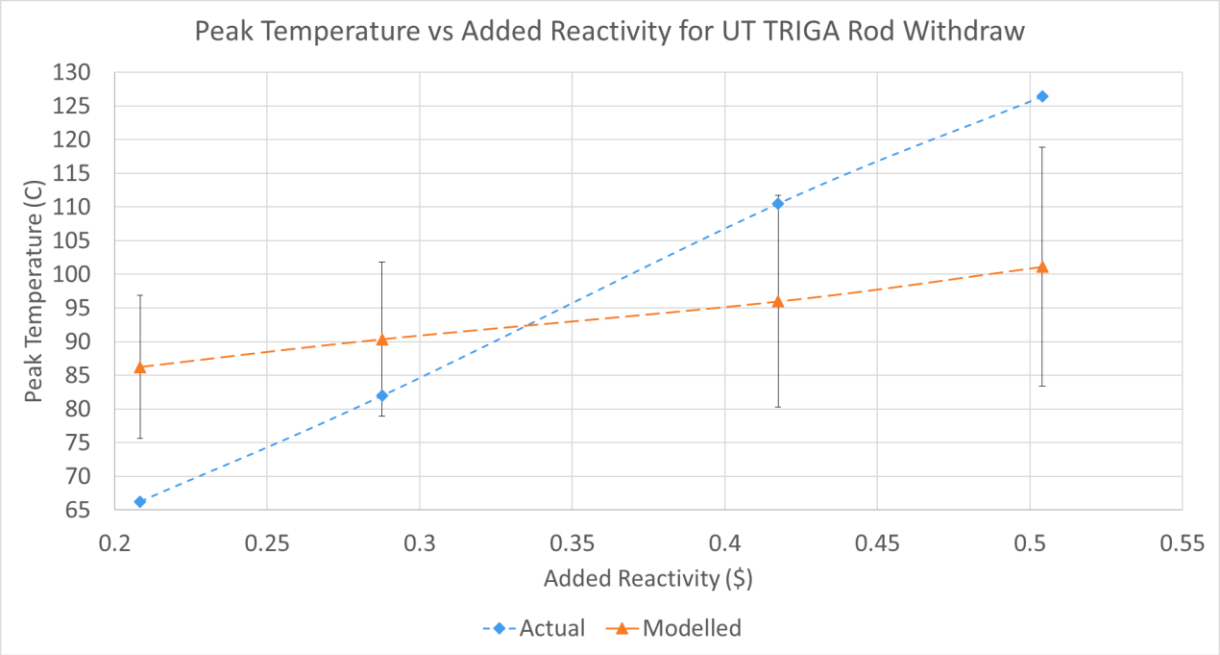
Four rod withdraw events were run in addition to the pulses. Reactivity additions above $.50 were avoided as the continuous withdraw causes periods beyond the measurability of the NM-1000 and this is considered undesirable.



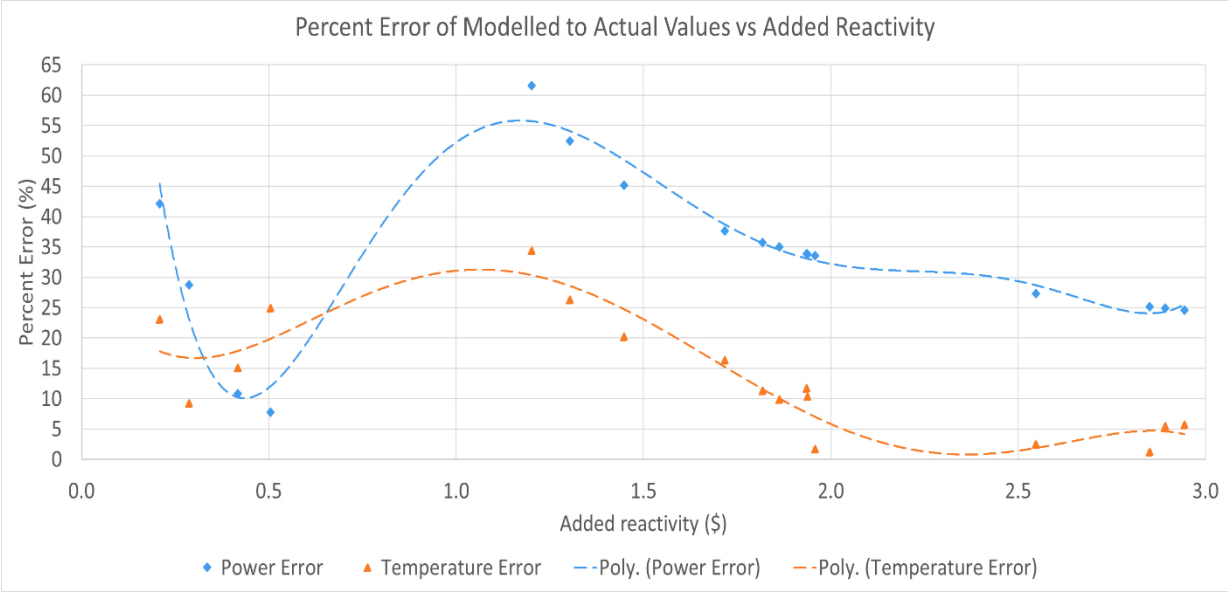
22. Peak Power Vs. Added Reactivity for Modelled and Actual Rod Withdraw Events at UT TRIGA

Much like the pulse event, the error decreases with reactivity. It is also worth noting the underestimation of temperature below $.35 and the overestimation above it, additionally the power follows the same trend but at $.47. This effect will be analyzed during the momentum model.

Both temperature and peak power errors decrease in absolute value with reactivity within the event. Further investigation is required for events with low reactivity additions and those in the area of prompt criticality.



24. Peak Temperature Vs Added Reactivity for Modelled and Actual Rod Withdraw Events



23. Percent Error of Modelled to Actual Values Vs. Event Reactivity

It is worth noting that the slopes of the temperature error trends for both pulse and rod withdraw are in the same relative direction. This implies a global trend error, possibly corrected in the momentum version.

The median errors and standard deviations can be found in the table below. The error is split by region for the pulses since the behavior of higher reactivity additions tends to have lower overall error between model and reality.

Table 4. Error Percentages in Model VnV

	Median Error (%)	Standard Deviation (%)
Pulse Peak Power	33.9	10.6
Pulse Peak Temperature	10.4	9.6
Pulse Peak Power ($1-1.5)	52.5	8.2
Pulse Peak Temperature ($1-1.5)	3.5	7.1
Pulse Peak Power ($1.5-2)	35.0	1.6
Pulse Peak Temperature ($1.5-2)	11.4	2.6
Pulse Peak Power (>$2.5)	25.2	3.8
Pulse Peak Temperature (>$2.5)	2.6	2.1
Rod Withdraw Peak Power	19.8	13.9
Rod Withdraw Peak Temperature	19.1	6.3

Sources of Error

The greatest source of error in the $\{\mathbb{R}^0, \mathbb{R}^2\}$ model is the lack of dimensions. Point kinetics using a reactivity balance is much less representative than an $\{\mathbb{R}^3, \mathbb{R}^3\}$ kinetic model would be.[2] However, the complexity increases by the power of n for each additional dimension in the calculations alone.[2] Adding to the point kinetics error is the use of a single energy neutron speed. Lethargy and core geometrics play key roles that are estimated, vice accounted for, in the reactivity balance.[2], [14] Additionally, an $\{\mathbb{R}^3, \mathbb{R}^3\}$ model would solve locally for each pin and its interactions with the surrounding pins, as well as control rod effects. This would remove the need for a peaking factor correction, as it would be part of the solution.

The constant mass flow rate assumption is better represented by a constant momentum balance over an increased amount of regions. This would incorporate the Darcy friction factor and flow processes occurring at the endpoints of the fuel pin.

The use of GA-7882 temperature coefficient of reactivity takes complex nuclear processes and lumps them into a temperature dependence. It is worth noting, the fuel temperature accuracy increased markedly in the model when switching from a constant value to this curve. This shows good correlation between actual UT TRIGA kinetics and those predicted in GA-7882.

The material properties' temperature dependence is curve fit to engineering tables. These tables and the measurements to make them carry inherent error. Alongside this, most fluids correlations are best fit data assumptions. The complexity of fluid flow goes well beyond that which can be correlated with a simple equation.

The ODE solver error was large at first but then reduced, however, the equation set has not been perturbed in an effort to find the bookend error values.

The site-specific values carry measurement error, both the values from UT SAR and drawings for the model as well as the measured values used in model VnV as the actual. For example, peak power measured by the QNX software.

When performing VnV, the input reactivity for the actual pulses is calculated using the Fuchs-Northeim model.[54], [56] This error pays into the final outcome but would have little to no bearing on the error

within the system. It acts almost as a mild perturbation on the ODE input, which is accounted for in ODE113.[57]–[61]

Unanticipated Transient Analyses

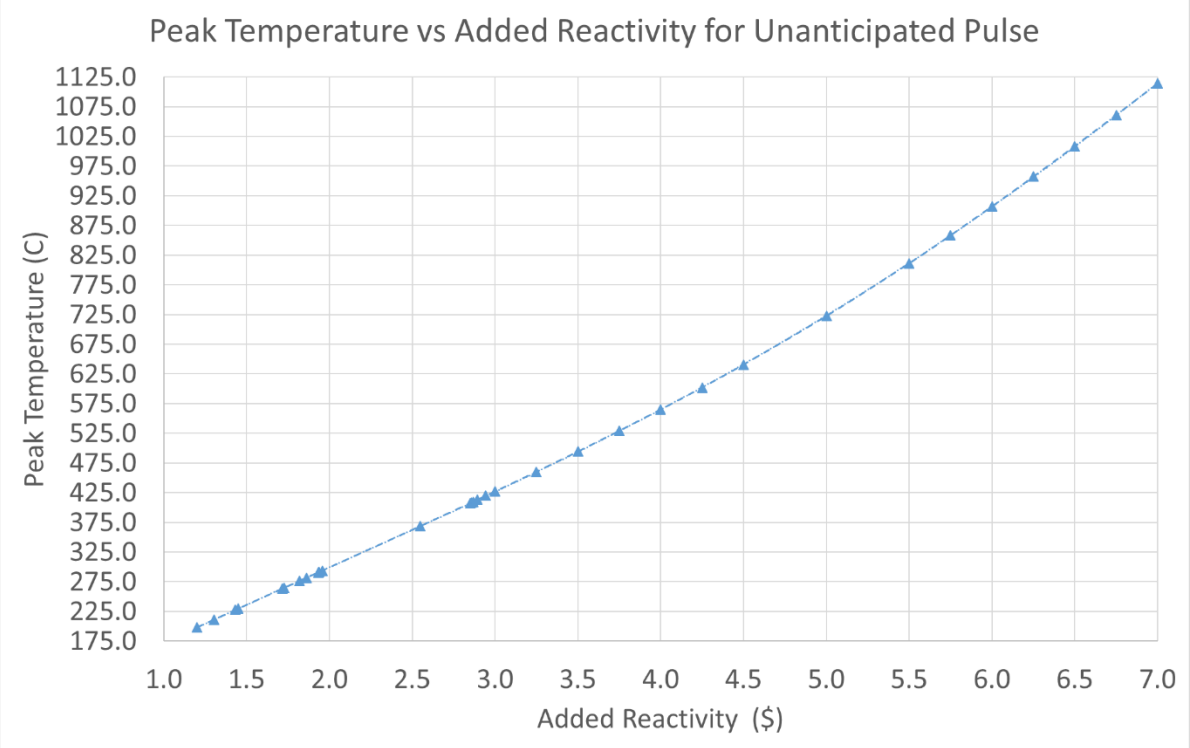
The model can be used to predict the outcome of events such as unanticipated reactivity addition events. This allows reactivity insertion limits to be analyzed. Using the criteria in previous work,[5], [7], [8], [13] and the model outputs, the limiting pulse reactivity additions are found below:

Table 5. Reactivity Addition Limits for Limiting Fuel Temperatures for TRIGA LEU

Limit	Reactivity Added ($)	Peak Temperature (C)	Percentage of Technical Specifications Limit (%)
Safety Limit[13]	6.21	950	213
Hawley Limit[5]	6.16	940	211
Argonne Pulse Limit[7]	5.60	830	187
Transient Rod Worth (2016)	3.36	475	107
UT Technical Specification	3.14	445	100

Pulse Event

The maximum reactivity added due to an unanticipated pulse event is limited by the total rod worth of the transient rod. The rack and pinion design of the standard control rods prevents any rod ejection event on the order of a pulse. Thus for the current core configuration the limiting pulse causes a peak temperature of only 475°C.

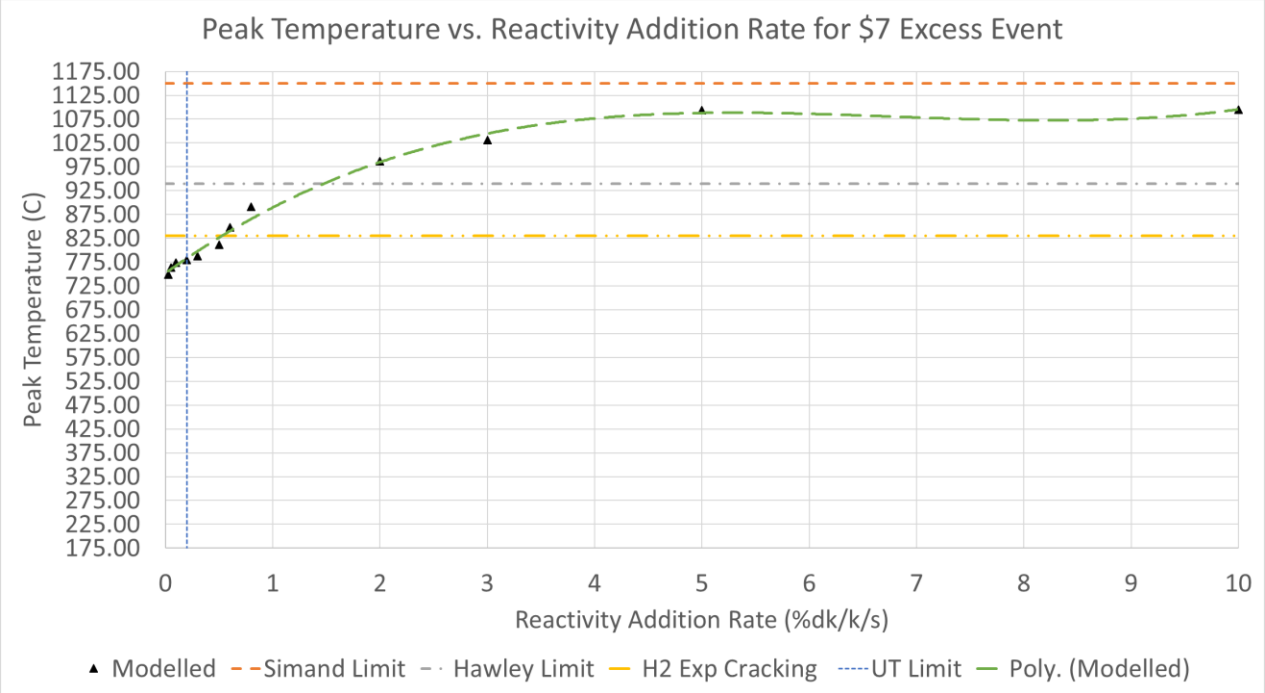


25. Predicted Peak Fuel Element Temperature Vs Added Reactivity

Given the extreme rate of rod ejection of the pulse under normal circumstances, the event is considered a step insertion and variations in transient rod pressure are ignored.

Rod Withdraw Event

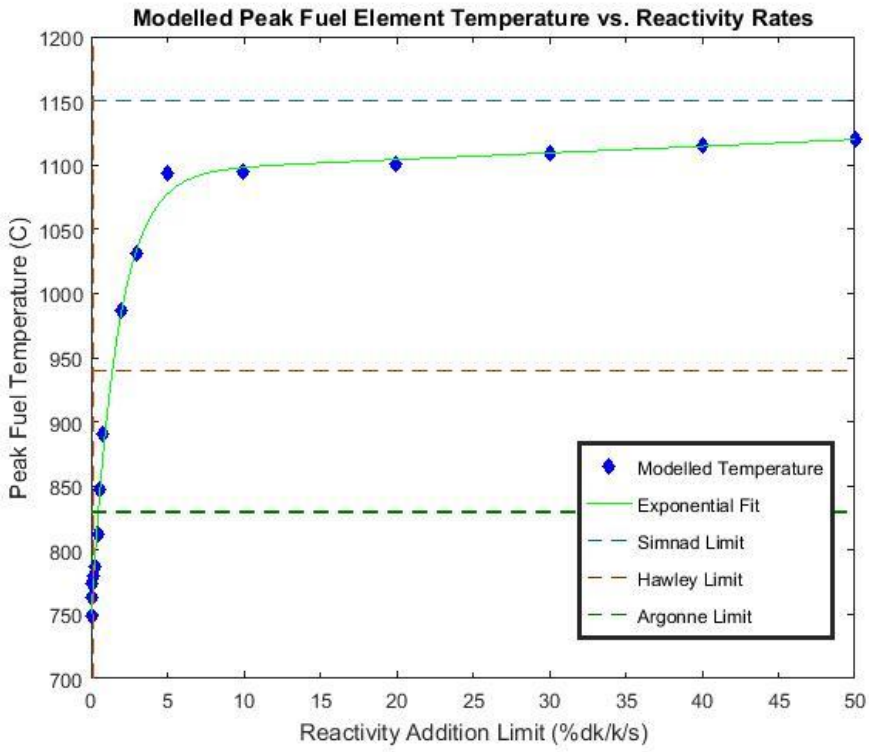
An unanticipated rod withdraw event can be caused by multiple sources including: grounded armature magnets, failed relay inputs, or control circuitry; however, any failure results in the same general outcome so only the reactivity addition event is analyzed. The maximum reactivity value used is the UT SAR's excess reactivity limit of $7.[13] This considers the worst possible case: a critical reactor with maximum possible excess reactivity.



26. Peak Temperature Vs. Reactivity Addition Rate for an Unanticipated Rod Withdraw Event

The figure above shows the peak temperatures modelled as a result of an unanticipated $7 insertion from a rod withdraw. The vertical blue line represents the UT reactivity addition limit, which shows that the if the event were to take place at UT with the maximum allowable reactivity, the Argonne suggested limit to prevent hydrogen cracking would not be exceeded.

It is also worth noting that above ~2%dk/k the peak temperature flattens and approaches, but does not exceed the Simnad limit.



27. Extended Analysis of Reactivity Addition Rate

Reactivity Addition Rate (%dk/k)	Added Energy (MWs)
0.025	26.78
0.05	26.81
0.10	27.93
0.20	28.81
0.30	29.98
0.50	31.52
0.60	33.71
0.70	37.02
0.80	42.53
0.90	44.60
2.00	45.02
3.00	58.53
5.00	92.07
10.00	129.57
Pulse	130.58

28. Energy added for a $7 reactivity additon

The energy additions for the events vary considerably and are directly related to reactivity addition rate.

Conclusions

From the modelling and data above it can be seen that peak fuel temperature is related to both the total reactivity added and the rate at which it is added. It can also be seen that lower reactivity addition rates may help to prevent exceeding the lower end temperature limits if combined with excess reactivity limits.

The trend toward levelling off peak temperature past 2%dk/k shows that, if considering only the Simnad limit, the need to limit rate is unnecessary but not prudent.

Lower addition rates allow for lower overall energy in the fueled region. This is presumed to be from a slower transient time. The slower time allows for more energy to be dissipated out of the fueled region to the pin and coolant.

Finally, it is evident that although the two events share the same process, they are fairly independent. This trend is due to the transient time of the event, and requires separate consideration.

Future Work

$\{\mathbb{R}^3, \mathbb{R}^3\}$

Expansion of the model into three dimensional vector spaces would require considerable overhaul. The neutronics would shift from a reactivity based equation to energy vectorization. This would include tracking neutron energy populations from birth through the scattered resonance absorption region until thermalization. This would require scattering and absorption probabilities based on materials and geometry. The fuel region would be divided into three dimensional sections with neutron diffusion tracking.

The feedback effects of the fuel would be quantified by tracking hydrogen energy separate from zirconium. Additionally, factors for Einstein oscillations, nucleus energy levels, cell effect and other effects of the fuel matrix would be broken into equations modelling their spate contributions to the energy vectorization.

The coolant would incorporate radial velocity as well as axial. Viscous effects would be considered. The fluidic grid size would be reduced. The additional cells would better profile the flow.

Quality Factor and Nucleate Boiling

Currently, the water is considered purely liquid, while in actuality nucleate boiling would change the water channel quality and increase the heat transfer from the pin.

GUI

A fully functional graphical user interface would be built. This would incorporate the option to choose which model will be run, allow for easy change in properties, allow multiple outputs, and provide a constantly updating output.

Improved Solution Methodology

Better methods for solving large ODE sets efficiently would be utilized. Additionally, a trend towards CFD solution methodology revolving around regionalization of time and residual tracking would allow the event to be solved in multiple sections and meshed together.

References

[1] M. Johnson and M. Johnson, “Modeling of Reactor Kinetics and Dynamics Modeling of Reactor Kinetics and Dynamics,” Idaho Falls, 2010.

[2] L. M. S. Ziya Akcasu, Gerald S. Lellouche, *Mathematical Methods in Nuclear Reactor Dynamics*. New York, NY: Academic Press, 1971.

[3] G. Atomics, “Technical Foundation of TRIGA,” San Diego, CA, 1958.

[4] D. R. Tobergte and S. Curtis, “Kinetic Behavior of TRIGA Reactors,” in *Conference on Utilization of Research Reactors*, 1967.

[5] R. L. K. S. C. Hawley, “NUREG/CR-2387: Credible Accident Analyses for TRIGA and TRIGA-Fueled Reactors,” 1982.

[6] M. T. Simnad, “The U-ZrHx Alloy: Its Properties and Use in TRIGA Fuel,” *Nucl. Eng. Des.*, vol. 64, pp. 403–422, 1981.

[7] Argonne National Laboratory, “Pulsing Temperature Limit for TRIGA LEU Fuel.pdf,” San Diego , CA, 2008.

[8] Argonne National Laboratory, “Fundamental Approach to TRIGA Steady-State Thermal-Hydraulic CHF Analysis,” San Diego, CA, 2007.

[9] G. Kline, “UT LOCA 2016,” 2016.

[10] G. Kline, “LOSS OF COOLANT ACCIDENT ANALYSIS FOR THE UNIVERSITY OF TEXAS AT AUSTIN,” 2016.

[11] D. Karnopp, R. Rosenberg, and A. S. Perelson, “System Dynamics: A Unified Approach,” *IEEE Trans. Syst. Man. Cybern.*, vol. 6, no. 10, 1976.

[12] D. R. Tobergte and S. Curtis, “GA-7882 Kinetic Behavior of TRIGA Reactors,” in *Conference on Utilization of Research Reactors*, 1967, p. 39.

[13] M.Krause, “The University of Texas at Austin TRIGA Safety and Analysis Report,” Austin, TX, 1991.

[14] L. E. Weaver, *Reactor Dynamics and Control*. New York, NY: American Elsevier Publishing Company, 1968.

[15] P. N. Haubenreich, “Prediction of Effective Yields of Delayed Neutrons in MSRE,” 1962.

[16] K. Dayman, “Laboratory 5 : Temperature Feedbacks on Reactivity,” 2013.

[17] V. E. Schrock, “A Revised ANS Standard for Decay Heat from Fission Products,” 1973.

[18] R. G. R. G. Rehm, R. Howard, and H. R. Baum, “The equations of motion for thermally driven, buoyant flows,” *J. Res. Natl. Bur. Stand. (1934).*, vol. 8, no. 3, p. 297, 1978.

[19] A. Wirth, “A Guided Tour Through Buoyancy Driven Flows and Mixing,” p. 66, 2015.

[20] F. P. Incropera, D. P. DeWitt, T. L. Bergman, and A. S. Lavine, *Fundamentals of Heat and Mass Transfer*, vol. 6th. 2007.

[21] T. L. Bergman, A. S. Lavine, F. P. Incropera, and D. P. DeWitt, *Fundamentals of Heat and Mass*

Transfer. 2011.

[22] J. Cleveland, N. Aksan, P. Vijayan, and A. Nayak, “Natural circulation in water cooled nuclear power plants,” *Ewp.Rpi.Edu*, no. November, 2005.

[23] D. GmBH, “Liquid Density Calculaiton,” 2016. [Online]. Available: http://ddbonline.ddbst.de/DIPPR105DensityCalculation/DIPPR105CalculationCGI.exe?component=Water.

[24] Engineeringtoolbox.com, “Property Tables,” *Engineeringtoolbox.com*, 2016. [Online]. Available: http://www.engineeringtoolbox.com/water-thermal-properties-d_162.html.

[25] N. Convection, “Natural Convection,” vol. 1, pp. 735–777, 2003.

[26] P. Talukdar, “Natural/Free Convection.”

[27] D. Coast, “Plumes and Thermals,” pp. 163–180.

[28] D. T. Allen and C. J. Durrenberger, “Gaussian Plume Modeling.” 2014.

[29] R. Huang, “Lecture 7: Reduced Gravity models of the wind-driven circulation,” pp. 1–19, 2006.

[30] A. Example and G. C. Equation, “Extended Surface Heat Transfer,” no. 2, pp. 1–10.

[31] D. Roncati, “Iterative calculation of the heat transfer coefficient,” no. 2.

[32] N. G. Narve, N. K. Sane, and R. T. Jadhav, “Natural Convection Heat Transfer from Symmetrical Triangular Fin Arrays on Vertical Surface,” vol. 4, no. 5, pp. 775–780, 2013.

[33] S. Edition, “10–6 ■ heat transfer from finned surfaces,” pp. 432–447, 2008.

[34] D. W. Mackowski, “Conduction Heat Transfer Notes for MECH 7210.”

[35] L. F. Crabtree, R. L. Dommett, and J. G. Woodley, “Estimation of Heat Transfer to Flat Plates , Cones and Blunt Bodies : Estimation of Heat Transfer to Flat Plates , Cones and Blunt Bodies,” no. 3637, 1970.

[36] P. Talukdar, “HEAT CONDUCTION THROUGH.”

[37] X. He, D. P. M. Van Gils, E. Bodenschatz, and G. Ahlers, “Prandtl- and Rayleigh-number dependence of Reynolds numbers in turbulent enard convection at high Rayleigh and small Prandtl numbers,” pp. 1–5.

[38] R. S. Subramanian, “Natural or Free Convection,” pp. 1–7.

[39] S. Mirapalli and P. S. Kishore, “Heat Transfer Analysis on a Triangular Fin,” vol. 19, no. 5, pp. 279–284, 2015.

[40] F. M. White, *Fluid Mechanics*. 2003.

[41] P. State, “External Flow Correlations (Average , Isothermal Surface) Internal Flow Correlations (Local , Fully Developed Flow),” 2016.

[42] B. R. S. Bartlett and B. R. S. Bartlett, “Tables of Supersonic Symmetrica ! Flow around Right Circular Cones , with and without the Addition of Heat at the Wave Tables of Supersonic Symmetrical Flow around Right Circular Cones , with and without the Addition of Heat at the

Wave," 1968.

[43] "G5A0V." .

[44] J. F. Wendt and P. Fn, "AIR FORCE SYSTEMS COMMAND," 1972.

[45] K. J. Yaser Hadad, "Laminar Forced Convection Heat Transfer from Isothermal Bodies with Unit Aspect Ratio," pp. 443–451, 2008.

[46] M. OCW, "fin-design." .

[47] W. G. Luscher and K. J. Geelhood, "Material Property Correlations : Comparisons between FRAPCON, FRAPTRAN and MATPRO," no. August, 2010.

[48] "Material Properties of Metals," 2016. [Online]. Available: makeitfrom.com.

[49] Henri Fenech, *Heat Transfer and Fluid Flow in Nuclear Systems*. Pergamon Press, 1981.

[50] Kansas State, "Kansas State University Safety and Analysis Report '06." KSU, Manhatten, 2006.

[51] "Hydrogen properties." .

[52] "Mathworks," *Mathworks.com*, 2016. [Online]. Available: http://www.mathworks.com/help/matlab/ref/ode45.html?s_tid=gn_loc_drop#References.

[53] "Mathworks," *Mathworks.com*, 2016. .

[54] C. Johnson, "Lab 6 – Reactor Pulse," 2013.

[55] G. Kline, "PXIe_ICS_Power_Cal_Etc_2015." Greg Kline, Austin, TX, p. 100, 2015.

[56] N. I. S. D. of the IAEA, "IAEA TRIGA," 2004. [Online]. Available: https://ansn.iaea.org/Common/documents/Training/TRIGA Reactors (Safety and Technology)/chapter1/characteristics33.htm. [Accessed: 01-Jan-2016].

[57] P. Howard, *Analysis of ODE Models*. 2009.

[58] L. F. Shampine, "Error Estimation and Control for ODEs," *J. Sci. Comput.*, vol. 25, no. 1, pp. 3–16, 2005.

[59] O. A. Chkrebtii and A. Science, "Probabilistic solution of differential equations for Bayesian uncertainty quantification and inference," 2013.

[60] H. Report, T. Mark, and I. I. Pulsing, "General. dynamics," 1998.

[61] H. Banks and S. Hu, "Propagation of Uncertainty in Dynamical Systems," *Ncsu.Edu*, 2011.

[62] G. Kline, "UT {R^0,R^2 } Model." 2016.

Appendix

Mass Balance Program

Main Function File Version 4.6.2

```
function [ P_out, T_out, t_out, rho_out, max_P, max_T, max_P_t, max_T_t ] =
in_hour_EQN_4_6_2_FUNC( R_in, t_in, RR_in, type)
%% HEADER
% In-hour EQN solver
% Author: Greg Kline
% Date: 08 Jan 2016
% Revision Date 15 Jun 2016
% Revision: 4.6.2
%
% Modelling of a pulse insertion as well as a rod withdraw event.
%
%
% The purpose of this code is to simulate both a pulse event and a continuous
% rod withdraw event.
%
% The parameters are modelled using the University of Texas parameters
% Revisions
% 4.1.0
%     - Added Plutonium 239 to mixture
%     - Accounted for core fuel burnup
% 4.6.0
%     - Expanded temperature regions to 6 from 2 ( 1 fuel 1 mod, to 3 fuel 3 mod)
%
% P_out is the vector of power vs time (MW)
% T_out is the vector of temperature vs time (C)
% t_out is time (s)
% rho_out is the event reactivity vs time (dk/k)
% max_P is max Power (MW)
% max_T is max temperature (C)
% max_P_t is the time of peak power (s)
% max_T_t is the time of peak temperature (s)
% R_in is the reactivity limit ($)
% t_in is the max time (s)
% RR_in is the reactivity addition rate in (dk/k/s)
% type is model type: 'Rod Withdraw' or 'Pulse'
%
% Example:
% [ P_out, T_out, t_out, rho_out, max_P, max_T, max_P_t, max_T_t ] =
in_hour_EQN_4_6_2_FUNC( 3, 10, .001170, 'Rod Withdraw')
%
% Pseudo Code
% Global Variables
%     - Define the global variables that talk to the ODE solving function
%
% User Input
%     - When not in function mode, the user input pings the MATLAB command
%         line for the same function inputs
%
% Fuel Constants
%     - Define the geometric constants IAW local values
%     - Define material properties IAW Simnad and local values
%
% Neutronics Constants
%     - Define isotope fractions
%     - Find the mixture delayed neutron fraction and decay constant
%     - Account for spontaneous fission
%
% Decay Heat Constants
%     - Define the ANS Standard decay heat constants
%
% Define State Dependent Functions using the initial conditions as the state
%     - Find material state properties
%     - Find initial flux
%     - Find instantaneous power
%
```

```
% Find Total Power
%
% Build Initial Condition Vector
%     - Neutronics ICs
%     - Thermal-hydraulic ICs
%     - Event Reactivity IC
%
% Output
%     - Find the instantaneous power from neutron population
%     - Find maximum
%     - Calculate peaking factor vector
%     - Apply to temperature vector
%     - Downsize output vectors
%

tic

%% Global Constants
global Tinit_fuel Tinit_mod  Max_reactivity Reactivity_add_rate
global  N_238 N_235 surface_area_fuel volume_core pin_height
global mass_U_235 mass_U_238 mass_fuel_mix  Event_type density_fuel
global N_Pu_239 mass_Pu_239 l_star  alpha_mod volume_cooling_core
global Area_pin_internal dz_cond_fuel mass_graphite_half_core
global surface_area_graphite area_cooling_pin graphite_height number_pins
global volume_cooling_graphite_core dz_cond_grap radius_pin inner_radius

%% User Input
% Default values are shown in []

% Model Type
% Constants for use in function method
Initial_Power = 50;
Tinit_fuel = 16;
Tinit_mod = 20;

% From user input
Max_reactivity = R_in;
t_f = t_in;
Reactivity_add_rate = RR_in;
Model_type = type;
Event_type = type;

% Make a nice display for the output
display(sprintf(['Your model is: \n' ...
    '   Model Type: %s \n' ...
    '   Reactivity addition rate(dk/k): %2.2d dk/k \n' ...
    '   Maximum Reactivity($): $%d \n' ...
    '   Initial Power(W): %dW \n' ...
    '   Initial Fuel Temperature(C): %dC \n' ...
    '   Initial Moderator Temperature(C): %dC \n' ...
    '   Duration (s): %ds \n'], Model_type, Reactivity_add_rate, ...
    Max_reactivity, Initial_Power, Tinit_fuel, Tinit_mod, t_f));

display(' Beginning Calculations  ... ');

 %% User Input Non-Function Version
% % Default values are shown in []
%
% % Model Type
%
% MT = input( 'Which model are you going to use? Pulse, Rod Withdraw, Both. [Both]',
's');
%
% if isempty(MT)
%     Model_type = 'Both';
%     Event_type = 'Pulse';
%
% elseif strcmp(MT, 'Rod Withdraw')
%     Model_type = 'Rod Withdraw';
%     Event_type = 'Rod Withdraw';
%
```

```
% elseif strcmp(MT, 'Pulse')
%     Model_type = 'Pulse';
%     Event_type = 'Pulse';
%
% elseif strcmp(MT, 'Both')
%     % This will run function file twice once with each model so select
%     % pulse first
%     Model_type = 'Both';
%     Event_type = 'Pulse';
%
% else
%     Model_type = 'Pulse';
%     Event_type = 'Pulse';
% end
%
% % Reactivity Limit/Insertion(Pulse)
% MR = input( 'What is the maximum reactivity for the event? ($) [$2.94]');
%
% if MR <= 0
%     display ('   Reactivity must be a positive value, using default');
%     Max_reactivity = 2.944447748;
%
% elseif MR >= 13
%     display (sprintf(['   This reactivity exceeds the total rod worth of the '...
%     'UT TRIGA core, results may not be accurate']));
%     Max_reactivity = MR;
%
% elseif isnan(MR)
%     display ('   Reactivity must be a numeric value, using default');
%     Max_reactivity = 2.944447748;
%
% elseif isempty(MR)
%     display ('   Reactivity must be a numeric value, using default');
%     Max_reactivity = 2.944447748;
%
% elseif MR > 3.1429 && MR < 13
%     display ('   Reactivity exceeds UT TechSpec limits ');
%     Max_reactivity = MR;
%
% else
%     Max_reactivity = MR;
% end
%
% % Initial Power (W)
% IP = input('What is the initial power? (W) [50W]');
%
% if IP <= 0
%     display ('   Initial Power must be a positive value, using default');
%     Initial_Power = 50;
%
% elseif isnan(IP)
%     display ('   Initial Power must be a numeric value, using default');
%     Initial_Power = 50;
%
% elseif isempty(IP)
%     display ('   Initial Power must be a numeric value, using default');
%     Initial_Power = 50;
%
% elseif IP > 1.1e6
%     display ('   Initial Power exceeds UT TechSpec limits ');
%     Initial_Power = IP;
%
% else
%     Initial_Power = IP;
% end
%
% % Initial Fuel Temperature (C)
% iFT = input('What is the initial fuel temperature? (C) [20C]');
%
% if iFT <= 0
%     display ('   Initial Fuel Temperature must be a positive value, using default');
```

```
%     Tinit_fuel = 20;
%
% elseif isnan(iFT)
%     display ('   Initial Fuel Temperature must be a numeric value, using default');
%     Tinit_fuel = 20;
%
% elseif isempty(iFT)
%     display ('   Initial Fuel Temperature must be a numeric value, using default');
%     Tinit_fuel = 20;
%
% elseif iFT > 950
%     display ('   Initial Fuel Temperature exceeds UT TechSpec limits ');
%     Tinit_fuel = iFT;
%
% else
%     Tinit_fuel = iFT;
% end
%
% % Initial Moderator Temperature (C)
% iMT = input('What is the initial moderator temperature? (C) [20C]');
%
% if iMT <= 0
%     display ('   Initial Moderator Temperature must be a positive value, using
default');
%     Tinit_mod = 20;
%
% elseif isnan(iMT)
%     display ('   Initial Moderator Temperature must be a numeric value, using
default');
%     Tinit_mod = 20;
%
% elseif isempty(iMT)
%     display ('   Initial Moderator Temperature must be a numeric value, using
default');
%     Tinit_mod = 20;
%
% elseif iMT > 48
%     display ('   Initial Moderator Temperature exceeds UT TechSpec limits ');
%     Tinit_mod = iMT;
%
% else
%     Tinit_mod = iMT;
% end
%
% % time length
% ft = input('How long of a run? (s)');
% % check values
% if ft <= -0
%     t_f = 100;
%     display('Time cannot be negative, using 100s');
%
% elseif isnan(ft)
%     t_f = 100;
%     display('Time must be a number, using 100s');
%
% elseif isempty(ft)
%     t_f = 100;
%     display('Time must be a number, using 100s');
%
% else
%     t_f = ft;
%
% end
%
% if sum(strcmp(Model_type, {'Both','Rod Withdraw'}))
%     RR = input('What is the reactivity addition rate?(dk/k) [.002dk/k]');
%
%     if RR <= 0
%         display ('   Reactivity must be a positive value, using default');
%         Reactivity_add_rate = .002;
%
```

```
%     elseif isnan(RR)
%         display ('   Reactivity must be a numeric value, using default');
%         Reactivity_add_rate = .002;
%
%     elseif isempty(RR)
%         display ('   Reactivity must be a numeric value, using default');
%         Reactivity_add_rate = .002;
%
%     elseif RR > .2
%         display ('   Reactivity exceeds UT TechSpec limits ');
%         Reactivity_add_rate = RR;
%
%     else
%         Reactivity_add_rate = RR;
%     end
%
%     % Make a  nice display for the output
%     display(sprintf(['Your model is: \n' ...
%         '   Model Type: %s \n' ...
%         '   Reactivity addition rate(dk/k): %2.2d dk/k \n' ...
%         '   Maximum Reactivity($): $%d \n' ...
%         '   Initial Power(W): %dW \n' ...
%         '   Initial Fuel Temperature(C): %dC \n' ...
%         '   Initial Moderator Temperature(C): %dC \n' ...
%         '   Duration (s): %ds \n'], Model_type, Reactivity_add_rate, ...
%         Max_reactivity, Initial_Power, Tinit_fuel, Tinit_mod, t_f));
%
% elseif ~sum(strcmp(Model_type, {'Both','Rod Withdraw'}))
%
%     Reactivity_add_rate = 0;
%
%     % Make a  nice display for the output
%     display(sprintf(['Your model is: \n' ...
%     '   Model Type: %s \n' ...
%     '   Maximum Reactivity($): $%d \n' ...
%     '   Initial Power(W): %dW \n' ...
%     '   Initial Fuel Temperature(C): %dC \n' ...
%     '   Initial Moderator Temperature(C): %dC \n' ...
%     '   Duration(s): %d \n'], Model_type, Max_reactivity, ...
%     Initial_Power, Tinit_fuel, Tinit_mod, t_f));
%
% else
%     display(' There is a model error');
%
% end
%
% display(' Beginning Calculations  ... ');

%% fuel constants %%%

%%% Geometry %%%
% Pin radius (m)
radius_pin = 0.018771; % .735in
inner_radius = 0.003175; %.125in

% clad width (m)
dl_clad = 0.000508;

% Active region (m)
pin_height = .381;
graphite_height = 0.087376;
dz_cond_fuel = .5 * pin_height;
dz_cond_grap = .5 * graphite_height;
Graphite_offset_from_tip = .087884;

% pin fuel volume (m^3)
volume_pin = pi * pin_height * ((radius_pin-dl_clad)^2 - inner_radius^2);

% pin graphite volume (m^3)
volume_graphite_pin = pi * radius_pin^2 * graphite_height;
```

```
% surface area graphite (m^2)
surface_area_graphite = 2 * pi * radius_pin * graphite_height;

% surface area of pin (m^2)
surface_area_fuel = 2 * pi * radius_pin * pin_height;

%%% Reactor Constants %%%
% Number of pins (#)
number_pins = 113.7; % account for FFCR smaller diameter

% Volume of the core (m^3)
volume_core = volume_pin * number_pins;

% Volume of graphite in one half of core (m^3)
volume_graphite_half_core = volume_graphite_pin * number_pins;

% Surface area core (m^2)
surface_area_fuel = surface_area_fuel * number_pins;

% Radius of cooling hexagon (m)
inner_hex = 0.0217678;
outer_hex = 0.025146;

% Total spatial area of cooling hex and pin (m^2)
total_space = sqrt(3)/2 * (2 * inner_hex)^2;

% Area of pin radially (m^2)
Area_pin_internal = pi * radius_pin^2;

% Area of cooling hexagon around pin (m^2)
area_cooling_pin = total_space - Area_pin_internal;

% Volume of cooling (m^3)
% total pin height (m)
total_pin_height = 0.73152;

% the water mass uses the entire height of pin, not just heated length
volume_cooling_pin = area_cooling_pin * pin_height;
volume_cooling_graphite_pin = area_cooling_pin * graphite_height;

volume_cooling_core = volume_cooling_pin * number_pins;
volume_cooling_graphite_core = volume_cooling_graphite_pin * number_pins;

% Flow inlet area (m^2)
% pin top (m^2)
area_top_pin = .00063789;

% area of grid plate hole (m^2)
area_grid_hole = pi * (0.038227/2)^2;

% flow area of a pin (m^2)
flow_area_pin_upper = area_grid_hole - area_top_pin;

flow_outlet_core = flow_area_pin_upper * number_pins;

flow_area_pin_lower = 0.00063885; %m^2

flow_inlet_core = flow_area_pin_lower * number_pins;

% Channel width (m)
width_channel = 0.0061976;

% Percent Burn (Fraction)
burn_factor_238 = 0.922896237;
burn_factor_235 = 0.863175801;

%%% Fuel meat constants %%%
% wt% U_238
U_wt = .085;

% enrichment percentage (fraction) [UT SAR]
```

```
Enrich = .197; % 19.7%

% density (kg/m^3) [Simnad]
density_U = 19070;

% density of ZrH based on ratio (kg/m^3)[Simnad]
density_Zr = 1 / (.1706 + .0042 * 1.6) * 1000;

% Material Densities (kg/m^3)
density_fuel = 1 / ( U_wt / density_U + (1 - U_wt) / density_Zr ); %[Simnad]

% density of Pu 239 based on burnup equations (kg/m^3)
density_Pu239 = 19618;

% density of graphite (kg/m3)
density_graphite = 2500;

% Avogadro's number (atoms/mol)
N_A = 6.022e23;

% Molar mass (kg/mol) [Burns]
M_U = .23807;

% Molar mass Pu239 (kg/mol)
M_Pu = .2390521634;

% Molar mass Sm (kg/mol)
M_Sm = .15036;

% Molar mass Zr (kg/mol)
M_Zr = .091224;

%%% Masses %%%
% Mass Uranium (kg)
mass_U_pin = density_fuel * volume_pin * U_wt;

% Mass U235 (kg)
mass_U_235_pin = mass_U_pin * Enrich;

% Mass U238 (kg)
mass_U_238_pin = mass_U_pin - mass_U_235_pin;

% Mass Pu239 (kg)
mass_Pu239_pin = density_Pu239 * volume_pin;

% Mass Zr per pin (kg)
mass_ZrH_pin = density_fuel * volume_pin * ( 1- U_wt );

% Total U238 mass (kg)
mass_U_238 = mass_U_238_pin * number_pins;

% Total U235 mass (kg)
mass_U_235 = mass_U_235_pin * number_pins;

% Total Pu239 mass (kg)
mass_Pu_239 = mass_Pu239_pin * number_pins;

% Total Fuel Mass (kg)
mass_fuel_mix = density_fuel * volume_pin * number_pins;

% Total mass of one half of graphite (kg)
mass_graphite_half_core = density_graphite * volume_graphite_half_core * number_pins;

%%% Number Densities %%%
% Number density of U238 per pin (atoms/m^3)
N_238_pin = ( mass_U_238_pin / M_U * N_A ) / volume_pin  * burn_factor_238;

% Number density of U235 per pin (atoms/m^3)
N_235_pin = ( mass_U_235_pin / M_U * N_A ) / volume_pin  * burn_factor_235;

% Number density Pu239 per pin (atoms/m^3)
```

```
% N_Pu_239_pin = ( mass_Pu_239 / M_Pu * N_A ) / volume_pin;
N_Pu_239_pin = 0.406333333 / M_Pu * N_A;

% Number density of Sm per pin (atoms )
% constants for Sm and Pu are found from neutronics burnup data and
% are in units of kg/m^3
N_Sm_pin = 0.077307608 / M_Sm * N_A * volume_pin;

% Number density of ZR per pin (atoms)
N_Zr_pin = ( mass_ZrH_pin / M_Zr * N_A ) / 2.6;

% Number density U238 per core (atoms/m^3)
N_238 = N_238_pin * number_pins;

% Number density U235 per core (atoms/m^3)
N_235 = N_235_pin * number_pins;

% Number Density Pu239 per core (atoms/m^3)
% This value is taken from burn data so it does not need corrected
N_Pu_239 = N_Pu_239_pin * number_pins;

% Number of Sm (atoms)
N_Sm = N_Sm_pin * number_pins;

% Number of Zr (atoms)
N_Zr = N_Zr_pin * number_pins;

% fuel negative coefficient of reactivity in del_k/k/C [UT SAR]
alpha_fuel = -1e-4;

% Moderator temperature coefficient (del_k/k/C)
 alpha_mod = 3.8952e-6;

%% neutronics constants
% Group Lambdas
% U235
U235_L_1 = .0127;
U235_L_2 = .0317;
U235_L_3 = .115;
U235_L_4 = .311;
U235_L_5 = 1.40;
U235_L_6 = 3.87;

% U238
U238_L_1 = .0132;
U238_L_2 = .0321;
U238_L_3 = .139;
U238_L_4 = .358;
U238_L_5 = 1.41;
U238_L_6 = 4.02;

% Pu239
Pu239_L_1 = .0129;
Pu239_L_2 = .0313;
Pu239_L_3 = .135;
Pu239_L_4 = .333;
Pu239_L_5 = 1.36;
Pu239_L_6 = 4.04;

% total delayed neutron fraction U238 [Weaver, 1968]
B_238 = .0157;
% group i fraction is B * relative abundance
B_1_238 = B_238 * .013; %group 1
B_2_238 = B_238 * .137;
B_3_238 = B_238 * .162;
B_4_238 = B_238 * .388;
B_5_238 = B_238 * .225;
B_6_238 = B_238 * .075;

% total delayed neutron fraction U235 [Weaver, 1968]
B_235 = .0065;
```

```
B_1_235 = B_235 * .038;
B_2_235 = B_235 * .213;
B_3_235 = B_235 * .188;
B_4_235 = B_235 * .407;
B_5_235 = B_235 * .128;
B_6_235 = B_235 * .026;

% total delayed neutron fraction Pu239
B_Pu_239 = .0026;
B_1_Pu_239 = B_Pu_239 * .038;
B_2_Pu_239 = B_Pu_239 * .280;
B_3_Pu_239 = B_Pu_239 * .216;
B_4_Pu_239 = B_Pu_239 * .328;
B_5_Pu_239 = B_Pu_239 * .103;
B_6_Pu_239 = B_Pu_239 * .035;

% define fission cross sections for precursor ratios
%fission cross section (m^2)
sig_f_238 = 3.3e-28;

%fission cross section (m^2)
sig_f_235 = 584.4e-28;

% fission cross section (m^2)
sig_f_Pu_239 = 747.4e-28;

% Absorption cross sections for poisons (m^2)
% Zr, Zirconium has 5 stable isotopes, NIST provides cross sections for each
% so a weighted average is taken Zr 90, 91, 92, 94, 95
sig_a_Zr = .5145 * .011e-28 + .1132 * 1.17e-28 + .1719 * .22e-28 ...
    + .1728 * .0499e-28 + .0276 * .0229e-28;

% Samarium has 7 stable states (m^2)
sig_a_Sm = .031 * .7e-28 + .151 * 57e-28 + .113 * 2.4e-28 ...
    + .139 * 42080e-28 + .074 * 104e-28 + .266 * 206e-28 ...
    + .226 * 8.4e-28;

% neutron velocity (m/s)

% Boltzmann constant (m2 kg s-2 K-1)
k_b = 1.38064852e-23;

% Mass of neutron (kg)
m_nu = 1.674927471e-27;

% velocity neutron = 2197;
velocity_neutron = sqrt( 2 * k_b * (Tinit_fuel + 273.15) / m_nu );

% U235 Beta factor (atoms/s)
Beta_factor_U235 = velocity_neutron * sig_f_235 * N_235;

% U238 Beta factor (atoms/s)
Beta_factor_U238 = velocity_neutron * sig_f_238 * N_238;

% Pu239 Beta factor (atoms/s)
Beta_factor_Pu239 = velocity_neutron * sig_f_Pu_239 * N_Pu_239;

% Beta of the DNP mix
B_1_mix = ( Beta_factor_U238 * B_1_238 + Beta_factor_U235 * B_1_235 ...
    + Beta_factor_Pu239 * B_1_Pu_239) ...
    / ( Beta_factor_U238 + Beta_factor_U235 + Beta_factor_Pu239);
B_2_mix = ( Beta_factor_U238 * B_2_238 + Beta_factor_U235 * B_2_235 ...
    + Beta_factor_Pu239 * B_2_Pu_239) ...
    / ( Beta_factor_U238 + Beta_factor_U235 + Beta_factor_Pu239);
B_3_mix = ( Beta_factor_U238 * B_3_238 + Beta_factor_U235 * B_3_235 ...
    + Beta_factor_Pu239 * B_3_Pu_239) ...
    / ( Beta_factor_U238 + Beta_factor_U235 + Beta_factor_Pu239);
B_4_mix = ( Beta_factor_U238 * B_4_238 + Beta_factor_U235 * B_4_235 ...
    + Beta_factor_Pu239 * B_4_Pu_239) ...
    / ( Beta_factor_U238 + Beta_factor_U235 + Beta_factor_Pu239);
B_5_mix = ( Beta_factor_U238 * B_5_238 + Beta_factor_U235 * B_5_235 ...
```

```
    + Beta_factor_Pu239 * B_5_Pu_239) ...
    / ( Beta_factor_U238 + Beta_factor_U235 + Beta_factor_Pu239);
B_6_mix = ( Beta_factor_U238 * B_6_238 + Beta_factor_U235 * B_6_235 ...
    + Beta_factor_Pu239 * B_6_Pu_239) ...
    / ( Beta_factor_U238 + Beta_factor_U235 + Beta_factor_Pu239);

% Lambda of the DNP mix
lambda_1 = ( Beta_factor_U238 * U238_L_1 + Beta_factor_U235 * U235_L_1 ...
    + Beta_factor_Pu239 * Pu239_L_1) ...
    / ( Beta_factor_U238 + Beta_factor_U235 + Beta_factor_Pu239);
lambda_2 = ( Beta_factor_U238 * U238_L_2 + Beta_factor_U235 * U235_L_2 ...
    + Beta_factor_Pu239 * Pu239_L_2) ...
    / ( Beta_factor_U238 + Beta_factor_U235 + Beta_factor_Pu239);
lambda_3 = ( Beta_factor_U238 * U238_L_3 + Beta_factor_U235 * U235_L_3 ...
    + Beta_factor_Pu239 * Pu239_L_3) ...
    / ( Beta_factor_U238 + Beta_factor_U235 + Beta_factor_Pu239);
lambda_4 = ( Beta_factor_U238 * U238_L_4 + Beta_factor_U235 * U235_L_4 ...
    + Beta_factor_Pu239 * Pu239_L_4) ...
    / ( Beta_factor_U238 + Beta_factor_U235 + Beta_factor_Pu239);
lambda_5 = ( Beta_factor_U238 * U238_L_5 + Beta_factor_U235 * U235_L_5 ...
    + Beta_factor_Pu239 * Pu239_L_5) ...
    / ( Beta_factor_U238 + Beta_factor_U235 + Beta_factor_Pu239);
lambda_6 = ( Beta_factor_U238 * U238_L_6 + Beta_factor_U235 * U235_L_6 ...
    + Beta_factor_Pu239 * Pu239_L_6) ...
    / ( Beta_factor_U238 + Beta_factor_U235 + Beta_factor_Pu239);

% Beta mix
% Beff correction factor based on Beff/B ratio for U235 (.007/.0065)
% Weaver and Hetrick show up to 30% difference between B and Beff
Beta_correction = 1.076923076923077;

% Beta mix
Beta_mix = B_1_mix + B_2_mix + B_3_mix + B_4_mix + B_5_mix + B_6_mix;

% Beff mix
Beff_mix = Beta_mix * Beta_correction;

%prompt neutron lifetime (s) [UT SAR]
l_star = 51.9e-6;
% l_star = 41e-6;

% 2 Ci AmBe source (atoms/s)
AmBe_source = 7.4e10;

% U238 spontaneous fission (atoms/s)
U238_spontaneous = 1.80e-2 * mass_U_238 * 1000; % factor in (n/g-s)

% U235 spontaneous fission (atoms/s)
U235_spontaneous = 7.43e-4 * mass_U_235 * 1000; % factor in (n/g-s)

% Pu239 spontaneous fission (atoms/s)
Pu_239_spontaneuos = 2.30e-2 * mass_Pu_239 * 1000;

%% Decay Heat Constants %%%
% Decay (1/s)
% U235
H_lambda_235(1) = 2.2138e1;
H_lambda_235(2) = 5.1587e-1;
H_lambda_235(3) = 1.9594e-1;
H_lambda_235(4) = 1.0314e-1;
H_lambda_235(5) = 3.3656e-2;
H_lambda_235(6) = 1.1681e-2;
H_lambda_235(7) = 3.5870e-3;
H_lambda_235(8) = 1.3930e-3;
H_lambda_235(9) = 6.2630e-4;
H_lambda_235(10) = 1.8906e-4;
H_lambda_235(11) = 5.4988e-5;
H_lambda_235(12) = 2.0958e-5;
H_lambda_235(13) = 1.0010e-5;
H_lambda_235(14) = 2.5438e-6;
H_lambda_235(15) = 6.6361e-7;
```

```
H_lambda_235(16) = 1.2290e-7;
H_lambda_235(17) = 2.7213e-8;
H_lambda_235(18) = 4.3714e-9;
H_lambda_235(19) = 7.5780e-10;
H_lambda_235(20) = 2.4786e-10;
H_lambda_235(21) = 2.2384e-13;
H_lambda_235(22) = 2.4600e-14;
H_lambda_235(23) = 1.5699e-14;

% U238
H_lambda_238(1) = 3.2881e00;
H_lambda_238(2) = 9.3805e-1;
H_lambda_238(3) = 3.7073e-1;
H_lambda_238(4) = 1.1118e-1;
H_lambda_238(5) = 3.6143e-2;
H_lambda_238(6) = 1.3272e-2;
H_lambda_238(7) = 5.0133e-3;
H_lambda_238(8) = 1.3655e-3;
H_lambda_238(9) = 5.5158e-4;
H_lambda_238(10) = 1.7873e-4;
H_lambda_238(11) = 4.9032e-5;
H_lambda_238(12) = 1.7058e-5;
H_lambda_238(13) = 7.0465e-6;
H_lambda_238(14) = 2.3190e-6;
H_lambda_238(15) = 6.4480e-7;
H_lambda_238(16) = 1.2649e-7;
H_lambda_238(17) = 2.5548e-8;
H_lambda_238(18) = 8.4782e-9;
H_lambda_238(19) = 7.5130e-10;
H_lambda_238(20) = 2.4188e-10;
H_lambda_238(21) = 2.2739e-13;
H_lambda_238(22) = 9.0536e-14;
H_lambda_238(23) = 5.6098e-15;

% Pu239
H_lambda_Pu239(1) = 1.0020e1;
H_lambda_Pu239(2) = 6.4330e-1;
H_lambda_Pu239(3) = 2.1860e-1;
H_lambda_Pu239(4) = 1.0040e-1;
H_lambda_Pu239(5) = 3.7280e-2;
H_lambda_Pu239(6) = 1.4350e-2;
H_lambda_Pu239(7) = 4.5490e-3;
H_lambda_Pu239(8) = 1.3280e-3;
H_lambda_Pu239(9) = 5.3560e-4;
H_lambda_Pu239(10) = 1.7300e-4;
H_lambda_Pu239(11) = 4.8810e-5;
H_lambda_Pu239(12) = 2.0060e-5;
H_lambda_Pu239(13) = 8.3190e-6;
H_lambda_Pu239(14) = 2.3580e-6;
H_lambda_Pu239(15) = 6.4500e-7;
H_lambda_Pu239(16) = 1.2780e-7;
H_lambda_Pu239(17) = 2.4660e-8;
H_lambda_Pu239(18) = 9.3780e-9;
H_lambda_Pu239(19) = 7.4500e-10;
H_lambda_Pu239(20) = 2.4260e-10;
H_lambda_Pu239(21) = 2.2100e-13;
H_lambda_Pu239(22) = 2.6400e-14;
H_lambda_Pu239(23) = 1.3800e-14;

% Power fraction
% U235
H_fraction_235(1) = 6.5057e-1;
H_fraction_235(2) = 5.1264e-1;
H_fraction_235(3) = 2.4384e-1;
H_fraction_235(4) = 1.3850e-1;
H_fraction_235(5) = 5.5440e-2;
H_fraction_235(6) = 2.2225e-2;
H_fraction_235(7) = 3.3088e-3;
H_fraction_235(8) = 9.3015e-4;
H_fraction_235(9) = 8.0943e-4;
H_fraction_235(10) = 1.9567e-4;
```

```
H_fraction_235(11) = 3.2535e-5;
H_fraction_235(12) = 7.5595e-6;
H_fraction_235(13) = 2.5232e-6;
H_fraction_235(14) = 4.9948e-7;
H_fraction_235(15) = 1.8531e-7;
H_fraction_235(16) = 2.6608e-8;
H_fraction_235(17) = 2.2398e-9;
H_fraction_235(18) = 8.1641e-12;
H_fraction_235(19) = 8.7797e-11;
H_fraction_235(20) = 2.5131e-14;
H_fraction_235(21) = 3.2176e-16;
H_fraction_235(22) = 4.5038e-17;
H_fraction_235(23) = 7.4791e-17;

% U238
H_fraction_238(1) = 1.2311e00;
H_fraction_238(2) = 1.1486e00;
H_fraction_238(3) = 7.0701e-1;
H_fraction_238(4) = 2.5209e-1;
H_fraction_238(5) = 7.1870e-2;
H_fraction_238(6) = 2.8291e-2;
H_fraction_238(7) = 6.8382e-3;
H_fraction_238(8) = 1.2322e-3;
H_fraction_238(9) = 6.8409e-4;
H_fraction_238(10) = 1.6975e-4;
H_fraction_238(11) = 2.4182e-5;
H_fraction_238(12) = 6.6356e-6;
H_fraction_238(13) = 1.0075e-6;
H_fraction_238(14) = 4.9894e-7;
H_fraction_238(15) = 1.6352e-7;
H_fraction_238(16) = 2.3355e-8;
H_fraction_238(17) = 2.8094e-9;
H_fraction_238(18) = 3.6236e-11;
H_fraction_238(19) = 6.4577e-11;
H_fraction_238(20) = 4.4963e-14;
H_fraction_238(21) = 3.6654e-16;
H_fraction_238(22) = 5.6293e-17;
H_fraction_238(23) = 7.1602e-17;

% Pu239
H_fraction_Pu239(1) = 2.0830e-1;
H_fraction_Pu239(2) = 3.8530e-1;
H_fraction_Pu239(3) = 2.2130e-1;
H_fraction_Pu239(4) = 9.4600e-2;
H_fraction_Pu239(5) = 3.5310e-2;
H_fraction_Pu239(6) = 2.2920e-2;
H_fraction_Pu239(7) = 3.9460e-3;
H_fraction_Pu239(8) = 1.3170e-3;
H_fraction_Pu239(9) = 7.0520e-4;
H_fraction_Pu239(10) = 1.4320e-4;
H_fraction_Pu239(11) = 1.7650e-5;
H_fraction_Pu239(12) = 7.3470e-6;
H_fraction_Pu239(13) = 1.7470e-6;
H_fraction_Pu239(14) = 5.4810e-7;
H_fraction_Pu239(15) = 1.6710e-7;
H_fraction_Pu239(16) = 2.1120e-8;
H_fraction_Pu239(17) = 2.9960e-9;
H_fraction_Pu239(18) = 5.1070e-11;
H_fraction_Pu239(19) = 5.7300e-11;
H_fraction_Pu239(20) = 4.1380e-14;
H_fraction_Pu239(21) = 1.0880e-15;
H_fraction_Pu239(22) = 2.4540e-17;
H_fraction_Pu239(23) = 7.5570e-17;

%% State dependent functions
display(' Calculating State Dependent Functions ...');

% place holder for delta t
t_last = 0;

% volumetric heat capacity from Simnad (J/ m3 K)
```

```
cp_fuel_vol = (2.04 + 4.17e-3 * Tinit_fuel ) * 1e6;

% Convert to specific heat ( J / kg K )
cp_fuel = cp_fuel_vol / density_fuel;

% find keff
current_keff = 1;

% keff adjusted neutron lifetime (s)
A = l_star / current_keff;

% Heat transfer coefficient (W/m^2K) UT LOCA
h = 3200;  %3200 default

% Instantaneous Power (W)
% Find the initial flux and neutron population from user input
% Flux ( n / m^2 s)
initial_flux = Initial_Power / ( ...
    (200e6 * 1.602677e-19) * N_235 * sig_f_235 * volume_core ...
    + (200e6 * 1.602677e-19) * N_238 * sig_f_238 * volume_core ...
    + (200e6 * 1.602677e-19) * N_Pu_239 * sig_f_Pu_239 * volume_core );

% neutron density (n/m^3)
initial_neutron_density = initial_flux / velocity_neutron;

% U235 contribution
P_inst_235 = initial_flux * (200e6 * 1.602677e-19) * N_235 ...
    * sig_f_235 * volume_core;

% U238 contribution
P_inst_238 = initial_flux * (200e6 * 1.602677e-19) * N_238 ...
    * sig_f_238 * volume_core;

% Pu239 contribution (W)
P_inst_Pu239 = initial_flux * velocity_neutron * (200e6 * 1.602677e-19) * N_Pu_239 ...
    * sig_f_Pu_239 * volume_core;

% Total instantaneous (W)
P_inst = P_inst_235 + P_inst_238 + P_inst_Pu239;

% Temperature Dependent Density (kg/m^3) [ddbst.de, 273K-648K]
A_cp = .14395;
B_cp = .0112;
C_cp = 649.727;
D_cp = .05107;

density_init_water = A_cp / ( B_cp^(1 + (1 - (Tinit_mod + 273.15)/C_cp)^D_cp));

% Mass of water (kg)
mass_cooling_water = volume_cooling_core * density_init_water;

% Temperature dependent Specific heat (KJ/kg) [steam tables]
cp_H2O_KJ = 3.16744e-10 * Tinit_mod^4 - 1.05772e-7 * Tinit_mod^3 ...
    + 2.35330e-5 * Tinit_mod^2 - 1.47670e-3 * Tinit_mod + 4.20617e0;

% Convert to J/kg
cp_H2O = cp_H2O_KJ * 1000;

% specific heat capacity of graphite (cal / g C) [Entegris, inc.]
cp_graphite_cal = .10795e8 * Tinit_fuel^-3 - .61257e5 * Tinit_fuel^-2 ...
    + .30795e-4 * Tinit_fuel + .44391;
cp_graphite = 4183.995381 * cp_graphite_cal;

% Temperature dependent alpha_t
alpha_fuel_T = -1.62895e-7 * Tinit_fuel - 2.53543e-5;

%% Delayed Power State Function (W)
display(' Calculating Delayed Power Initial Condition... ');

% U235 delayed
P_d_235 =  ( P_inst_235 / 200 ) * sum( H_fraction_235 ./ H_lambda_235 );
```

```
% U238 delayed
P_d_238 =  ( P_inst_238 / 200 ) * sum( H_fraction_238 ./ H_lambda_238 );

% Pu239 delayed (W)
P_d_Pu_239 = ( P_inst_Pu239 / 200 ) * sum( H_fraction_Pu239 ./ H_lambda_Pu239 );

%% Total power by isotope (W)
% U235
P_i_235 = P_inst_235 + P_d_235;

% U238
P_i_238 = P_inst_238 + P_d_238;

% Pu239
P_i_Pu239 = P_inst_Pu239 + P_d_Pu_239;

%% Initial Condition Vector
display(' Building Initial Condition Vector... ');

% Build initial condition vector based on inputs
% In hour items
IC(1) = initial_neutron_density;
IC(2) =  (B_1_mix * initial_neutron_density / A ) / lambda_1;
IC(3) =  (B_2_mix * initial_neutron_density / A ) / lambda_2;
IC(4) =  (B_3_mix * initial_neutron_density / A ) / lambda_3;
IC(5) =  (B_4_mix * initial_neutron_density / A ) / lambda_4;
IC(6) =  (B_5_mix * initial_neutron_density / A ) / lambda_5;
IC(7) =  (B_6_mix * initial_neutron_density / A ) / lambda_6;

% Sources
% AmBe (n/s/m^3)
IC(8) = AmBe_source;
IC(9) = U238_spontaneous;
IC(10) = U235_spontaneous;

% Reactivity (assuming keff = 1, this IC is too balance the moderator and fuel
IC(12) = Tinit_fuel;
IC(13) = Tinit_mod;

% Betas
IC(14) = Beff_mix;
IC(15) = B_1_mix;
IC(16) = B_2_mix;
IC(17) = B_3_mix;
IC(18) = B_4_mix;
IC(19) = B_5_mix;
IC(20) = B_6_mix;

display(' Building Decay Heat Matrix Initial Condition... ');

% Decay heat matrix
% U235 Fraction
IC(21) = (H_fraction_235(1) * P_i_235) / (200 * H_lambda_235(1));
IC(22) = (H_fraction_235(2) * P_i_235) / (200 * H_lambda_235(2));
IC(23) = (H_fraction_235(3) * P_i_235) / (200 * H_lambda_235(3));
IC(24) = (H_fraction_235(4) * P_i_235) / (200 * H_lambda_235(4));
IC(25) = (H_fraction_235(5) * P_i_235) / (200 * H_lambda_235(5));
IC(26) = (H_fraction_235(6) * P_i_235) / (200 * H_lambda_235(6));
IC(27) = (H_fraction_235(7) * P_i_235) / (200 * H_lambda_235(7));
IC(28) = (H_fraction_235(8) * P_i_235) / (200 * H_lambda_235(8));
IC(29) = (H_fraction_235(9) * P_i_235) / (200 * H_lambda_235(9));
IC(30) = (H_fraction_235(10) * P_i_235) / (200 * H_lambda_235(10));
IC(31) = (H_fraction_235(11) * P_i_235) / (200 * H_lambda_235(11));
IC(32) = (H_fraction_235(12) * P_i_235) / (200 * H_lambda_235(12));
IC(33) = (H_fraction_235(13) * P_i_235) / (200 * H_lambda_235(13));
IC(34) = (H_fraction_235(14) * P_i_235) / (200 * H_lambda_235(14));
IC(35) = (H_fraction_235(15) * P_i_235) / (200 * H_lambda_235(15));
IC(36) = (H_fraction_235(16) * P_i_235) / (200 * H_lambda_235(16));
IC(37) = (H_fraction_235(17) * P_i_235) / (200 * H_lambda_235(17));
IC(38) = (H_fraction_235(18) * P_i_235) / (200 * H_lambda_235(18));
```

```
IC(39) = (H_fraction_235(19) * P_i_235) / (200 * H_lambda_235(19));
IC(40) = (H_fraction_235(20) * P_i_235) / (200 * H_lambda_235(20));
IC(41) = (H_fraction_235(21) * P_i_235) / (200 * H_lambda_235(21));
IC(42) = (H_fraction_235(22) * P_i_235) / (200 * H_lambda_235(22));
IC(43) = (H_fraction_235(23) * P_i_235) / (200 * H_lambda_235(23));

% U238 Fraction
IC(44) = (H_fraction_238(1) * P_i_238) / (200 * H_lambda_238(1));
IC(45) = (H_fraction_238(2) * P_i_238) / (200 * H_lambda_238(2));
IC(46) = (H_fraction_238(3) * P_i_238) / (200 * H_lambda_238(3));
IC(47) = (H_fraction_238(4) * P_i_238) / (200 * H_lambda_238(4));
IC(48) = (H_fraction_238(5) * P_i_238) / (200 * H_lambda_238(5));
IC(49) = (H_fraction_238(6) * P_i_238) / (200 * H_lambda_238(6));
IC(50) = (H_fraction_238(7) * P_i_238) / (200 * H_lambda_238(7));
IC(51) = (H_fraction_238(8) * P_i_238) / (200 * H_lambda_238(8));
IC(52) = (H_fraction_238(9) * P_i_238) / (200 * H_lambda_238(9));
IC(53) = (H_fraction_238(10) * P_i_238) / (200 * H_lambda_238(10));
IC(54) = (H_fraction_238(11) * P_i_238) / (200 * H_lambda_238(11));
IC(55) = (H_fraction_238(12) * P_i_238) / (200 * H_lambda_238(12));
IC(56) = (H_fraction_238(13) * P_i_238) / (200 * H_lambda_238(13));
IC(57) = (H_fraction_238(14) * P_i_238) / (200 * H_lambda_238(14));
IC(58) = (H_fraction_238(15) * P_i_238) / (200 * H_lambda_238(15));
IC(59) = (H_fraction_238(16) * P_i_238) / (200 * H_lambda_238(16));
IC(60) = (H_fraction_238(17) * P_i_238) / (200 * H_lambda_238(17));
IC(61) = (H_fraction_238(18) * P_i_238) / (200 * H_lambda_238(18));
IC(62) = (H_fraction_238(19) * P_i_238) / (200 * H_lambda_238(19));
IC(63) = (H_fraction_238(20) * P_i_238) / (200 * H_lambda_238(20));
IC(64) = (H_fraction_238(21) * P_i_238) / (200 * H_lambda_238(21));
IC(65) = (H_fraction_238(22) * P_i_238) / (200 * H_lambda_238(22));
IC(66) = (H_fraction_238(23) * P_i_238) / (200 * H_lambda_238(23));

IC(68) = (H_fraction_Pu239(1) * P_i_Pu239) / (200 * H_lambda_Pu239(1));
IC(69) = (H_fraction_Pu239(2) * P_i_Pu239) / (200 * H_lambda_Pu239(2));
IC(70) = (H_fraction_Pu239(3) * P_i_Pu239) / (200 * H_lambda_Pu239(3));
IC(71) = (H_fraction_Pu239(4) * P_i_Pu239) / (200 * H_lambda_Pu239(4));
IC(72) = (H_fraction_Pu239(5) * P_i_Pu239) / (200 * H_lambda_Pu239(5));
IC(73) = (H_fraction_Pu239(6) * P_i_Pu239) / (200 * H_lambda_Pu239(6));
IC(74) = (H_fraction_Pu239(7) * P_i_Pu239) / (200 * H_lambda_Pu239(7));
IC(75) = (H_fraction_Pu239(8) * P_i_Pu239) / (200 * H_lambda_Pu239(8));
IC(76) = (H_fraction_Pu239(9) * P_i_Pu239) / (200 * H_lambda_Pu239(9));
IC(77) = (H_fraction_Pu239(10) * P_i_Pu239) / (200 * H_lambda_Pu239(10));
IC(78) = (H_fraction_Pu239(11) * P_i_Pu239) / (200 * H_lambda_Pu239(11));
IC(79) = (H_fraction_Pu239(12) * P_i_Pu239) / (200 * H_lambda_Pu239(12));
IC(80) = (H_fraction_Pu239(13) * P_i_Pu239) / (200 * H_lambda_Pu239(13));
IC(81) = (H_fraction_Pu239(14) * P_i_Pu239) / (200 * H_lambda_Pu239(14));
IC(82) = (H_fraction_Pu239(15) * P_i_Pu239) / (200 * H_lambda_Pu239(15));
IC(83) = (H_fraction_Pu239(16) * P_i_Pu239) / (200 * H_lambda_Pu239(16));
IC(84) = (H_fraction_Pu239(17) * P_i_Pu239) / (200 * H_lambda_Pu239(17));
IC(85) = (H_fraction_Pu239(18) * P_i_Pu239) / (200 * H_lambda_Pu239(18));
IC(86) = (H_fraction_Pu239(19) * P_i_Pu239) / (200 * H_lambda_Pu239(19));
IC(87) = (H_fraction_Pu239(20) * P_i_Pu239) / (200 * H_lambda_Pu239(20));
IC(88) = (H_fraction_Pu239(21) * P_i_Pu239) / (200 * H_lambda_Pu239(21));
IC(89) = (H_fraction_Pu239(22) * P_i_Pu239) / (200 * H_lambda_Pu239(22));
IC(90) = (H_fraction_Pu239(23) * P_i_Pu239) / (200 * H_lambda_Pu239(23));

display(' Building Model Specific Reactivity IC... ');

% Rod reactivity
if sum(strcmp(Model_type, {'Rod Withdraw'}))
    % Start
    IC(67) = 0;
    IC(11) = -(alpha_fuel_T * Tinit_fuel + alpha_mod * Tinit_mod );

elseif ~sum(strcmp(Model_type, {'Rod Withdraw'}))
    IC(67) = Max_reactivity * Beff_mix;
    IC(11) = -(alpha_fuel_T * Tinit_fuel + alpha_mod * Tinit_mod ) ...
        + Max_reactivity * Beff_mix;

else
    IC(67) = 0;
    IC(11) = -(alpha_fuel_T * Tinit_fuel + alpha_mod * Tinit_mod );
```

```
end

% Temperature expansion
IC(91) = Tinit_fuel;
IC(92) = Tinit_mod;
IC(93) = Tinit_fuel;
IC(94) = Tinit_mod;

% Initial coolant velocity (m/s) [<POAH]
IC(95) = 0;

% Gas / SS expansion
IC(95) = Tinit_mod;
IC(96) = Tinit_mod;
IC(97) = Tinit_mod;
IC(98) = Tinit_mod;
IC(99) = Tinit_mod;
IC(100) = Tinit_mod;
IC(101) = Tinit_mod;

%% Solve for time dependent ODE
% Always run at least one
display(' Beginning ODE calculations... ');

% Establish arrays for saving
Y_out_L = [];
T_out_L = [];
t_out_calc = [];
IC_loop(1,:) = IC;
rho_event = [];
Tm_out_L = [];

% Divide into time for display output
N = 10000;

% Build the option set for the ODE
options = odeset('RelTol', 1e-9,'AbsTol',1e-9);

% Loop the solution of the ODE top allow dumping of array values needed for
% solution but not the output
for i = 1:N
    display(sprintf(' \nBeginning Run %d of %d', i, N));

    IC_loop(i,:);
    [t_out_loop, Y_out_loop] = ode113(@in_hour_FUN_6_2_0, [(i-1)*t_f/N i*t_f/N],
IC_loop(i,:), options);

    Y_out_L = [ Y_out_L Y_out_loop(:,1)' ];
    T_out_L = [ T_out_L Y_out_loop(:,12)' ];
    t_out_calc = [ t_out_calc t_out_loop' ];
    rho_event = [rho_event Y_out_loop(:,67)' ];
    Tm_out_L = [ Tm_out_L Y_out_loop(:,13)' ];

    IC_loop(i+1,:) = Y_out_loop(end,:);

    if ~isreal(Y_out_loop)
        Y_out_loop
        return;
    end

    clear Y_out_loop t_out_loop;

end

%% Output
display(' Building Output Displays... ');

% Find output items of concern
% Find the power out in MW
P_out_temp = 1e-6 * Y_out_L * velocity_neutron * (200e6 * 1.602677e-19) * volume_core ...
```

```
    * ( N_235 * sig_f_235 + N_238 * sig_f_238 + N_Pu_239 * sig_f_Pu_239 );

% Find the maximum power (MW) and time it occurs (s)
[ max_P, max_P_t_i] = max(P_out_temp);

% peaking factors based on power, this accounts for a higher pin temperature in the B
ring
% vs the average core temperature from the calculations (unit less)
% y = 2.4338E-19x3 - 5.3307E-13x2 + 4.1352E-07x + 1.0094E+00
peaking_factor_12_1 = 2.433e-19 .* P_out_temp.^3 - 5.3307e-13 .* P_out_temp.^2 ...
    + 4.1352e-7 .* P_out_temp + 1.0094;

% Peak core temperatures corrected (C)
T_out_temp = peaking_factor_12_1 .* T_out_L;

% Maximum core average temperature (C) and time it occurs (s)
[ max_T, max_T_t_i ] = max(T_out_temp);

max_P_t = t_out_calc(max_P_t_i);

max_T_t = t_out_calc(max_T_t_i);

% In order to save memory, make subsets of the gigantic arrays.  This shows a trend
% but saves space in the calling function of this function
Np = floor(linspace(1,length(t_out_calc), 10000));

P_out = P_out_temp(Np);

T_out = T_out_temp(Np);

t_out = t_out_calc(Np);

rho_out = rho_event(Np);

Tm_out = Tm_out_L(Np);

toc
```

ODE Function File 6.2.0

```
function [ R_dot ] = in_hour_FUN_6_2_0(t, R)
%% In hour equation function file
% The University of Texas at Austin (UT)
% Author: Greg Kline
% Date: 7/3/2015
% Revision: 6.2.0
%
% Modelling of a pulse insertion as well as a rod withdraw event using the
% information in Simnad, 1980 and Johnson, Lucas Tsvetkov, 2010
%
% The purpose of this code is to simulate both a pulse event and a continuous
% rod withdraw event.
%
% The parameters are modelled using the University of Texas parameters
%
% Revisions
% 5.1.0
%     - Added Plutonium 239 to mixture
%     - Accounted for core fuel burnup
%
% 5.2.0
%     - Added changes in moderator temperature based on temperature dependent
%         properties
%
% 5.3.0
%     - Added changing h based on channel properties
%
% 6.0.0
%    - Segmented fuel pin temperature regions and moderator regions
%
```

```
% 6.1.0
%    - Added gas and SS interactions
%
% 6.2.0
%    - Finalized state dependencies
%
%ODE Variables
% R(1) - neutron density
% R(2) - group 1 concentration
% R(3) - group 2 concentration
% R(4) - group 3 concentration
% R(5) - group 4 concentration
% R(6) - group 5 concentration
% R(7) - group 6 concentration
% R(8) - AmBe source concentration
% R(9) - spontaneous U238 concentration
% R(10) - spontaneous U235 concentration
% R(11) - reactivity at t
% R(12) - temperature of the fuel at t
% R(13) - temperature of the moderator at t
% R(14) - B_eff_mix at t
% R(15) - B_mix_1 at t
% R(16) - B_mix_2 at t
% R(17) - B_mix_3 at t
% R(18) - B_mix_4 at t
% R(19) - B_mix_5 at t
% R(20) - B_mix_6 at t
% R(21) - H_U2325_group_1 decay heat matrix
% R(22) - H_U2325_group_2
% R(23) - H_U2325_group_3
% R(24) - H_U2325_group_4
% R(25) - H_U2325_group_5
% R(26) - H_U2325_group_6
% R(27) - H_U2325_group_7
% R(28) - H_U2325_group_8
% R(29) - H_U2325_group_9
% R(30) - H_U2325_group_10
% R(31) - H_U2325_group_11
% R(32) - H_U2325_group_12
% R(33) - H_U2325_group_13
% R(34) - H_U2325_group_14
% R(35) - H_U2325_group_15
% R(36) - H_U2325_group_16
% R(37) - H_U2325_group_17
% R(38) - H_U2325_group_18
% R(39) - H_U2325_group_19
% R(40) - H_U2325_group_20
% R(41) - H_U2325_group_21
% R(42) - H_U2325_group_22
% R(43) - H_U2325_group_23
% R(44) - H_U2328_group_1
% R(45) - H_U2328_group_2
% R(46) - H_U2328_group_3
% R(47) - H_U2328_group_4
% R(48) - H_U2328_group_5
% R(49) - H_U2328_group_6
% R(50) - H_U2328_group_7
% R(51) - H_U2328_group_8
% R(52) - H_U2328_group_9
% R(53) - H_U2328_group_10
% R(54) - H_U2328_group_11
% R(55) - H_U2328_group_12
% R(56) - H_U2328_group_13
% R(57) - H_U2328_group_14
% R(58) - H_U2328_group_15
% R(59) - H_U2328_group_16
% R(60) - H_U2328_group_17
% R(61) - H_U2328_group_18
% R(62) - H_U2328_group_19
% R(63) - H_U2328_group_20
% R(64) - H_U2328_group_21
```

```
% R(65) - H_U2328_group_22
% R(66) - H_U2328_group_23
% R(67) - control rod reactivity at t
% R(68-90) - Pu239
% R(91) - Lower graphite temp
% R(92) - Lower moderator temp
% R(93) - Upper graphite temp
% R(94) - Upper mod temp
% R(95) - velocity
% R(96) - Lower Gas region
% R(97) - Mid Gas region
% R(98) - Upper Gas region
% R(99) - Lower SS Region
% R(100) - Mid SS region
% R(101) - Upper SS region

%% Constants
%%% global variables %%
global Tinit_fuel Tinit_mod  Max_reactivity Reactivity_add_rate
global  N_238 N_235 surface_area_fuel volume_core pin_height
global mass_U_235 mass_U_238 mass_fuel_mix  Event_type density_fuel
global N_Pu_239 mass_Pu_239 l_star  alpha_mod volume_cooling_core
global Area_pin_internal dz_cond_fuel mass_graphite_half_core
global surface_area_graphite area_cooling_pin graphite_height number_pins
global volume_cooling_graphite_core dz_cond_grap radius_pin inner_radius

%% neutronics constants %%%

% Group Lambdas
% U235
U235_L_1 = .0127;
U235_L_2 = .0317;
U235_L_3 = .115;
U235_L_4 = .311;
U235_L_5 = 1.40;
U235_L_6 = 3.87;

% U238
U238_L_1 = .0132;
U238_L_2 = .0321;
U238_L_3 = .139;
U238_L_4 = .358;
U238_L_5 = 1.41;
U238_L_6 = 4.02;

% Pu239
Pu239_L_1 = .0129;
Pu239_L_2 = .0313;
Pu239_L_3 = .135;
Pu239_L_4 = .333;
Pu239_L_5 = 1.36;
Pu239_L_6 = 4.04;

% total delayed neutron fraction U238 [Weaver, 1968]
B_238 = .0157;
% group i fraction is B * relative abundance
B_1_238 = B_238 * .013; %group 1
B_2_238 = B_238 * .137;
B_3_238 = B_238 * .162;
B_4_238 = B_238 * .388;
B_5_238 = B_238 * .225;
B_6_238 = B_238 * .075;

% total delayed neutron fraction U235 [Weaver, 1968]
B_235 = .0065;
B_1_235 = B_235 * .038;
B_2_235 = B_235 * .213;
B_3_235 = B_235 * .188;
B_4_235 = B_235 * .407;
B_5_235 = B_235 * .128;
B_6_235 = B_235 * .026;
```

```
% total delayed neutron fraction Pu239
B_Pu_239 = .0026;
B_1_Pu_239 = B_Pu_239 * .038;
B_2_Pu_239 = B_Pu_239 * .280;
B_3_Pu_239 = B_Pu_239 * .216;
B_4_Pu_239 = B_Pu_239 * .328;
B_5_Pu_239 = B_Pu_239 * .103;
B_6_Pu_239 = B_Pu_239 * .035;

% define fission cross sections for precursor ratios
%fission cross section (m^2) [NIST]
sig_f_238 = 3.3e-28;

%fission cross section (m^2)
sig_f_235 = 584.4e-28;

% fission cross section (m^2)
sig_f_Pu_239 = 747.4e-28;

% Boltzmann constant (m2 kg s-2 K-1)
k_b = 1.38064852e-23;

% Mass of neutron (kg)
m_nu = 1.674927471e-27;

% neutron velocity (m/s)
 velocity_neutron = sqrt( 2 * k_b * (R(12) + 273.15) / m_nu );
% velocity_neutron = 2197;

% U235 Beta factor (atoms/s)
Beta_factor_U235 = velocity_neutron * sig_f_235 * N_235;

% U238 Beta factor (atoms/s)
Beta_factor_U238 = velocity_neutron * sig_f_238 * N_238;

% Pu239 Beta factor (atoms/s)
Beta_factor_Pu239 = velocity_neutron * sig_f_Pu_239 * N_Pu_239;

% Beta of the DNP mix
B_1_mix = ( Beta_factor_U238 * B_1_238 + Beta_factor_U235 * B_1_235 ...
    + Beta_factor_Pu239 * B_1_Pu_239) ...
    / ( Beta_factor_U238 + Beta_factor_U235 + Beta_factor_Pu239);
B_2_mix = ( Beta_factor_U238 * B_2_238 + Beta_factor_U235 * B_2_235 ...
    + Beta_factor_Pu239 * B_2_Pu_239) ...
    / ( Beta_factor_U238 + Beta_factor_U235 + Beta_factor_Pu239);
B_3_mix = ( Beta_factor_U238 * B_3_238 + Beta_factor_U235 * B_3_235 ...
    + Beta_factor_Pu239 * B_3_Pu_239) ...
    / ( Beta_factor_U238 + Beta_factor_U235 + Beta_factor_Pu239);
B_4_mix = ( Beta_factor_U238 * B_4_238 + Beta_factor_U235 * B_4_235 ...
    + Beta_factor_Pu239 * B_4_Pu_239) ...
    / ( Beta_factor_U238 + Beta_factor_U235 + Beta_factor_Pu239);
B_5_mix = ( Beta_factor_U238 * B_5_238 + Beta_factor_U235 * B_5_235 ...
    + Beta_factor_Pu239 * B_5_Pu_239) ...
    / ( Beta_factor_U238 + Beta_factor_U235 + Beta_factor_Pu239);
B_6_mix = ( Beta_factor_U238 * B_6_238 + Beta_factor_U235 * B_6_235 ...
    + Beta_factor_Pu239 * B_6_Pu_239) ...
    / ( Beta_factor_U238 + Beta_factor_U235 + Beta_factor_Pu239);

% Lambda of the DNP mix
lambda_1 = ( Beta_factor_U238 * U238_L_1 + Beta_factor_U235 * U235_L_1 ...
    + Beta_factor_Pu239 * Pu239_L_1) ...
    / ( Beta_factor_U238 + Beta_factor_U235 + Beta_factor_Pu239);
lambda_2 = ( Beta_factor_U238 * U238_L_2 + Beta_factor_U235 * U235_L_2 ...
    + Beta_factor_Pu239 * Pu239_L_2) ...
    / ( Beta_factor_U238 + Beta_factor_U235 + Beta_factor_Pu239);
lambda_3 = ( Beta_factor_U238 * U238_L_3 + Beta_factor_U235 * U235_L_3 ...
    + Beta_factor_Pu239 * Pu239_L_3) ...
    / ( Beta_factor_U238 + Beta_factor_U235 + Beta_factor_Pu239);
lambda_4 = ( Beta_factor_U238 * U238_L_4 + Beta_factor_U235 * U235_L_4 ...
    + Beta_factor_Pu239 * Pu239_L_4) ...
```

```
    / ( Beta_factor_U238 + Beta_factor_U235 + Beta_factor_Pu239);
lambda_5 = ( Beta_factor_U238 * U238_L_5 + Beta_factor_U235 * U235_L_5 ...
    + Beta_factor_Pu239 * Pu239_L_5) ...
    / ( Beta_factor_U238 + Beta_factor_U235 + Beta_factor_Pu239);
lambda_6 = ( Beta_factor_U238 * U238_L_6 + Beta_factor_U235 * U235_L_6 ...
    + Beta_factor_Pu239 * Pu239_L_6) ...
    / ( Beta_factor_U238 + Beta_factor_U235 + Beta_factor_Pu239);

% Beta mix
% Beff correction factor based on Beff/B ratio for U235 .007/,0065
Beta_correction = 1.076923076923077;

% Beta mix
Beta_mix = B_1_mix + B_2_mix + B_3_mix + B_4_mix + B_5_mix + B_6_mix;

% Beff mix
Beff_mix = Beta_mix * Beta_correction;

%prompt neutron lifetime (s) [UT SAR]
% l_star = 51.9e-6;

% 2 Ci AmBe source (atoms/s/m^3)
AmBe_source = 7.4e10;

% U238 spontaneous fission (atoms/s/m^3)
U238_spontaneous = 1.80e-2 * mass_U_238 * 1000; % factor in (n/g-s)

% U235 spontaneous fission (atoms/s/m^3)
U235_spontaneous = 7.43e-4 * mass_U_235 * 1000; % factor in (n/g-s)

% Pu239 spontaneous fission (atoms/s/m^3)
Pu_239_spontaneuos = 2.30e-2 * mass_Pu_239 * 1000;

%% Decay Heat Constants %%%
% Decay (1/s)
% U235
H_lambda_235(1) = 2.2138e1;
H_lambda_235(2) = 5.1587e-1;
H_lambda_235(3) = 1.9594e-1;
H_lambda_235(4) = 1.0314e-1;
H_lambda_235(5) = 3.3656e-2;
H_lambda_235(6) = 1.1681e-2;
H_lambda_235(7) = 3.5870e-3;
H_lambda_235(8) = 1.3930e-3;
H_lambda_235(9) = 6.2630e-4;
H_lambda_235(10) = 1.8906e-4;
H_lambda_235(11) = 5.4988e-5;
H_lambda_235(12) = 2.0958e-5;
H_lambda_235(13) = 1.0010e-5;
H_lambda_235(14) = 2.5438e-6;
H_lambda_235(15) = 6.6361e-7;
H_lambda_235(16) = 1.2290e-7;
H_lambda_235(17) = 2.7213e-8;
H_lambda_235(18) = 4.3714e-9;
H_lambda_235(19) = 7.5780e-10;
H_lambda_235(20) = 2.4786e-10;
H_lambda_235(21) = 2.2384e-13;
H_lambda_235(22) = 2.4600e-14;
H_lambda_235(23) = 1.5699e-14;

% U238
H_lambda_238(1) = 3.2881e00;
H_lambda_238(2) = 9.3805e-1;
H_lambda_238(3) = 3.7073e-1;
H_lambda_238(4) = 1.1118e-1;
H_lambda_238(5) = 3.6143e-2;
H_lambda_238(6) = 1.3272e-2;
H_lambda_238(7) = 5.0133e-3;
H_lambda_238(8) = 1.3655e-3;
H_lambda_238(9) = 5.5158e-4;
H_lambda_238(10) = 1.7873e-4;
```

```
H_lambda_238(11) = 4.9032e-5;
H_lambda_238(12) = 1.7058e-5;
H_lambda_238(13) = 7.0465e-6;
H_lambda_238(14) = 2.3190e-6;
H_lambda_238(15) = 6.4480e-7;
H_lambda_238(16) = 1.2649e-7;
H_lambda_238(17) = 2.5548e-8;
H_lambda_238(18) = 8.4782e-9;
H_lambda_238(19) = 7.5130e-10;
H_lambda_238(20) = 2.4188e-10;
H_lambda_238(21) = 2.2739e-13;
H_lambda_238(22) = 9.0536e-14;
H_lambda_238(23) = 5.6098e-15;

% Pu239
H_lambda_Pu239(1) = 1.0020e1;
H_lambda_Pu239(2) = 6.4330e-1;
H_lambda_Pu239(3) = 2.1860e-1;
H_lambda_Pu239(4) = 1.0040e-1;
H_lambda_Pu239(5) = 3.7280e-2;
H_lambda_Pu239(6) = 1.4350e-2;
H_lambda_Pu239(7) = 4.5490e-3;
H_lambda_Pu239(8) = 1.3280e-3;
H_lambda_Pu239(9) = 5.3560e-4;
H_lambda_Pu239(10) = 1.7300e-4;
H_lambda_Pu239(11) = 4.8810e-5;
H_lambda_Pu239(12) = 2.0060e-5;
H_lambda_Pu239(13) = 8.3190e-6;
H_lambda_Pu239(14) = 2.3580e-6;
H_lambda_Pu239(15) = 6.4500e-7;
H_lambda_Pu239(16) = 1.2780e-7;
H_lambda_Pu239(17) = 2.4660e-8;
H_lambda_Pu239(18) = 9.3780e-9;
H_lambda_Pu239(19) = 7.4500e-10;
H_lambda_Pu239(20) = 2.4260e-10;
H_lambda_Pu239(21) = 2.2100e-13;
H_lambda_Pu239(22) = 2.6400e-14;
H_lambda_Pu239(23) = 1.3800e-14;

% Power fraction
% U235
H_fraction_235(1) = 6.5057e-1;
H_fraction_235(2) = 5.1264e-1;
H_fraction_235(3) = 2.4384e-1;
H_fraction_235(4) = 1.3850e-1;
H_fraction_235(5) = 5.5440e-2;
H_fraction_235(6) = 2.2225e-2;
H_fraction_235(7) = 3.3088e-3;
H_fraction_235(8) = 9.3015e-4;
H_fraction_235(9) = 8.0943e-4;
H_fraction_235(10) = 1.9567e-4;
H_fraction_235(11) = 3.2535e-5;
H_fraction_235(12) = 7.5595e-6;
H_fraction_235(13) = 2.5232e-6;
H_fraction_235(14) = 4.9948e-7;
H_fraction_235(15) = 1.8531e-7;
H_fraction_235(16) = 2.6608e-8;
H_fraction_235(17) = 2.2398e-9;
H_fraction_235(18) = 8.1641e-12;
H_fraction_235(19) = 8.7797e-11;
H_fraction_235(20) = 2.5131e-14;
H_fraction_235(21) = 3.2176e-16;
H_fraction_235(22) = 4.5038e-17;
H_fraction_235(23) = 7.4791e-17;

% U238
H_fraction_238(1) = 1.2311e00;
H_fraction_238(2) = 1.1486e00;
H_fraction_238(3) = 7.0701e-1;
H_fraction_238(4) = 2.5209e-1;
H_fraction_238(5) = 7.1870e-2;
```

```
H_fraction_238(6) = 2.8291e-2;
H_fraction_238(7) = 6.8382e-3;
H_fraction_238(8) = 1.2322e-3;
H_fraction_238(9) = 6.8409e-4;
H_fraction_238(10) = 1.6975e-4;
H_fraction_238(11) = 2.4182e-5;
H_fraction_238(12) = 6.6356e-6;
H_fraction_238(13) = 1.0075e-6;
H_fraction_238(14) = 4.9894e-7;
H_fraction_238(15) = 1.6352e-7;
H_fraction_238(16) = 2.3355e-8;
H_fraction_238(17) = 2.8094e-9;
H_fraction_238(18) = 3.6236e-11;
H_fraction_238(19) = 6.4577e-11;
H_fraction_238(20) = 4.4963e-14;
H_fraction_238(21) = 3.6654e-16;
H_fraction_238(22) = 5.6293e-17;
H_fraction_238(23) = 7.1602e-17;

% Pu239
H_fraction_Pu239(1) = 2.0830e-1;
H_fraction_Pu239(2) = 3.8530e-1;
H_fraction_Pu239(3) = 2.2130e-1;
H_fraction_Pu239(4) = 9.4600e-2;
H_fraction_Pu239(5) = 3.5310e-2;
H_fraction_Pu239(6) = 2.2920e-2;
H_fraction_Pu239(7) = 3.9460e-3;
H_fraction_Pu239(8) = 1.3170e-3;
H_fraction_Pu239(9) = 7.0520e-4;
H_fraction_Pu239(10) = 1.4320e-4;
H_fraction_Pu239(11) = 1.7650e-5;
H_fraction_Pu239(12) = 7.3470e-6;
H_fraction_Pu239(13) = 1.7470e-6;
H_fraction_Pu239(14) = 5.4810e-7;
H_fraction_Pu239(15) = 1.6710e-7;
H_fraction_Pu239(16) = 2.1120e-8;
H_fraction_Pu239(17) = 2.9960e-9;
H_fraction_Pu239(18) = 5.1070e-11;
H_fraction_Pu239(19) = 5.7300e-11;
H_fraction_Pu239(20) = 4.1380e-14;
H_fraction_Pu239(21) = 1.0880e-15;
H_fraction_Pu239(22) = 2.4540e-17;
H_fraction_Pu239(23) = 7.5570e-17;

%% State dependent functions
% ALPHA T
% Temperature dependent alpha T of fuel from UT TRIGA Lab experiment
% dp/dT = -8.14475e-8 * T^2 - 2.53543e-5 * T - 1.56396e-4 ->
% alpha_t_T = -1.62895e-7 * T - 2.53543e-5
% alpha_fuel_T = -1.62895e-7 * R(12) - 2.53543e-5
% alpha_fuel_T = -1.65149e-7 * R(12) - 2.5056e-5;
% alpha_fuel_T = -6.9235e-5;
% alpha_fuel_T = -5.6196e-5;

% constant Alpha_T from SAR
% alpha_fuel_T = -1e-4;

% Curve fits to GA chart from SAR
% 6th order poly
% alpha_fuel_T = -1.9631e-21*R(12)^6 + 6.5610e-18*R(12)^5 - 8.210e-15*R(12)^4 ...
%    + 4.5051e-12*R(12)^3 - 7.8572e-10*R(12)^2 - 1.1061e-7*R(12) - 7.4965e-5;
% 4th order poly
  alpha_fuel_T = 1.4990e-16*R(12)^4 - 5.3734e-13*R(12)^3 + 6.5947e-10*R(12)^2 ...
    - 2.8159e-7*R(12) - 6.9571e-5;
% 2nd order poly
% alpha_fuel_T = 3.1802e-10*R(12)^2 - 2.1800e-7*R(12) - 7.2126e-5;

% volumetric heat capacity from Simnad (J/ m3 K)
cp_fuel_vol = (2.04 + 4.17e-3 * R(12) ) * 1e6;

% Convert to specific heat ( J / kg K )
```

```
cp_fuel = cp_fuel_vol / density_fuel;

% find keff
current_keff = 1 / ( 1 - ( R(11) + (alpha_fuel_T * R(12) + alpha_mod * R(13) ) ) );

% keff adjusted neutron lifetime (s)
 A = l_star / current_keff;
% A = l_star;

% Instantaneous Power (W)
% U235 contribution
P_inst_235 = R(1) * velocity_neutron * (200e6 * 1.602677e-19) * N_235 ...
    * sig_f_235 * volume_core;

% U238 contribution (W)
P_inst_238 = R(1) * velocity_neutron * (200e6 * 1.602677e-19) * N_238 ...
    * sig_f_238 * volume_core;

% Pu239 contribution (W)
P_inst_Pu239 = R(1) * velocity_neutron * (200e6 * 1.602677e-19) * N_Pu_239 ...
    * sig_f_Pu_239 * volume_core;

% Total instantaneous (W)
P_inst = P_inst_235 + P_inst_238 + P_inst_Pu239;

% specific heat capacity of graphite (cal / g C) [Entegris, inc.]
cp_graphite_cal_91 = .10795e8 * R(91)^-3 - .61257e5 * R(91)^-2 ...
    + .30795e-4 * R(91) + .44391;
cp_graphite_91 = 4183.995381 * cp_graphite_cal_91; %Conversion to J / kg K

cp_graphite_cal_93 = .10795e8 * R(93)^-3 - .61257e5 * R(93)^-2 ...
    + .30795e-4 * R(93) + .44391;
cp_graphite_93 = 4183.995381 * cp_graphite_cal_93;

%% Moderator state equations

% Temperature Dependent Density (kg/m^3) [ddbst.de, 273K-648K]
A_cp = .14395;
B_cp = .0112;
C_cp = 649.727;
D_cp = .05107;
low_temp = R(94);
c_temp = R(101);
f_temp = R(12);

% density of outlet water/core area water (kg/m^3)
density_water_13 = A_cp / ( B_cp^(1 + (1 - (R(13) + 273.15)/C_cp)^D_cp));
density_water_92 = A_cp / ( B_cp^(1 + (1 - (R(92) + 273.15)/C_cp)^D_cp));
density_water_94 = A_cp / ( B_cp^(1 + (1 - (R(94) + 273.15)/C_cp)^D_cp));

% density of inlet water, bulk tank (kg/m^3)
density_water_0 =  A_cp / ( B_cp^(1 + (1 - (Tinit_mod + 273.15)/C_cp)^D_cp));

% Mass of water (kg)
mass_cooling_water_fuel = volume_cooling_core * density_water_13;
mass_cooling_water_graphite_92 = volume_cooling_graphite_core * density_water_92;
mass_cooling_water_graphite_94 = volume_cooling_graphite_core * density_water_94;

% specific heat of inlet water (KJ/kg K)
cp_H2O_KJ_0 = 3.16744e-10 * Tinit_mod^4 - 1.05772e-7 * Tinit_mod^3 ...
    + 2.35330e-5 * Tinit_mod^2 - 1.47670e-3 * Tinit_mod + 4.20617e0;

% Convert to J/kg
cp_H2O_0 = cp_H2O_KJ_0 * 1000;

% Temperature dependent Specific heat (J/kgK) [steam tables]
cp_H2O_1 = ( 3.16744e-10 * R(92)^4 - 1.05772e-7 * R(92)^3 ...
    + 2.35330e-5 * R(92)^2 - 1.47670e-3 * R(92) + 4.20617e0 ) * 1000;

cp_H2O_2 = ( 3.16744e-10 * R(13)^4 - 1.05772e-7 * R(13)^3 ...
    + 2.35330e-5 * R(13)^2 - 1.47670e-3 * R(13) + 4.20617e0 ) * 1000;
```

```
cp_H2O_3 = ( 3.16744e-10 * R(94)^4 - 1.05772e-7 * R(94)^3 ...
    + 2.35330e-5 * R(94)^2 - 1.47670e-3 * R(94) + 4.20617e0 ) * 1000;

% Use the Plume formulas from Dartmouth Paper/UT LOCA to find current mass flow
% gravity (m/s^2)
gravity = 9.8066;

% Prandtl Number (Pr)
Pr_water = 1.76;

% Kinetic Viscosity (m^2/s)
kinetic_viscosity_water = .279e-6;

% Expansion Coefficient (1/K)
B_water = .207e-3;

% Thermal conductivity [NIST](W/mK)
k_water = -1.48445 + 4.12292 * ((R(13) + 273.15)/298.15) ...
    - 1.63866 * ((R(13) + 273.15)/298.15)^2;
k_fuel = 17.5730;  % [Simnad]
k_graphite = 112.4;

% Gradient Prandtl (gPr)
gPr_water = (.75 * Pr_water)^.5 / (.609 + 1.221 * Pr_water^.5 + 1.238 * Pr_water)^.25;

% Thermal diffusivity (m^2/s)
% therm_diff_water = 0.143e-6;
therm_diff_water_13 = k_water / ( density_water_13 * cp_H2O_2 );
therm_diff_water_92 = k_water / ( density_water_92 * cp_H2O_1 );
therm_diff_water_94 = k_water / ( density_water_94 * cp_H2O_3 );

% find this loops transient factor (m(T) * cp(T)) [J/K, J/C]
trans_factor_13 = mass_cooling_water_fuel * cp_H2O_2;
trans_factor_92 = mass_cooling_water_graphite_92 * cp_H2O_1;
trans_factor_94 = mass_cooling_water_graphite_94 * cp_H2O_3;

% find the mass flow rate of inlet (kg/s) rho(T) * A_opening * velocity
m_dot = R(95) * density_water_13 * area_cooling_pin * number_pins;

% Enthalpy (J/kgK)
% find saturation temperature for depth of 7.5m (C)
T_sat_water = 114.79;

% find current specific enthalpy h = cp * (T(t) - T_ref) + h_ref;
enthalpy_ref = 9007; % J/kg @ 0C saturated

h_0 = cp_H2O_0 * Tinit_mod + enthalpy_ref;

h_1 = cp_H2O_1 * R(92) + enthalpy_ref;

h_2 = cp_H2O_2 * R(13) + enthalpy_ref;

h_3 = cp_H2O_3 * R(94) + enthalpy_ref;

%% Gas and SS expansion coefficients [ UT LOCA]
% Mass (kg)
mass_gas_fuel = .08375 * pi * ( (radius_pin - .000508)^2 - (radius_pin - .000508 -
1.310588235294116e-04)^2 ) * pin_height;
mass_gas_gr = .08375 * pi * ( (radius_pin - .000508)^2 - (radius_pin - .000508 -
1.310588235294116e-04)^2 ) * graphite_height;
mass_clad_fuel = 7740 * pi * ( radius_pin^2 - (radius_pin - .000508)^2 ) * pin_height;
mass_clad_gr = 7740 * pi * ( radius_pin^2 - (radius_pin - .000508)^2 ) * graphite_height;

% cp (J/kg K)
cp_gas = 14.53e3;
cp_clad = 500;

% dz for conduction (m)
inner_radius = 0.003175;
dz_gas = 1.310588235294116e-04/2;
```

```
dz_clad = 0.000508/2;
dz_fuel_unit = ( radius_pin - .000508 - 1.310588235294116e-04 - inner_radius ) /2;

% Gas heat transfer from KSU SAR (W/m^2 K) [Whaley]
h_gas = 2.84e3;

% Thermal conductivity (W/m K)
k_gas = h_gas * 1.310588235294116e-04; % k ~ h/dx
k_clad = 16.2;

% Area (m^2)
A_gas = pi * ( (radius_pin - .000508)^2 - (radius_pin - .000508 - 1.310588235294116e-
04)^2 );
A_clad = pi * ( (radius_pin )^2 - (radius_pin - .000508)^2 );

% Surface areas (m^2)
surface_gas_out_grap = 2 * pi * (radius_pin - .000508) * graphite_height;
surface_gas_out_fuel = 2 * pi * (radius_pin - .000508) * pin_height;
surface_gas_in_grap = 2 * pi * (radius_pin - .000508 - 1.310588235294116e-04) *
graphite_height;
surface_gas_in_fuel = 2 * pi * (radius_pin - .000508 - 1.310588235294116e-04) *
pin_height;

%% heat transfer coefficient
% Grashof Number [Clarkson.edu]
% Gr_s = reduced_gravity * pin_height^3 / kinetic_viscosity_water^2

% Find the Ra_s [ Kaminski ]
% Ra_s = Gr_s * Pr_water;
Ra_L_1 =  ( gravity * B_water ) / ( kinetic_viscosity_water * therm_diff_water_13 ) ...
    * abs( R(99) - R(92) ) * graphite_height^3;
Ra_L_2 =  ( gravity * B_water ) / ( kinetic_viscosity_water * therm_diff_water_92 ) ...
    * abs( R(100) - R(13) ) * pin_height^3;
Ra_L_3 =  ( gravity * B_water ) / ( kinetic_viscosity_water * therm_diff_water_94 ) ...
    * abs( R(101) - R(94) ) * graphite_height^3;

% Nusselt number
% external free convection vertical cylinder
Nu_L_1 = ( .825 + ( .387 * Ra_L_1^(1/6) ) / ( 1 + ( .492/Pr_water)^(9/16) )^(8/27))^2;
%penn.edu
Nu_L_2 = ( .825 + ( .387 * Ra_L_2^(1/6) ) / ( 1 + ( .492/Pr_water)^(9/16) )^(8/27))^2;
%penn.edu
Nu_L_3 = ( .825 + ( .387 * Ra_L_3^(1/6) ) / ( 1 + ( .492/Pr_water)^(9/16) )^(8/27))^2;
%penn.edu

% Heat transfer coefficient ( W/ m^2 K )
h_bar_1 = Nu_L_1 * k_water / graphite_height;
h_bar_2 = Nu_L_2 * k_water / pin_height;
h_bar_3 = Nu_L_3 * k_water / graphite_height;

%% Delayed Power State Function (W)
% U235 delayed (W)
P_d_235 =  ( P_inst_235 / 200 ) * sum( H_fraction_235 ./ H_lambda_235 );

% U238 delayed (W)
P_d_238 =  ( P_inst_238 / 200 ) * sum( H_fraction_238 ./ H_lambda_238 );

% Pu239 delayed (W)
P_d_Pu_239 = ( P_inst_Pu239 / 200 ) * sum( H_fraction_Pu239 ./ H_lambda_Pu239 );

P_delay = P_d_235 + P_d_238 + P_d_Pu_239;

%% Total power by isotope (W)
% U235
P_i_235 = P_inst_235 + P_d_235;

% U238
P_i_238 = P_inst_238 + P_d_238;

% Pu239
P_i_Pu239 = P_inst_Pu239 + P_d_Pu_239;
```

```
P_eff = P_inst - P_delay + sum(R(21:66));

%% Differentials
% Reactor Kinetics
% overall in hour
R_dot(1) = ( R(11) + alpha_fuel_T * ( R(12) - Tinit_fuel ) ...
    + alpha_mod * ( R(13) - Tinit_mod ) - Beff_mix ) / A * R(1) ...
    + lambda_1 * R(2) ...
    + lambda_2 * R(3) ...
    + lambda_3 * R(4) ...
    + lambda_4 * R(5) ...
    + lambda_5 * R(6) ...
    + lambda_6 * R(7) ...
    + AmBe_source ...
    + U238_spontaneous + U235_spontaneous + Pu_239_spontaneuos; % ...
    % - R(1) * velocity_neutron * N_Sm * sig_a_Sm;
    % - R(1) * velocity_neutron * N_Zr * sig_a_Zr;

% DNP groups
R_dot(2) = (B_1_mix / A) * R(1) - lambda_1 * R(2);
R_dot(3) = (B_2_mix / A) * R(1) - lambda_2 * R(3);
R_dot(4) = (B_3_mix / A) * R(1) - lambda_3 * R(4);
R_dot(5) = (B_4_mix / A) * R(1) - lambda_4 * R(5);
R_dot(6) = (B_5_mix / A) * R(1) - lambda_5 * R(6);
R_dot(7) = (B_6_mix / A) * R(1) - lambda_6 * R(7);

% Source strength
R_dot(8) = 0;
R_dot(9) = 0;
R_dot(10) = 0;

% Reactivities
% event reactivity
switch Event_type
    case 'Pulse'
        R_dot(11) = 0;

    case 'Rod Withdraw'
        if R(11) < Max_reactivity * Beff_mix
            R_dot(11) = Reactivity_add_rate;

        elseif R(11) >= Max_reactivity * Beff_mix
            R_dot(11) = 0;

        else
            display (' Error in Reactivity Addition ');
        end

    otherwise
        R_dot(11) = 0;
end

% Fuel Temperature
R_dot(12) = ( 1 / ( mass_fuel_mix * cp_fuel ) ) * ( P_eff ...
    + ( R(91) - R(12) ) / ( dz_cond_fuel/k_fuel + dz_cond_grap/k_graphite ) *
Area_pin_internal ...
    + ( R(93) - R(12) ) / ( dz_cond_fuel/k_fuel + dz_cond_grap/k_graphite ) *
Area_pin_internal ...
    + ( R(97) - R(12) ) / ( dz_fuel_unit/k_fuel + dz_gas/k_gas ) *  surface_gas_in_fuel
);

% Moderator Temperature
R_dot(13) = 1/trans_factor_13 * ( h_bar_2 * surface_area_fuel * (R(100) - R(13)) ...
        + m_dot * ( h_1 + gravity * graphite_height) ...
        - m_dot * ( h_2 + gravity * (graphite_height + pin_height) ) );

% Changes in Beta effective
% [ current model will be 0 ]
R_dot(14) = 0;
R_dot(15) = 0;
```

```
R_dot(16) = 0;
R_dot(17) = 0;
R_dot(18) = 0;
R_dot(19) = 0;
R_dot(20) = 0;

%% Decay Heat calculations
% Delayed heat from U235 antecedes
R_dot(21) = H_fraction_235(1) * P_i_235 / 200 - H_lambda_235(1) * R(21);
R_dot(22) = H_fraction_235(2) * P_i_235 / 200 - H_lambda_235(2) * R(22);
R_dot(23) = H_fraction_235(3) * P_i_235 / 200 - H_lambda_235(3) * R(23);
R_dot(24) = H_fraction_235(4) * P_i_235 / 200 - H_lambda_235(4) * R(24);
R_dot(25) = H_fraction_235(5) * P_i_235 / 200 - H_lambda_235(5) * R(25);
R_dot(26) = H_fraction_235(6) * P_i_235 / 200 - H_lambda_235(6) * R(26);
R_dot(27) = H_fraction_235(7) * P_i_235 / 200 - H_lambda_235(7) * R(27);
R_dot(28) = H_fraction_235(8) * P_i_235 / 200 - H_lambda_235(8) * R(28);
R_dot(29) = H_fraction_235(9) * P_i_235 / 200 - H_lambda_235(9) * R(29);
R_dot(30) = H_fraction_235(10) * P_i_235 / 200 - H_lambda_235(10) * R(30);
R_dot(31) = H_fraction_235(11) * P_i_235 / 200 - H_lambda_235(11) * R(31);
R_dot(32) = H_fraction_235(12) * P_i_235 / 200 - H_lambda_235(12) * R(32);
R_dot(33) = H_fraction_235(13) * P_i_235 / 200 - H_lambda_235(13) * R(33);
R_dot(34) = H_fraction_235(14) * P_i_235 / 200 - H_lambda_235(14) * R(34);
R_dot(35) = H_fraction_235(15) * P_i_235 / 200 - H_lambda_235(15) * R(35);
R_dot(36) = H_fraction_235(16) * P_i_235 / 200 - H_lambda_235(16) * R(36);
R_dot(37) = H_fraction_235(17) * P_i_235 / 200 - H_lambda_235(17) * R(37);
R_dot(38) = H_fraction_235(18) * P_i_235 / 200 - H_lambda_235(18) * R(38);
R_dot(39) = H_fraction_235(19) * P_i_235 / 200 - H_lambda_235(19) * R(39);
R_dot(40) = H_fraction_235(20) * P_i_235 / 200 - H_lambda_235(20) * R(40);
R_dot(41) = H_fraction_235(21) * P_i_235 / 200 - H_lambda_235(21) * R(41);
R_dot(42) = H_fraction_235(22) * P_i_235 / 200 - H_lambda_235(22) * R(42);
R_dot(43) = H_fraction_235(23) * P_i_235 / 200 - H_lambda_235(23) * R(43);

% Delayed heat from U238 antecedes
R_dot(44) = H_fraction_238(1) * P_i_238 / 200 - H_lambda_238(1) * R(44);
R_dot(45) = H_fraction_238(2) * P_i_238 / 200 - H_lambda_238(2) * R(45);
R_dot(46) = H_fraction_238(3) * P_i_238 / 200 - H_lambda_238(3) * R(46);
R_dot(47) = H_fraction_238(4) * P_i_238 / 200 - H_lambda_238(4) * R(47);
R_dot(48) = H_fraction_238(5) * P_i_238 / 200 - H_lambda_238(5) * R(48);
R_dot(49) = H_fraction_238(6) * P_i_238 / 200 - H_lambda_238(6) * R(49);
R_dot(50) = H_fraction_238(7) * P_i_238 / 200 - H_lambda_238(7) * R(50);
R_dot(51) = H_fraction_238(8) * P_i_238 / 200 - H_lambda_238(8) * R(51);
R_dot(52) = H_fraction_238(9) * P_i_238 / 200 - H_lambda_238(9) * R(52);
R_dot(53) = H_fraction_238(10) * P_i_238 / 200 - H_lambda_238(10) * R(53);
R_dot(54) = H_fraction_238(11) * P_i_238 / 200 - H_lambda_238(11) * R(54);
R_dot(55) = H_fraction_238(12) * P_i_238 / 200 - H_lambda_238(12) * R(55);
R_dot(56) = H_fraction_238(13) * P_i_238 / 200 - H_lambda_238(13) * R(56);
R_dot(57) = H_fraction_238(14) * P_i_238 / 200 - H_lambda_238(14) * R(57);
R_dot(58) = H_fraction_238(15) * P_i_238 / 200 - H_lambda_238(15) * R(58);
R_dot(59) = H_fraction_238(16) * P_i_238 / 200 - H_lambda_238(16) * R(59);
R_dot(60) = H_fraction_238(17) * P_i_238 / 200 - H_lambda_238(17) * R(60);
R_dot(61) = H_fraction_238(18) * P_i_238 / 200 - H_lambda_238(18) * R(61);
R_dot(62) = H_fraction_238(19) * P_i_238 / 200 - H_lambda_238(19) * R(62);
R_dot(63) = H_fraction_238(20) * P_i_238 / 200 - H_lambda_238(20) * R(63);
R_dot(64) = H_fraction_238(21) * P_i_238 / 200 - H_lambda_238(21) * R(64);
R_dot(65) = H_fraction_238(22) * P_i_238 / 200 - H_lambda_238(22) * R(65);
R_dot(66) = H_fraction_238(23) * P_i_238 / 200 - H_lambda_238(23) * R(66);

R_dot(68) = H_fraction_Pu239(1) * P_i_Pu239 / 200 - H_lambda_Pu239(1) * R(68);
R_dot(69) = H_fraction_Pu239(2) * P_i_Pu239 / 200 - H_lambda_Pu239(2) * R(69);
R_dot(70) = H_fraction_Pu239(3) * P_i_Pu239 / 200 - H_lambda_Pu239(3) * R(70);
R_dot(71) = H_fraction_Pu239(4) * P_i_Pu239 / 200 - H_lambda_Pu239(4) * R(71);
R_dot(72) = H_fraction_Pu239(5) * P_i_Pu239 / 200 - H_lambda_Pu239(5) * R(72);
R_dot(73) = H_fraction_Pu239(6) * P_i_Pu239 / 200 - H_lambda_Pu239(6) * R(73);
R_dot(74) = H_fraction_Pu239(7) * P_i_Pu239 / 200 - H_lambda_Pu239(7) * R(74);
R_dot(75) = H_fraction_Pu239(8) * P_i_Pu239 / 200 - H_lambda_Pu239(8) * R(75);
R_dot(76) = H_fraction_Pu239(9) * P_i_Pu239 / 200 - H_lambda_Pu239(9) * R(76);
R_dot(77) = H_fraction_Pu239(10) * P_i_Pu239 / 200 - H_lambda_Pu239(10) * R(77);
R_dot(78) = H_fraction_Pu239(11) * P_i_Pu239 / 200 - H_lambda_Pu239(11) * R(78);
R_dot(79) = H_fraction_Pu239(12) * P_i_Pu239 / 200 - H_lambda_Pu239(12) * R(79);
R_dot(80) = H_fraction_Pu239(13) * P_i_Pu239 / 200 - H_lambda_Pu239(13) * R(80);
R_dot(81) = H_fraction_Pu239(14) * P_i_Pu239 / 200 - H_lambda_Pu239(14) * R(81);
```

```
R_dot(82) = H_fraction_Pu239(15) * P_i_Pu239 / 200 - H_lambda_Pu239(15) * R(82);
R_dot(83) = H_fraction_Pu239(16) * P_i_Pu239 / 200 - H_lambda_Pu239(16) * R(83);
R_dot(84) = H_fraction_Pu239(17) * P_i_Pu239 / 200 - H_lambda_Pu239(17) * R(85);
R_dot(85) = H_fraction_Pu239(18) * P_i_Pu239 / 200 - H_lambda_Pu239(18) * R(85);
R_dot(86) = H_fraction_Pu239(19) * P_i_Pu239 / 200 - H_lambda_Pu239(19) * R(86);
R_dot(87) = H_fraction_Pu239(20) * P_i_Pu239 / 200 - H_lambda_Pu239(20) * R(87);
R_dot(88) = H_fraction_Pu239(21) * P_i_Pu239 / 200 - H_lambda_Pu239(21) * R(88);
R_dot(89) = H_fraction_Pu239(22) * P_i_Pu239 / 200 - H_lambda_Pu239(22) * R(89);
R_dot(90) = H_fraction_Pu239(23) * P_i_Pu239 / 200 - H_lambda_Pu239(23) * R(90);

%% Input reactivity
switch Event_type
    case 'Pulse'
        R_dot(67) = 0;

    case 'Rod Withdraw'
        if R(67) < Max_reactivity * Beff_mix
            R_dot(67) = Reactivity_add_rate;

        elseif R(67) >= Max_reactivity * Beff_mix
            R_dot(67) = 0;
        else
            display (' Error in Reactivity Addition ');
        end

    otherwise
        R_dot(67) = 0;
end

%% Temperature expansion
% Lower Graphite
R_dot(91) = ( 1 / ( mass_graphite_half_core * cp_graphite_91 ) ) ...
    * ( ( R(91) - R(12) ) / ( dz_cond_fuel/k_fuel + dz_cond_grap/k_graphite ) *
Area_pin_internal ...
        + ( R(96) - R(91) ) / ( dz_fuel_unit/k_fuel + dz_gas/k_gas ) *
surface_gas_in_grap );

% upper graphite
R_dot(93) = ( 1 / ( mass_graphite_half_core * cp_graphite_93 ) ) ...
    * ( ( R(93) - R(12) ) / ( dz_cond_fuel/k_fuel + dz_cond_grap/k_graphite ) *
Area_pin_internal ...
        + ( R(98) - R(93) ) / ( dz_fuel_unit/k_fuel + dz_gas/k_gas ) *
surface_gas_in_grap );

% Lower water channel
R_dot(92) = 1/trans_factor_92 * ( h_bar_1 * surface_area_graphite * (R(99) - R(92)) ...
        + m_dot * ( h_0 + gravity * ( 0 ) )...
        - m_dot * ( h_1 + gravity * (graphite_height) ) );

% Upper water channel
R_dot(94) = 1/trans_factor_94 * ( h_bar_3 * surface_area_graphite * (R(101) - R(94)) ...
        + m_dot * ( h_2 + gravity * (graphite_height + pin_height) )...
        - m_dot * ( h_3 + gravity * ( 2 * graphite_height + pin_height) ) );

% Velocity
R_dot(95) = sqrt( therm_diff_water_13 * abs( R(13) - R(94) ) * ( dz_cond_fuel +
dz_cond_grap)  );

%% Gas/ SS expansion
% Lower gas
R_dot(96) = 1/(mass_gas_gr * cp_gas) * ( ( R(97) - R(96) )* k_gas * A_gas / (
dz_cond_fuel + dz_cond_grap) ...
        + (  R(91) - R(96) ) / ( dz_fuel_unit/k_graphite + dz_gas/k_gas ) *
surface_gas_in_grap ...
        + (  R(99) - R(96) ) / ( dz_clad/k_clad + dz_gas/k_gas ) * surface_gas_out_grap
);

% Mid gas
R_dot(97) = 1/(mass_gas_fuel * cp_gas) * ( ( R(96) - R(97) )* k_gas * A_gas / (
dz_cond_fuel + dz_cond_grap) ...
```

```
        + (  R(12) - R(97) ) / ( dz_fuel_unit/k_fuel + dz_gas/k_gas ) *
surface_gas_in_fuel ...
        + (  R(100) - R(97) ) / ( dz_clad/k_clad + dz_gas/k_gas ) * surface_gas_out_fuel
...
        + (  R(98) - R(97) ) * k_gas * A_gas / ( dz_cond_fuel + dz_cond_grap) );

% Upper gas
R_dot(98) = 1/(mass_gas_gr * cp_gas) * ( ( R(97) - R(98) ) * k_gas * A_gas / (
dz_cond_fuel + dz_cond_grap) ...
        + (  R(93) - R(98) ) / ( dz_fuel_unit/k_graphite + dz_gas/k_gas ) *
surface_gas_in_grap ...
        + (  R(101) - R(98) ) / ( dz_clad/k_clad + dz_gas/k_gas ) * surface_gas_out_grap
);

% Lower Clad (SS304)
R_dot(99) = 1/(mass_clad_gr * cp_clad) * ( ( R(100) - R(99) )* k_clad * A_clad / (
dz_cond_fuel + dz_cond_grap) ...
    + (  R(96) - R(99) ) / ( dz_clad/k_clad + dz_gas/k_gas ) * surface_gas_out_grap  ...
    - h_bar_1 * surface_area_graphite * (R(99) - R(92)) );

% Mid Clad (SS304)
R_dot(100) = 1/(mass_clad_fuel * cp_clad) * ( ( R(99) - R(100) )* k_clad * A_clad / (
dz_cond_fuel + dz_cond_grap) ...
    + ( R(101) - R(100) )* k_clad * A_clad / ( dz_cond_fuel + dz_cond_grap) ...
    + (  R(97) - R(100) ) / ( dz_clad/k_clad + dz_gas/k_gas ) * surface_gas_out_fuel ...
    - h_bar_2 * surface_area_graphite * (R(100) - R(13)) );

% Upper Clad (SS304)
R_dot(101) = 1/(mass_clad_gr * cp_clad) * ( ( R(100) - R(101) )* k_clad * A_clad / (
dz_cond_fuel + dz_cond_grap) ...
    + (  R(98) - R(101) ) / ( dz_clad/k_clad + dz_gas/k_gas ) * surface_gas_out_grap ...
    - h_bar_3 * surface_area_graphite * (R(101) - R(94)) );

%% Output
R_dot = R_dot';

if ~isreal(R)
    low_temp
    c_temp
    f_temp
    R94_1
    R94_2
    R94_3
    cp_H2O_3
    return;
end
```

Momentum Balance Program

Main Function

```
function [ P_out, T_out, t_out, rho_out, max_P, max_T, max_P_t, max_T_t ] = in_hour_EQN_4_7_0_FUNC( R_in, t_in, RR_in, type)
%% HEADER
% In_hour EQN solver
% Author: Greg Kline
% Date: 08 Jan 2016
% Revision Date 15 Jun 2016
% Revision: 4.6.2
%
% Modelling of a pulse insertion as well as a rod withdraw event.
%
%
% The purpose of this code is to simulate both a pulse event and a continuous
% rod withdraw event.
%
% The parameters are modelled using the University of Texas parameters
% Revisions
% 4.1.0
%    - Added Plutonium 239 to mixture
%    - Accounted for core fuel burnup
```

```
% 4.6.0
%     - Expanded temperature regions to 6 from 2 ( 1 fuel 1 mod, to 3 fuel 3 mod)
%
% P_out is the vector of power vs time (MW)
% T_out is the vector of temperature vs time (C)
% t_out is time (s)
% rho_out is the event reactivity vs time (dk/k)
% max_P is max Power (MW)
% max_T is max temperature (C)
% max_P_t is the time of peak power (s)
% max_T_t is the time of peak temperature (s)
% R_in is the reactivity limit ($)
% t_in is the max time (s)
% RR_in is the reactivity addition rate in (dk/k/s)
% type is model type: 'Rod Withdraw' or 'Pulse'
%
% Example:
% [ P_out, T_out, t_out, rho_out, max_P, max_T, max_P_t, max_T_t ] = in_hour_EQN_4_6_2_FUNC( 3, 10, .001170, 'Rod Withdraw')
%
% Pseudo Code
% Global Variables
%     - Define the global variables that talk to the ODE solving function
%
% User Input
%     - When not in function mode, the user input pings the MATLAB command
%       line for the same function inputs
%
% Fuel Constants
%     - Define the geometric constants IAW local values
%     - Define material properties IAW Simnad and local values
%
% Neutronics Constants
%     - Define isotope fractions
%     - Find the mixture delayed neutron fraction and decay constant
%     - Account for spontaneous fission
%
% Decay Heat Constants
%     - Define the ANS Standard decay heat constants
%
% Define State Dependent Functions using the initial conditions as the state
%     - Find material state properties
%     - Find initial flux
%     - Find instantaneous power
%
% Find Total Power
%
% Build Initial Condition Vector
%     - Neutronics ICs
%     - Thermal-hydraulic ICs
%     - Event Reactivity IC
%
% Output
%     - Find the instantaneous power from neutron population
%     - Find maximum
%     - Calculate peaking factor vector
%     - Apply to temperature vector
%     - Downsize output vectors
%

tic

%% Global Constants
% global Tinit_fuel Tinit_mod  Max_reactivity Reactivity_add_rate
% global  N_238 N_235 surface_area_fuel volume_core pin_height inner_hex
% global mass_U_235 mass_U_238 mass_fuel_mix  Event_type density_fuel
% global N_Pu_239 mass_Pu_239 l_star  alpha_mod volume_cooling_core
```

```
% global Area_pin_internal dz_cond_fuel mass_graphite_half_core
% global surface_area_graphite area_cooling_pin graphite_height number_pins
% global volume_cooling_graphite_core dz_cond_grap radius_pin inner_radius
% global Area_pin_internal_cond flow_outlet_core flow_inlet_core
% global low_fin_height upper_fin_height A_flow_outlet_grid P_flow_outlet_grid
% global lower_cone_height upper_cone_height A_flow_inlet_grid P_flow_inlet_grid
% global volume_cooling_lower_conical volume_cooling_upper_conical

%% User Input
% Default values are shown in []

% Model Type
% Constants for use in function method
Initial_Power = 50;
Tinit_fuel = 16;
Tinit_mod = 16;

% From user input
Max_reactivity = R_in;
t_f = t_in;
Reactivity_add_rate = RR_in;
Model_type = type;
Event_type = type;

% Make a  nice display for the output
display(sprintf(['Your model is: \n' ...
   '   Model Type: %s \n' ...
   '   Reactivity addition rate(dk/k): %2.2d dk/k \n' ...
   '   Maximum Reactivity($): $%d \n' ...
   '   Initial Power(W): %dW \n' ...
   '   Initial Fuel Temperature(C): %dC \n' ...
   '   Initial Moderator Temperature(C): %dC \n' ...
   '   Duration (s): %ds \n'], Model_type, Reactivity_add_rate, ...
   Max_reactivity, Initial_Power, Tinit_fuel, Tinit_mod, t_f));

display(' Beginning Calculations  ... ');

 %% User Input Non-Function Version
% % Default values are shown in []
%
% % Model Type
%
% MT = input( 'Which model are you going to use? Pulse, Rod Withdraw, Both. [Both]', 's');
%
% if isempty(MT)
%     Model_type = 'Both';
%     Event_type = 'Pulse';
%
% elseif strcmp(MT, 'Rod Withdraw')
%     Model_type = 'Rod Withdraw';
%     Event_type = 'Rod Withdraw';
%
% elseif strcmp(MT, 'Pulse')
%     Model_type = 'Pulse';
%     Event_type = 'Pulse';
%
% elseif strcmp(MT, 'Both')
%     % This will run function file twice once with each model so select
%     % pulse first
%     Model_type = 'Both';
%     Event_type = 'Pulse';
%
% else
%     Model_type = 'Pulse';
%     Event_type = 'Pulse';
% end
```

```
%
% % Reactivity Limit/Insertion(Pulse)
% MR = input( 'What is the maximum reactivity for the event? ($) [$2.94]');
%
% if MR <= 0
%    display ('   Reactivity must be a positive value, using default');
%    Max_reactivity = 2.944447748;
%
% elseif MR >= 13
%    display (sprintf(['   This reactivity exceeds the total rod worth of the '...
%    'UT TRIGA core, results may not be accurate']));
%    Max_reactivity = MR;
%
% elseif isnan(MR)
%    display ('   Reactivity must be a numeric value, using default');
%    Max_reactivity = 2.944447748;
%
% elseif isempty(MR)
%    display ('   Reactivity must be a numeric value, using default');
%    Max_reactivity = 2.944447748;
%
% elseif MR > 3.1429 && MR < 13
%    display ('   Reactivity exceeds UT TechSpec limits ');
%    Max_reactivity = MR;
%
% else
%    Max_reactivity = MR;
% end
%
% % Initial Power (W)
% IP = input('What is the initial power? (W) [50W]');
%
% if IP <= 0
%    display ('   Initial Power must be a positive value, using default');
%    Initial_Power = 50;
%
% elseif isnan(IP)
%    display ('   Initial Power must be a numeric value, using default');
%    Initial_Power = 50;
%
% elseif isempty(IP)
%    display ('   Initial Power must be a numeric value, using default');
%    Initial_Power = 50;
%
% elseif IP > 1.1e6
%    display ('   Initial Power exceeds UT TechSpec limits ');
%    Initial_Power = IP;
%
% else
%    Initial_Power = IP;
% end
%
% % Initial Fuel Temperature (C)
% iFT = input('What is the initial fuel temperature? (C) [20C]');
%
% if iFT <= 0
%    display ('   Initial Fuel Temperature must be a positive value, using default');
%    Tinit_fuel = 20;
%
% elseif isnan(iFT)
%    display ('   Initial Fuel Temperature must be a numeric value, using default');
%    Tinit_fuel = 20;
%
% elseif isempty(iFT)
%    display ('   Initial Fuel Temperature must be a numeric value, using default');
%    Tinit_fuel = 20;
```

```
%
% elseif iFT > 950
%    display ('   Initial Fuel Temperature exceeds UT TechSpec limits ');
%    Tinit_fuel = iFT;
%
% else
%    Tinit_fuel = iFT;
% end
%
% % Initial Moderator Temperature (C)
% iMT = input('What is the initial moderator temperature? (C) [20C]');
%
% if iMT <= 0
%    display ('   Initial Moderator Temperature must be a positive value, using default');
%    Tinit_mod = 20;
%
% elseif isnan(iMT)
%    display ('   Initial Moderator Temperature must be a numeric value, using default');
%    Tinit_mod = 20;
%
% elseif isempty(iMT)
%    display ('   Initial Moderator Temperature must be a numeric value, using default');
%    Tinit_mod = 20;
%
% elseif iMT > 48
%    display ('   Initial Moderator Temperature exceeds UT TechSpec limits ');
%    Tinit_mod = iMT;
%
% else
%    Tinit_mod = iMT;
% end
%
% % time length
% ft = input('How long of a run? (s)');
% % check values
% if ft <= -0
%    t_f = 100;
%    display('Time cannot be negative, using 100s');
%
% elseif isnan(ft)
%    t_f = 100;
%    display('Time must be a number, using 100s');
%
% elseif isempty(ft)
%    t_f = 100;
%    display('Time must be a number, using 100s');
%
% else
%    t_f = ft;
%
% end
%
% if sum(strcmp(Model_type, {'Both','Rod Withdraw'}))
%    RR = input('What is the reactivity addition rate?(dk/k) [.002dk/k]');
%
%    if RR <= 0
%       display ('   Reactivity must be a positive value, using default');
%       Reactivity_add_rate = .002;
%
%    elseif isnan(RR)
%       display ('   Reactivity must be a numeric value, using default');
%       Reactivity_add_rate = .002;
%
%    elseif isempty(RR)
%       display ('   Reactivity must be a numeric value, using default');
%       Reactivity_add_rate = .002;
```

```
%
%    elseif RR > .2
%       display ('   Reactivity exceeds UT TechSpec limits ');
%       Reactivity_add_rate = RR;
%
%    else
%       Reactivity_add_rate = RR;
%    end
%
%    % Make a  nice display for the output
%    display(sprintf(['Your model is: \n' ...
%       '   Model Type: %s \n' ...
%       '   Reactivity addition rate(dk/k): %2.2d dk/k \n' ...
%       '   Maximum Reactivity($): $%d \n' ...
%       '   Initial Power(W): %dW \n' ...
%       '   Initial Fuel Temperature(C): %dC \n' ...
%       '   Initial Moderator Temperature(C): %dC \n' ...
%       '   Duration (s): %ds \n'], Model_type, Reactivity_add_rate, ...
%       Max_reactivity, Initial_Power, Tinit_fuel, Tinit_mod, t_f));
%
% elseif ~sum(strcmp(Model_type, {'Both','Rod Withdraw'}))
%
%    Reactivity_add_rate = 0;
%
%    % Make a  nice display for the output
%    display(sprintf(['Your model is: \n' ...
%    '   Model Type: %s \n' ...
%    '   Maximum Reactivity($): $%d \n' ...
%    '   Initial Power(W): %dW \n' ...
%    '   Initial Fuel Temperature(C): %dC \n' ...
%    '   Initial Moderator Temperature(C): %dC \n' ...
%    '   Duration(s): %d \n'], Model_type, Max_reactivity, ...
%    Initial_Power, Tinit_fuel, Tinit_mod, t_f));
%
% else
%    display(' There is a model error');
%
% end
%
% display(' Beginning Calculations  ... ');

%% fuel constants %%%

%%% Geometry %%%
% Pin radius (m)
radius_pin = 0.018771; % .735in
inner_radius = 0.003175; %.125in

% clad width (m)
dl_clad = 0.000508;
dl_gas = - 1.310588235294116e-04;

% Active region (m)
pin_height = .381;
graphite_height = 0.08763;
dz_cond_fuel = .5 * pin_height;
dz_cond_grap = .5 * graphite_height;
Graphite_offset_from_tip = .087884;

%% momentum expansion
% Heights (m)
lower_cone_height = .035;
upper_cone_height = .03;
lower_pin_below_gr = .041;
upper_pin_above_gr = 0.072;
lower_fin_height = lower_cone_height; % long direction
```

```
upper_fin_height = .0625; %(m)
lower_cyl_height = lower_pin_below_gr - lower_fin_height;
upper_cyl_height = upper_pin_above_gr - upper_fin_height;
dz_lower_ss_cond = lower_cyl_height + lower_cone_height / 4;
dz_upper_ss_cond = upper_cyl_height + upper_cone_height / 4;

% Areas and parameters (m^2 , m)
A_grid = pi * (.038227/2)^2;
P_grid = 2 * pi * (.038227/2);
P_pin_top = .12335;
A_pin_top = .00061;
A_flow_outlet_grid = 8.0742e-4;
P_flow_outlet_grid = .2358; %P_grid + P_pin_top;
A_flow_inlet_grid = 5.3771e-4;
P_flow_inlet_grid = P_grid + P_pin_top;
A_s_lower_fin = 2.28336e-3; %lower fin surface area
A_s_upper_fin = 4.63457e-3; % upper fin surface area
A_s_lower_cyl = 2 * pi * radius_pin * lower_cyl_height;
A_s_upper_cyl = 2 * pi * radius_pin * upper_cyl_height;
A_s_lower_cone = pi * radius_pin * sqrt( lower_cyl_height^2 + radius_pin^2);
A_s_upper_cone = pi * radius_pin * sqrt( upper_cyl_height^2 + radius_pin^2);

% Volumes (m^3)
volume_lower_cyl = pi * radius_pin^2 * lower_cyl_height;
volume_lower_cone = pi * radius_pin^2 * lower_cone_height / 3;
volume_upper_cyl = pi * radius_pin^2 * upper_cyl_height;
volume_upper_cone = pi * radius_pin^2 * upper_cone_height / 3;

% Mass and density (kg, kg/m^3)
density_304SS = 7800;
mass_lower_cyl = volume_lower_cyl * density_304SS;
mass_lower_cone = volume_lower_cone * density_304SS;
mass_upper_cyl = volume_upper_cyl * density_304SS;
mass_upper_cone = volume_upper_cone * density_304SS;
mass_104 = mass_lower_cyl + mass_lower_cone;
mass_105 = mass_upper_cyl + mass_upper_cone;

%% Volumes and areas
% pin fuel volume (m^3)
volume_pin = pi * pin_height * ((radius_pin - dl_clad - dl_gas)^2 - inner_radius^2);

% pin graphite volume (m^3)
volume_graphite_pin = pi * radius_pin^2 * graphite_height;

% surface area graphite (m^2)
surface_area_graphite = 2 * pi * radius_pin * graphite_height;

% surface area of pin (m^2)
surface_area_fuel = 2 * pi * radius_pin * pin_height;

%%% Reactor Constants %%%
% Number of pins (#)
number_pins = 113.7; % account for FFCR smaller diameter

% Volume of the core (m^3)
volume_core = volume_pin * number_pins;

% Volume of graphite in one half of core (m^3)
volume_graphite_half_core = volume_graphite_pin * number_pins;

% Surface area core (m^2)
surface_area_fuel = surface_area_fuel * number_pins;

% Radius of cooling hexagon (m)
inner_hex = 0.0217678;
outer_hex = 0.025146;
```

```
% Total spatial area of cooling hex and pin (m^2)
total_space = sqrt(3)/2 * (2 * inner_hex)^2;

% Area of pin radially (m^2)
Area_pin_internal = pi * radius_pin^2;
Area_pin_internal_cond = pi * ( (radius_pin - dl_clad - dl_gas)^2 - inner_radius^2);

% Area of cooling hexagon around pin (m^2)
area_cooling_pin = total_space - Area_pin_internal;

% Volume of cooling (m^3)
% total pin height (m)
total_pin_height = 0.73152;

% the water mass uses the entire height of pin, not just heated length
volume_cooling_pin = area_cooling_pin * pin_height;
volume_cooling_graphite_pin = area_cooling_pin * graphite_height;

volume_cooling_core = volume_cooling_pin * number_pins;
volume_cooling_graphite_core = volume_cooling_graphite_pin * number_pins;
volume_cooling_lower_conical = total_space * 0.059055 - pi * radius_pin^2 * lower_cone_height / 3;
volume_cooling_upper_conical = total_space * 0.059055 - pi * radius_pin^2 * upper_cone_height / 3;

% Flow inlet area (m^2)
% pin top (m^2)
area_top_pin = .00063789;

% area of grid plate hole (m^2)
area_grid_hole = pi * (0.038227/2)^2;

% flow area of a pin (m^2)
flow_area_pin_upper = area_grid_hole - area_top_pin;

flow_outlet_core = flow_area_pin_upper * number_pins;

flow_area_pin_lower = 0.00063885; %m^2

flow_inlet_core = flow_area_pin_lower * number_pins;

% Channel width (m)
width_channel = 0.0061976;

% Percent Burn (Fraction)
burn_factor_238 = 0.922896237;
burn_factor_235 = 0.863175801;

%%% Fuel meat constants %%%
% wt% U_238
U_wt = .085;

% enrichment percentage (fraction) [UT SAR]
Enrich = .197; % 19.7%

% density (kg/m^3) [Simnad]
density_U = 19070;

% density of ZrH based on ratio (kg/m^3)[Simnad]
density_Zr = 1 / (.1706 + .0042 * 1.6) * 1000;

% Material Densities (kg/m^3)
density_fuel = 1 / ( U_wt / density_U + (1 - U_wt) / density_Zr ); %[Simnad]

% density of Pu 239 based on burnup equations (kg/m^3)
density_Pu239 = 19618;
```

```
% density of graphite (kg/m3)
density_graphite = 2500;

% Avogadro's number (atoms/mol)
N_A = 6.022e23;

% Molar mass (kg/mol) [Burns]
M_U = .23807;

% Molar mass Pu239 (kg/mol)
M_Pu = .2390521634;

% Molar mass Sm (kg/mol)
M_Sm = .15036;

% Molar mass Zr (kg/mol)
M_Zr = .091224;

%%% Masses %%%
% Mass Uranium (kg)
mass_U_pin = density_fuel * volume_pin * U_wt;

% Mass U235 (kg)
mass_U_235_pin = mass_U_pin * Enrich;

% Mass U238 (kg)
mass_U_238_pin = mass_U_pin - mass_U_235_pin;

% Mass Pu239 (kg)
mass_Pu239_pin = density_Pu239 * volume_pin;

% Mass Zr per pin (kg)
mass_ZrH_pin = density_fuel * volume_pin * ( 1- U_wt );

% Total U238 mass (kg)
mass_U_238 = mass_U_238_pin * number_pins;

% Total U235 mass (kg)
mass_U_235 = mass_U_235_pin * number_pins;

% Total Pu239 mass (kg)
mass_Pu_239 = mass_Pu239_pin * number_pins;

% Total Fuel Mass (kg)
mass_fuel_mix = density_fuel * volume_pin * number_pins;

% Total mass of one half of graphite (kg)
mass_graphite_half_core = density_graphite * volume_graphite_half_core * number_pins;

%%% Number Densities %%%
% Number density of U238 per pin (atoms/m^3)
N_238_pin = ( mass_U_238_pin / M_U * N_A ) / volume_pin  * burn_factor_238;

% Number density of U235 per pin (atoms/m^3)
N_235_pin = ( mass_U_235_pin / M_U * N_A ) / volume_pin  * burn_factor_235;

% Number density Pu239 per pin (atoms/m^3)
% N_Pu_239_pin = ( mass_Pu_239 / M_Pu * N_A ) / volume_pin;
N_Pu_239_pin = 0.406333333 / M_Pu * N_A;

% Number density of Sm per pin (atoms )
% constants for Sm and Pu are found from nucleonics burnup data and
% are in units of kg/m^3
N_Sm_pin = 0.077307608 / M_Sm * N_A * volume_pin;

% Number density of ZR per pin (atoms)
```

```
N_Zr_pin = ( mass_ZrH_pin / M_Zr * N_A ) / 2.6;

% Number density U238 per core (atoms/m^3)
N_238 = N_238_pin * number_pins;

% Number density U235 per core (atoms/m^3)
N_235 = N_235_pin * number_pins;

% Number Density Pu239 per core (atoms/m^3)
% This value is taken from burn data so it does not need corrected
N_Pu_239 = N_Pu_239_pin * number_pins;

% Number of Sm (atoms)
N_Sm = N_Sm_pin * number_pins;

% Number of Zr (atoms)
N_Zr = N_Zr_pin * number_pins;

% fuel negative coefficient of reactivity in del_k/k/C [UT SAR]
alpha_fuel = -1e-4;

% Moderator temperature coefficient (del_k/k/C)
 alpha_mod = 3.8952e-6;

%% neutronics constants
% Group Lambdas
% U235
U235_L_1 = .0127;
U235_L_2 = .0317;
U235_L_3 = .115;
U235_L_4 = .311;
U235_L_5 = 1.40;
U235_L_6 = 3.87;

% U238
U238_L_1 = .0132;
U238_L_2 = .0321;
U238_L_3 = .139;
U238_L_4 = .358;
U238_L_5 = 1.41;
U238_L_6 = 4.02;

% Pu239
Pu239_L_1 = .0129;
Pu239_L_2 = .0313;
Pu239_L_3 = .135;
Pu239_L_4 = .333;
Pu239_L_5 = 1.36;
Pu239_L_6 = 4.04;

% total delayed neutron fraction U238 [Weaver, 1968]
B_238 = .0157;
% group i fraction is B * relative abundance
B_1_238 = B_238 * .013; %group 1
B_2_238 = B_238 * .137;
B_3_238 = B_238 * .162;
B_4_238 = B_238 * .388;
B_5_238 = B_238 * .225;
B_6_238 = B_238 * .075;

% total delayed neutron fraction U235 [Weaver, 1968]
B_235 = .0065;
B_1_235 = B_235 * .038;
B_2_235 = B_235 * .213;
B_3_235 = B_235 * .188;
B_4_235 = B_235 * .407;
```

```
B_5_235 = B_235 * .128;
B_6_235 = B_235 * .026;

% total delayed neutron fraction Pu239
B_Pu_239 = .0026;
B_1_Pu_239 = B_Pu_239 * .038;
B_2_Pu_239 = B_Pu_239 * .280;
B_3_Pu_239 = B_Pu_239 * .216;
B_4_Pu_239 = B_Pu_239 * .328;
B_5_Pu_239 = B_Pu_239 * .103;
B_6_Pu_239 = B_Pu_239 * .035;

% define fission cross sections for precursor ratios
%fission cross section (m^2)
sig_f_238 = 3.3e-28;

%fission cross section (m^2)
sig_f_235 = 584.4e-28;

% fission cross section (m^2)
sig_f_Pu_239 = 747.4e-28;

% Absorption cross sections for poisons (m^2)
% Zr, Zirconium has 5 stable isotopes, NIST provides cross sections for each
% so a weighted average is taken Zr 90, 91, 92, 94, 95
sig_a_Zr = .5145 * .011e-28 + .1132 * 1.17e-28 + .1719 * .22e-28 ...
   + .1728 * .0499e-28 + .0276 * .0229e-28;

% Samarium has 7 stable states (m^2)
sig_a_Sm = .031 * .7e-28 + .151 * 57e-28 + .113 * 2.4e-28 ...
   + .139 * 42080e-28 + .074 * 104e-28 + .266 * 206e-28 ...
   + .226 * 8.4e-28;

% neutron velocity (m/s)

% Boltzmann constant (m2 kg s-2 K-1)
k_b = 1.38064852e-23;

% Mass of neutron (kg)
m_nu = 1.674927471e-27;

% velocity neutron = 2197;
velocity_neutron = sqrt( 2 * k_b * (Tinit_fuel + 273.15) / m_nu );

% U235 Beta factor (atoms/s)
Beta_factor_U235 = velocity_neutron * sig_f_235 * N_235;

% U238 Beta factor (atoms/s)
Beta_factor_U238 = velocity_neutron * sig_f_238 * N_238;

% Pu239 Beta factor (atoms/s)
Beta_factor_Pu239 = velocity_neutron * sig_f_Pu_239 * N_Pu_239;

% Beta of the DNP mix
B_1_mix = ( Beta_factor_U238 * B_1_238 + Beta_factor_U235 * B_1_235 ...
   + Beta_factor_Pu239 * B_1_Pu_239) ...
   / ( Beta_factor_U238 + Beta_factor_U235 + Beta_factor_Pu239);
B_2_mix = ( Beta_factor_U238 * B_2_238 + Beta_factor_U235 * B_2_235 ...
   + Beta_factor_Pu239 * B_2_Pu_239) ...
   / ( Beta_factor_U238 + Beta_factor_U235 + Beta_factor_Pu239);
B_3_mix = ( Beta_factor_U238 * B_3_238 + Beta_factor_U235 * B_3_235 ...
   + Beta_factor_Pu239 * B_3_Pu_239) ...
   / ( Beta_factor_U238 + Beta_factor_U235 + Beta_factor_Pu239);
B_4_mix = ( Beta_factor_U238 * B_4_238 + Beta_factor_U235 * B_4_235 ...
   + Beta_factor_Pu239 * B_4_Pu_239) ...
   / ( Beta_factor_U238 + Beta_factor_U235 + Beta_factor_Pu239);
```

```
B_5_mix = ( Beta_factor_U238 * B_5_238 + Beta_factor_U235 * B_5_235 ...
   + Beta_factor_Pu239 * B_5_Pu_239) ...
   / ( Beta_factor_U238 + Beta_factor_U235 + Beta_factor_Pu239);
B_6_mix = ( Beta_factor_U238 * B_6_238 + Beta_factor_U235 * B_6_235 ...
   + Beta_factor_Pu239 * B_6_Pu_239) ...
   / ( Beta_factor_U238 + Beta_factor_U235 + Beta_factor_Pu239);

% Lambda of the DNP mix
lambda_1 = ( Beta_factor_U238 * U238_L_1 + Beta_factor_U235 * U235_L_1 ...
   + Beta_factor_Pu239 * Pu239_L_1) ...
   / ( Beta_factor_U238 + Beta_factor_U235 + Beta_factor_Pu239);
lambda_2 = ( Beta_factor_U238 * U238_L_2 + Beta_factor_U235 * U235_L_2 ...
   + Beta_factor_Pu239 * Pu239_L_2) ...
   / ( Beta_factor_U238 + Beta_factor_U235 + Beta_factor_Pu239);
lambda_3 = ( Beta_factor_U238 * U238_L_3 + Beta_factor_U235 * U235_L_3 ...
   + Beta_factor_Pu239 * Pu239_L_3) ...
   / ( Beta_factor_U238 + Beta_factor_U235 + Beta_factor_Pu239);
lambda_4 = ( Beta_factor_U238 * U238_L_4 + Beta_factor_U235 * U235_L_4 ...
   + Beta_factor_Pu239 * Pu239_L_4) ...
   / ( Beta_factor_U238 + Beta_factor_U235 + Beta_factor_Pu239);
lambda_5 = ( Beta_factor_U238 * U238_L_5 + Beta_factor_U235 * U235_L_5 ...
   + Beta_factor_Pu239 * Pu239_L_5) ...
   / ( Beta_factor_U238 + Beta_factor_U235 + Beta_factor_Pu239);
lambda_6 = ( Beta_factor_U238 * U238_L_6 + Beta_factor_U235 * U235_L_6 ...
   + Beta_factor_Pu239 * Pu239_L_6) ...
   / ( Beta_factor_U238 + Beta_factor_U235 + Beta_factor_Pu239);

% Beta mix
% Beff correction factor based on Beff/B ratio for U235 (.007/.0065)
% Weaver and Hetrick show up to 30% difference between B and Beff
Beta_correction = 1.076923076923077;

% Beta mix
Beta_mix = B_1_mix + B_2_mix + B_3_mix + B_4_mix + B_5_mix + B_6_mix;

% Beff mix
Beff_mix = Beta_mix * Beta_correction;

%prompt neutron lifetime (s) [UT SAR]
l_star = 51.9e-6;
% l_star = 41e-6;

% 2 Ci AmBe source (atoms/s)
AmBe_source = 7.4e10;

% U238 spontaneous fission (atoms/s)
U238_spontaneous = 1.80e-2 * mass_U_238 * 1000; % factor in (n/g-s)

% U235 spontaneous fission (atoms/s)
U235_spontaneous = 7.43e-4 * mass_U_235 * 1000; % factor in (n/g-s)

% Pu239 spontaneous fission (atoms/s)
Pu_239_spontaneuos = 2.30e-2 * mass_Pu_239 * 1000;

%% Decay Heat Constants %%%
% Decay (1/s)
% U235
H_lambda_235(1) = 2.2138e1;
H_lambda_235(2) = 5.1587e-1;
H_lambda_235(3) = 1.9594e-1;
H_lambda_235(4) = 1.0314e-1;
H_lambda_235(5) = 3.3656e-2;
H_lambda_235(6) = 1.1681e-2;
H_lambda_235(7) = 3.5870e-3;
H_lambda_235(8) = 1.3930e-3;
H_lambda_235(9) = 6.2630e-4;
```

```
H_lambda_235(10) = 1.8906e-4;
H_lambda_235(11) = 5.4988e-5;
H_lambda_235(12) = 2.0958e-5;
H_lambda_235(13) = 1.0010e-5;
H_lambda_235(14) = 2.5438e-6;
H_lambda_235(15) = 6.6361e-7;
H_lambda_235(16) = 1.2290e-7;
H_lambda_235(17) = 2.7213e-8;
H_lambda_235(18) = 4.3714e-9;
H_lambda_235(19) = 7.5780e-10;
H_lambda_235(20) = 2.4786e-10;
H_lambda_235(21) = 2.2384e-13;
H_lambda_235(22) = 2.4600e-14;
H_lambda_235(23) = 1.5699e-14;

% U238
H_lambda_238(1) = 3.2881e00;
H_lambda_238(2) = 9.3805e-1;
H_lambda_238(3) = 3.7073e-1;
H_lambda_238(4) = 1.1118e-1;
H_lambda_238(5) = 3.6143e-2;
H_lambda_238(6) = 1.3272e-2;
H_lambda_238(7) = 5.0133e-3;
H_lambda_238(8) = 1.3655e-3;
H_lambda_238(9) = 5.5158e-4;
H_lambda_238(10) = 1.7873e-4;
H_lambda_238(11) = 4.9032e-5;
H_lambda_238(12) = 1.7058e-5;
H_lambda_238(13) = 7.0465e-6;
H_lambda_238(14) = 2.3190e-6;
H_lambda_238(15) = 6.4480e-7;
H_lambda_238(16) = 1.2649e-7;
H_lambda_238(17) = 2.5548e-8;
H_lambda_238(18) = 8.4782e-9;
H_lambda_238(19) = 7.5130e-10;
H_lambda_238(20) = 2.4188e-10;
H_lambda_238(21) = 2.2739e-13;
H_lambda_238(22) = 9.0536e-14;
H_lambda_238(23) = 5.6098e-15;

% Pu239
H_lambda_Pu239(1) = 1.0020e1;
H_lambda_Pu239(2) = 6.4330e-1;
H_lambda_Pu239(3) = 2.1860e-1;
H_lambda_Pu239(4) = 1.0040e-1;
H_lambda_Pu239(5) = 3.7280e-2;
H_lambda_Pu239(6) = 1.4350e-2;
H_lambda_Pu239(7) = 4.5490e-3;
H_lambda_Pu239(8) = 1.3280e-3;
H_lambda_Pu239(9) = 5.3560e-4;
H_lambda_Pu239(10) = 1.7300e-4;
H_lambda_Pu239(11) = 4.8810e-5;
H_lambda_Pu239(12) = 2.0060e-5;
H_lambda_Pu239(13) = 8.3190e-6;
H_lambda_Pu239(14) = 2.3580e-6;
H_lambda_Pu239(15) = 6.4500e-7;
H_lambda_Pu239(16) = 1.2780e-7;
H_lambda_Pu239(17) = 2.4660e-8;
H_lambda_Pu239(18) = 9.3780e-9;
H_lambda_Pu239(19) = 7.4500e-10;
H_lambda_Pu239(20) = 2.4260e-10;
H_lambda_Pu239(21) = 2.2100e-13;
H_lambda_Pu239(22) = 2.6400e-14;
H_lambda_Pu239(23) = 1.3800e-14;

% Power fraction
```

```
% U235
H_fraction_235(1) = 6.5057e-1;
H_fraction_235(2) = 5.1264e-1;
H_fraction_235(3) = 2.4384e-1;
H_fraction_235(4) = 1.3850e-1;
H_fraction_235(5) = 5.5440e-2;
H_fraction_235(6) = 2.2225e-2;
H_fraction_235(7) = 3.3088e-3;
H_fraction_235(8) = 9.3015e-4;
H_fraction_235(9) = 8.0943e-4;
H_fraction_235(10) = 1.9567e-4;
H_fraction_235(11) = 3.2535e-5;
H_fraction_235(12) = 7.5595e-6;
H_fraction_235(13) = 2.5232e-6;
H_fraction_235(14) = 4.9948e-7;
H_fraction_235(15) = 1.8531e-7;
H_fraction_235(16) = 2.6608e-8;
H_fraction_235(17) = 2.2398e-9;
H_fraction_235(18) = 8.1641e-12;
H_fraction_235(19) = 8.7797e-11;
H_fraction_235(20) = 2.5131e-14;
H_fraction_235(21) = 3.2176e-16;
H_fraction_235(22) = 4.5038e-17;
H_fraction_235(23) = 7.4791e-17;

% U238
H_fraction_238(1) = 1.2311e00;
H_fraction_238(2) = 1.1486e00;
H_fraction_238(3) = 7.0701e-1;
H_fraction_238(4) = 2.5209e-1;
H_fraction_238(5) = 7.1870e-2;
H_fraction_238(6) = 2.8291e-2;
H_fraction_238(7) = 6.8382e-3;
H_fraction_238(8) = 1.2322e-3;
H_fraction_238(9) = 6.8409e-4;
H_fraction_238(10) = 1.6975e-4;
H_fraction_238(11) = 2.4182e-5;
H_fraction_238(12) = 6.6356e-6;
H_fraction_238(13) = 1.0075e-6;
H_fraction_238(14) = 4.9894e-7;
H_fraction_238(15) = 1.6352e-7;
H_fraction_238(16) = 2.3355e-8;
H_fraction_238(17) = 2.8094e-9;
H_fraction_238(18) = 3.6236e-11;
H_fraction_238(19) = 6.4577e-11;
H_fraction_238(20) = 4.4963e-14;
H_fraction_238(21) = 3.6654e-16;
H_fraction_238(22) = 5.6293e-17;
H_fraction_238(23) = 7.1602e-17;

% Pu239
H_fraction_Pu239(1) = 2.0830e-1;
H_fraction_Pu239(2) = 3.8530e-1;
H_fraction_Pu239(3) = 2.2130e-1;
H_fraction_Pu239(4) = 9.4600e-2;
H_fraction_Pu239(5) = 3.5310e-2;
H_fraction_Pu239(6) = 2.2920e-2;
H_fraction_Pu239(7) = 3.9460e-3;
H_fraction_Pu239(8) = 1.3170e-3;
H_fraction_Pu239(9) = 7.0520e-4;
H_fraction_Pu239(10) = 1.4320e-4;
H_fraction_Pu239(11) = 1.7650e-5;
H_fraction_Pu239(12) = 7.3470e-6;
H_fraction_Pu239(13) = 1.7470e-6;
H_fraction_Pu239(14) = 5.4810e-7;
H_fraction_Pu239(15) = 1.6710e-7;
```

```
H_fraction_Pu239(16) = 2.1120e-8;
H_fraction_Pu239(17) = 2.9960e-9;
H_fraction_Pu239(18) = 5.1070e-11;
H_fraction_Pu239(19) = 5.7300e-11;
H_fraction_Pu239(20) = 4.1380e-14;
H_fraction_Pu239(21) = 1.0880e-15;
H_fraction_Pu239(22) = 2.4540e-17;
H_fraction_Pu239(23) = 7.5570e-17;

%% State dependent functions
display(' Calculating State Dependent Functions ...');

% place holder for delta t
t_last = 0;

% volumetric heat capacity from Simnad (J/ m3 K)
cp_fuel_vol = (2.04 + 4.17e-3 * Tinit_fuel ) * 1e6;

% Convert to specific heat ( J / kg K )
cp_fuel = cp_fuel_vol / density_fuel;

% find keff
current_keff = 1;

% keff adjusted neutron lifetime (s)
A = l_star / current_keff;

% Heat transfer coefficient (W/m^2K) UT LOCA
h = 3200;  %3200 default

% Instantaneous Power (W)
% Find the initial flux and neutron population from user input
% Flux ( n / m^2 s)
initial_flux = Initial_Power / ( ...
   (200e6 * 1.602677e-19) * N_235 * sig_f_235 * volume_core ...
   + (200e6 * 1.602677e-19) * N_238 * sig_f_238 * volume_core ...
   + (200e6 * 1.602677e-19) * N_Pu_239 * sig_f_Pu_239 * volume_core );

% neutron density (n/m^3)
initial_neutron_density = initial_flux / velocity_neutron;

% U235 contribution
P_inst_235 = initial_flux * (200e6 * 1.602677e-19) * N_235 ...
   * sig_f_235 * volume_core;

% U238 contribution
P_inst_238 = initial_flux * (200e6 * 1.602677e-19) * N_238 ...
   * sig_f_238 * volume_core;

% Pu239 contribution (W)
P_inst_Pu239 = initial_flux * velocity_neutron * (200e6 * 1.602677e-19) * N_Pu_239 ...
   * sig_f_Pu_239 * volume_core;

% Total instantaneous (W)
P_inst = P_inst_235 + P_inst_238 + P_inst_Pu239;

% Temperature Dependent Density (kg/m^3) [ddbst.de, 273K-648K]
A_cp = .14395;
B_cp = .0112;
C_cp = 649.727;
D_cp = .05107;

density_init_water = A_cp / ( B_cp^(1 + (1 - (Tinit_mod + 273.15)/C_cp)^D_cp));

% Mass of water (kg)
mass_cooling_water = volume_cooling_core * density_init_water;
```

```
% Temperature dependent Specific heat (KJ/kg) [steam tables]
cp_H2O_KJ = 3.16744e-10 * Tinit_mod^4 - 1.05772e-7 * Tinit_mod^3 ...
   + 2.35330e-5 * Tinit_mod^2 - 1.47670e-3 * Tinit_mod + 4.20617e0;

% Convert to J/kg
cp_H2O = cp_H2O_KJ * 1000;

% specific heat capacity of graphite (cal / g C) [Entegris, inc.]
cp_graphite_cal = .10795e8 * Tinit_fuel^-3 - .61257e5 * Tinit_fuel^-2 ...
   + .30795e-4 * Tinit_fuel + .44391;
cp_graphite = 4183.995381 * cp_graphite_cal;

% Temperature dependent alpha_t
alpha_fuel_T = -1.62895e-7 * Tinit_fuel - 2.53543e-5;

%% Delayed Power State Function (W)
display(' Calculating Delayed Power Initial Condition... ');

% U235 delayed
P_d_235 =  ( P_inst_235 / 200 ) * sum( H_fraction_235 ./ H_lambda_235 );

% U238 delayed
P_d_238 =  ( P_inst_238 / 200 ) * sum( H_fraction_238 ./ H_lambda_238 );

% Pu239 delayed (W)
P_d_Pu_239 = ( P_inst_Pu239 / 200 ) * sum( H_fraction_Pu239 ./ H_lambda_Pu239 );

%% Total power by isotope (W)
% U235
P_i_235 = P_inst_235 + P_d_235;

% U238
P_i_238 = P_inst_238 + P_d_238;

% Pu239
P_i_Pu239 = P_inst_Pu239 + P_d_Pu_239;

%% Initial Condition Vector
display(' Building Initial Condition Vector... ');

% Build initial condition vector based on inputs
% In hour items
IC(1) = initial_neutron_density;
IC(2) =  (B_1_mix * initial_neutron_density / A ) / lambda_1;
IC(3) =  (B_2_mix * initial_neutron_density / A ) / lambda_2;
IC(4) =  (B_3_mix * initial_neutron_density / A ) / lambda_3;
IC(5) =  (B_4_mix * initial_neutron_density / A ) / lambda_4;
IC(6) =  (B_5_mix * initial_neutron_density / A ) / lambda_5;
IC(7) =  (B_6_mix * initial_neutron_density / A ) / lambda_6;

% Sources
% AmBe (n/s/m^3)
IC(8) = AmBe_source;
IC(9) = U238_spontaneous;
IC(10) = U235_spontaneous;

% Reactivity (assuming keff = 1, this IC is too balance the moderator and fuel
IC(12) = Tinit_fuel;
IC(13) = Tinit_mod;

% Betas
IC(14) = Beff_mix;
IC(15) = B_1_mix;
IC(16) = B_2_mix;
IC(17) = B_3_mix;
```

```
IC(18) = B_4_mix;
IC(19) = B_5_mix;
IC(20) = B_6_mix;

display(' Building Decay Heat Matrix Initial Condition... ');

% Decay heat matrix
% U235 Fraction
IC(21) = (H_fraction_235(1) * P_i_235) / (200 * H_lambda_235(1));
IC(22) = (H_fraction_235(2) * P_i_235) / (200 * H_lambda_235(2));
IC(23) = (H_fraction_235(3) * P_i_235) / (200 * H_lambda_235(3));
IC(24) = (H_fraction_235(4) * P_i_235) / (200 * H_lambda_235(4));
IC(25) = (H_fraction_235(5) * P_i_235) / (200 * H_lambda_235(5));
IC(26) = (H_fraction_235(6) * P_i_235) / (200 * H_lambda_235(6));
IC(27) = (H_fraction_235(7) * P_i_235) / (200 * H_lambda_235(7));
IC(28) = (H_fraction_235(8) * P_i_235) / (200 * H_lambda_235(8));
IC(29) = (H_fraction_235(9) * P_i_235) / (200 * H_lambda_235(9));
IC(30) = (H_fraction_235(10) * P_i_235) / (200 * H_lambda_235(10));
IC(31) = (H_fraction_235(11) * P_i_235) / (200 * H_lambda_235(11));
IC(32) = (H_fraction_235(12) * P_i_235) / (200 * H_lambda_235(12));
IC(33) = (H_fraction_235(13) * P_i_235) / (200 * H_lambda_235(13));
IC(34) = (H_fraction_235(14) * P_i_235) / (200 * H_lambda_235(14));
IC(35) = (H_fraction_235(15) * P_i_235) / (200 * H_lambda_235(15));
IC(36) = (H_fraction_235(16) * P_i_235) / (200 * H_lambda_235(16));
IC(37) = (H_fraction_235(17) * P_i_235) / (200 * H_lambda_235(17));
IC(38) = (H_fraction_235(18) * P_i_235) / (200 * H_lambda_235(18));
IC(39) = (H_fraction_235(19) * P_i_235) / (200 * H_lambda_235(19));
IC(40) = (H_fraction_235(20) * P_i_235) / (200 * H_lambda_235(20));
IC(41) = (H_fraction_235(21) * P_i_235) / (200 * H_lambda_235(21));
IC(42) = (H_fraction_235(22) * P_i_235) / (200 * H_lambda_235(22));
IC(43) = (H_fraction_235(23) * P_i_235) / (200 * H_lambda_235(23));

% U238 Fraction
IC(44) = (H_fraction_238(1) * P_i_238) / (200 * H_lambda_238(1));
IC(45) = (H_fraction_238(2) * P_i_238) / (200 * H_lambda_238(2));
IC(46) = (H_fraction_238(3) * P_i_238) / (200 * H_lambda_238(3));
IC(47) = (H_fraction_238(4) * P_i_238) / (200 * H_lambda_238(4));
IC(48) = (H_fraction_238(5) * P_i_238) / (200 * H_lambda_238(5));
IC(49) = (H_fraction_238(6) * P_i_238) / (200 * H_lambda_238(6));
IC(50) = (H_fraction_238(7) * P_i_238) / (200 * H_lambda_238(7));
IC(51) = (H_fraction_238(8) * P_i_238) / (200 * H_lambda_238(8));
IC(52) = (H_fraction_238(9) * P_i_238) / (200 * H_lambda_238(9));
IC(53) = (H_fraction_238(10) * P_i_238) / (200 * H_lambda_238(10));
IC(54) = (H_fraction_238(11) * P_i_238) / (200 * H_lambda_238(11));
IC(55) = (H_fraction_238(12) * P_i_238) / (200 * H_lambda_238(12));
IC(56) = (H_fraction_238(13) * P_i_238) / (200 * H_lambda_238(13));
IC(57) = (H_fraction_238(14) * P_i_238) / (200 * H_lambda_238(14));
IC(58) = (H_fraction_238(15) * P_i_238) / (200 * H_lambda_238(15));
IC(59) = (H_fraction_238(16) * P_i_238) / (200 * H_lambda_238(16));
IC(60) = (H_fraction_238(17) * P_i_238) / (200 * H_lambda_238(17));
IC(61) = (H_fraction_238(18) * P_i_238) / (200 * H_lambda_238(18));
IC(62) = (H_fraction_238(19) * P_i_238) / (200 * H_lambda_238(19));
IC(63) = (H_fraction_238(20) * P_i_238) / (200 * H_lambda_238(20));
IC(64) = (H_fraction_238(21) * P_i_238) / (200 * H_lambda_238(21));
IC(65) = (H_fraction_238(22) * P_i_238) / (200 * H_lambda_238(22));
IC(66) = (H_fraction_238(23) * P_i_238) / (200 * H_lambda_238(23));

IC(68) = (H_fraction_Pu239(1) * P_i_Pu239) / (200 * H_lambda_Pu239(1));
IC(69) = (H_fraction_Pu239(2) * P_i_Pu239) / (200 * H_lambda_Pu239(2));
IC(70) = (H_fraction_Pu239(3) * P_i_Pu239) / (200 * H_lambda_Pu239(3));
IC(71) = (H_fraction_Pu239(4) * P_i_Pu239) / (200 * H_lambda_Pu239(4));
IC(72) = (H_fraction_Pu239(5) * P_i_Pu239) / (200 * H_lambda_Pu239(5));
IC(73) = (H_fraction_Pu239(6) * P_i_Pu239) / (200 * H_lambda_Pu239(6));
IC(74) = (H_fraction_Pu239(7) * P_i_Pu239) / (200 * H_lambda_Pu239(7));
IC(75) = (H_fraction_Pu239(8) * P_i_Pu239) / (200 * H_lambda_Pu239(8));
IC(76) = (H_fraction_Pu239(9) * P_i_Pu239) / (200 * H_lambda_Pu239(9));
```

```
IC(77) = (H_fraction_Pu239(10) * P_i_Pu239) / (200 * H_lambda_Pu239(10));
IC(78) = (H_fraction_Pu239(11) * P_i_Pu239) / (200 * H_lambda_Pu239(11));
IC(79) = (H_fraction_Pu239(12) * P_i_Pu239) / (200 * H_lambda_Pu239(12));
IC(80) = (H_fraction_Pu239(13) * P_i_Pu239) / (200 * H_lambda_Pu239(13));
IC(81) = (H_fraction_Pu239(14) * P_i_Pu239) / (200 * H_lambda_Pu239(14));
IC(82) = (H_fraction_Pu239(15) * P_i_Pu239) / (200 * H_lambda_Pu239(15));
IC(83) = (H_fraction_Pu239(16) * P_i_Pu239) / (200 * H_lambda_Pu239(16));
IC(84) = (H_fraction_Pu239(17) * P_i_Pu239) / (200 * H_lambda_Pu239(17));
IC(85) = (H_fraction_Pu239(18) * P_i_Pu239) / (200 * H_lambda_Pu239(18));
IC(86) = (H_fraction_Pu239(19) * P_i_Pu239) / (200 * H_lambda_Pu239(19));
IC(87) = (H_fraction_Pu239(20) * P_i_Pu239) / (200 * H_lambda_Pu239(20));
IC(88) = (H_fraction_Pu239(21) * P_i_Pu239) / (200 * H_lambda_Pu239(21));
IC(89) = (H_fraction_Pu239(22) * P_i_Pu239) / (200 * H_lambda_Pu239(22));
IC(90) = (H_fraction_Pu239(23) * P_i_Pu239) / (200 * H_lambda_Pu239(23));

display(' Building Model Specific Reactivity IC... ');

% Rod reactivity
if sum(strcmp(Model_type, {'Rod Withdraw'}))
   % Start
   IC(67) = 0;
   IC(11) = -(alpha_fuel_T * Tinit_fuel + alpha_mod * Tinit_mod );

elseif ~sum(strcmp(Model_type, {'Rod Withdraw'}))
   IC(67) = Max_reactivity * Beff_mix;
   IC(11) = -(alpha_fuel_T * Tinit_fuel + alpha_mod * Tinit_mod ) ...
      + Max_reactivity * Beff_mix;

else
   IC(67) = 0;
   IC(11) = -(alpha_fuel_T * Tinit_fuel + alpha_mod * Tinit_mod );

end

% Temperature expansion
IC(91) = Tinit_fuel;
IC(92) = Tinit_mod;
IC(93) = Tinit_fuel;
IC(94) = Tinit_mod;

% Initial coolant velocity (m/s) [<POAH]
IC(95) = 0;

% Gas / SS expansion
IC(96) = Tinit_mod;
IC(97) = Tinit_mod;
IC(98) = Tinit_mod;
IC(99) = Tinit_mod;
IC(100) = Tinit_mod;
IC(101) = Tinit_mod;

% Momentum expansion
IC(102) = Tinit_mod;
IC(103) = Tinit_mod;
IC(104) = Tinit_fuel;
IC(105) = Tinit_fuel;

%% Build parameter vector to pass to ode function

GV = [Tinit_fuel Tinit_mod Max_reactivity Reactivity_add_rate N_238 N_235 ...
   surface_area_fuel volume_core pin_height inner_hex mass_U_235 mass_U_238 ...
   mass_fuel_mix 0 density_fuel N_Pu_239 mass_Pu_239 l_star alpha_mod ...
   volume_cooling_core Area_pin_internal dz_cond_fuel mass_graphite_half_core ...
   surface_area_graphite area_cooling_pin graphite_height number_pins ...
   volume_cooling_graphite_core dz_cond_grap radius_pin inner_radius ...
   Area_pin_internal_cond flow_outlet_core flow_inlet_core lower_fin_height ...
```

```
   upper_fin_height A_flow_outlet_grid P_flow_outlet_grid lower_cone_height ...
   upper_cone_height A_flow_inlet_grid P_flow_inlet_grid ...
   volume_cooling_lower_conical volume_cooling_upper_conical ...
   lower_pin_below_gr upper_pin_above_gr lower_fin_height ...
   lower_cyl_height upper_cyl_height mass_104 mass_105 ...
   dz_lower_ss_cond dz_upper_ss_cond A_s_lower_fin A_s_upper_fin ...
   A_s_lower_cyl A_s_upper_cyl A_s_lower_cone A_s_upper_cone ];


SV = [ Event_type ];
%% Solve for time dependent ODE
% Always run at least one
display(' Beginning ODE calculations... ');

% Establish arrays for saving
Y_out_L = [];
T_out_L = [];
t_out_calc = [];
IC_loop(1,:) = IC;
rho_event = [];
Tm_out_L = [];

% Divide into time for display output
N = 1000000000;

% Build the option set for the ODE
% Is there a peak in the power curve
peak_found = false;

% has the derivative fallen off enough to be past the power peak
dpdt_e = false;
e = 1;

% account for telling the user error switched
switch_error = false;

option = odeset('RelTol', 1e-5,'AbsTol',1e-5,'Vectorized','on');

% Loop the solution of the ODE to allow dumping of array values needed for
% solution but not the output
for i = 1:N
   if ~mod(i,10000)
      display(sprintf(' \nBeginning Run %d of %d', i, N));
   end

   IC_loop(i,:);

   % Establish error tolerances based on the curves
   if peak_found && dpdt_e && switch_error
      option = odeset('RelTol', 1e-4,'AbsTol',1e-4,'Vectorized','on');
   elseif peak_found && dpdt_e && ~switch_error
      option = odeset('RelTol', 1e-4,'AbsTol',1e-4,'Vectorized','on');
      display(' Switching error tolerances ');
      switch_error = true;
   else
      option = odeset('RelTol', 1e-5,'AbsTol',1e-5,'Vectorized','on');
   end

   % Calculate the subsections ODE
   [t_out_loop, Y_out_loop] = ode113(@(t,y)in_hour_FUN_7_0_0(t,y,GV,SV), [(i-1)*t_f/N i*t_f/N], IC_loop(i,:), option);

   % Update the output vectors
   Y_out_L = [ Y_out_L Y_out_loop(:,1)' ];
   T_out_L = [ T_out_L Y_out_loop(:,12)' ];
   t_out_calc = [ t_out_calc t_out_loop' ];
   rho_event = [rho_event Y_out_loop(:,67)' ];
```

```
    Tm_out_L = [ Tm_out_L Y_out_loop(:,13)' ];

    % Build the next loop's initial condition
    IC_loop(i+1,:) = Y_out_loop(end,:);

    % Check for error tolerance conditions
    if ~peak_found
        peak_found = ~isempty(findpeaks(Y_out_loop(:,1)));
    end

    dpdt_e = max(abs(diff(Y_out_loop(:,1)))) < e;

    if ~isreal(Y_out_loop)
        Y_out_loop
        return;
    end

    clear Y_out_loop t_out_loop;

end

%% Output
display(' Building Output Displays... ');

% Find output items of concern
% Find the power out in MW
P_out_temp = 1e-6 * Y_out_L * velocity_neutron * (200e6 * 1.602677e-19) * volume_core ...
    * ( N_235 * sig_f_235 + N_238 * sig_f_238 + N_Pu_239 * sig_f_Pu_239 );

% Find the maximum power (MW) and time it occurs (s)
[ max_P, max_P_t_i] = max(P_out_temp);

% peaking factors based on power, this accounts for a higher pin temperature in the B ring
% vs the average core temperature from the calculations (unit less)
% y = 2.4338E-19x3 - 5.3307E-13x2 + 4.1352E-07x + 1.0094E+00
peaking_factor_12_1 = 2.433e-19 .* P_out_temp.^3 - 5.3307e-13 .* P_out_temp.^2 ...
    + 4.1352e-7 .* P_out_temp + 1.0094;

% Peak core temperatures corrected (C)
T_out_temp = peaking_factor_12_1 .* T_out_L;

% Maximum core average temperature (C) and time it occurs (s)
[ max_T, max_T_t_i ] = max(T_out_temp);

max_P_t = t_out_calc(max_P_t_i);

max_T_t = t_out_calc(max_T_t_i);

% In order to save memory, make subsets of the gigantic arrays.  This shows a trend
% but saves space in the calling function of this function
Np = floor(linspace(1,length(t_out_calc), 10000));

P_out = P_out_temp(Np);

T_out = T_out_temp(Np);

t_out = t_out_calc(Np);

rho_out = rho_event(Np);

Tm_out = Tm_out_L(Np);

toc
```

ODE Solver

```
function [ R_dot ] = in_hour_FUN_7_0_0(t, R, GV, SV )
%% In hour equation function file
% The University of Texas at Austin (UT)
% Author: Greg Kline
% Date: 7/3/2015
% Revision: 6.2.0
%
% Modelling of a pulse insertion as well as a rod withdraw event using the
% information in Simnad, 1980 and Johnson, Lucas Tsvetkov, 2010
%
% The purpose of this code is to simulate both a pulse event and a continuous
% rod withdraw event.
%
% The parameters are modelled using the University of Texas parameters
%
% Revisions
% 5.1.0
%     - Added Plutonium 239 to mixture
%     - Accounted for core fuel burnup
%
% 5.2.0
%     - Added changes in moderator temperature based on temperature dependent
%         properties
%
% 5.3.0
%     - Added changing h based on channel properties
%
% 6.0.0
%    - Segmented fuel pin temperature regions and moderator regions
%
% 6.1.0
%    - Added gas and SS interactions
%
% 6.2.0
%    - Finalized state dependencies
% 7.0.0
%    - Momentum balance, functionalization of water properties
%
%ODE Variables
% R(1) - neutron density
% R(2) - group 1 concentration
% R(3) - group 2 concentration
% R(4) - group 3 concentration
% R(5) - group 4 concentration
% R(6) - group 5 concentration
% R(7) - group 6 concentration
% R(8) - AmBe source concentration
% R(9) - spontaneous U238 concentration
% R(10) - spontaneous U235 concentration
% R(11) - reactivity at t
% R(12) - temperature of the fuel at t
% R(13) - temperature of the moderator at t
% R(14) - B_eff_mix at t
% R(15) - B_mix_1 at t
% R(16) - B_mix_2 at t
% R(17) - B_mix_3 at t
% R(18) - B_mix_4 at t
% R(19) - B_mix_5 at t
% R(20) - B_mix_6 at t
% R(21) - H_U2325_group_1 decay heat matrix
% R(22) - H_U2325_group_2
% R(23) - H_U2325_group_3
% R(24) - H_U2325_group_4
% R(25) - H_U2325_group_5
% R(26) - H_U2325_group_6
% R(27) - H_U2325_group_7
% R(28) - H_U2325_group_8
% R(29) - H_U2325_group_9
```

```
% R(30) - H_U2325_group_10
% R(31) - H_U2325_group_11
% R(32) - H_U2325_group_12
% R(33) - H_U2325_group_13
% R(34) - H_U2325_group_14
% R(35) - H_U2325_group_15
% R(36) - H_U2325_group_16
% R(37) - H_U2325_group_17
% R(38) - H_U2325_group_18
% R(39) - H_U2325_group_19
% R(40) - H_U2325_group_20
% R(41) - H_U2325_group_21
% R(42) - H_U2325_group_22
% R(43) - H_U2325_group_23
% R(44) - H_U2328_group_1
% R(45) - H_U2328_group_2
% R(46) - H_U2328_group_3
% R(47) - H_U2328_group_4
% R(48) - H_U2328_group_5
% R(49) - H_U2328_group_6
% R(50) - H_U2328_group_7
% R(51) - H_U2328_group_8
% R(52) - H_U2328_group_9
% R(53) - H_U2328_group_10
% R(54) - H_U2328_group_11
% R(55) - H_U2328_group_12
% R(56) - H_U2328_group_13
% R(57) - H_U2328_group_14
% R(58) - H_U2328_group_15
% R(59) - H_U2328_group_16
% R(60) - H_U2328_group_17
% R(61) - H_U2328_group_18
% R(62) - H_U2328_group_19
% R(63) - H_U2328_group_20
% R(64) - H_U2328_group_21
% R(65) - H_U2328_group_22
% R(66) - H_U2328_group_23
% R(67) - control rod reactivity at t
% R(68-90) - Pu239
% R(91) - Lower graphite temp
% R(92) - Lower moderator temp
% R(93) - Upper graphite temp
% R(94) - Upper mod temp
% R(95) - velocity
% R(96) - Lower Gas region
% R(97) - Mid Gas region
% R(98) - Upper Gas region
% R(99) - Lower SS Region
% R(100) - Mid SS region
% R(101) - Upper SS region
% R(102) - Lower pin grid plate coolant region
% R(103) - Upper pin grid plate coolant region
% R(104) - Pin lower fins and region
% R(105) - Pin upper fins and region

%% Constants
%%% global variables %%
% global Tinit_fuel Tinit_mod  Max_reactivity Reactivity_add_rate
% global N_238 N_235 surface_area_fuel volume_core pin_height inner_hex
% global mass_U_235 mass_U_238 mass_fuel_mix  Event_type density_fuel
% global N_Pu_239 mass_Pu_239 l_star  alpha_mod volume_cooling_core
% global Area_pin_internal dz_cond_fuel mass_graphite_half_core
% global surface_area_graphite area_cooling_pin graphite_height number_pins
% global volume_cooling_graphite_core dz_cond_grap radius_pin inner_radius
% global Area_pin_internal_cond flow_outlet_core flow_inlet_core
% global low_fin_height upper_fin_height A_flow_outlet_grid P_flow_outlet_grid
% global lower_cone_height upper_cone_height A_flow_inlet_grid P_flow_inlet_grid
% global volume_cooling_lower_conical volume_cooling_upper_conical

%% Parameter arguments passed
```

```
% Tinit_fuel = GV(1);
% Tinit_mod = GV(2);
% Max_reactivity = GV(3);
% Reactivity_add_rate = GV(4);
% N_238 = GV(5);
% N_235 = GV(6);
% surface_area_fuel = GV(7);
% volume_core = GV(8);
% pin_height = GV(9);
% inner_hex = GV(10);
% mass_U_235 = GV(11);
% mass_U_238 = GV(12);
% mass_fuel_mix = GV(13);
% Event_type = GV(14);
% density_fuel = GV(15);
% N_Pu_239 = GV(16);
% mass_Pu_239 = GV(17);
% l_star = GV(18);
% alpha_mod = GV(19);
% volume_cooling_core = GV(20);
% Area_pin_internal = GV(21);
% dz_cond_fuel = GV(22);
% mass_graphite_half_core = GV(23);
% surface_area_graphite = GV(24);
% area_cooling_pin = GV(25);
% graphite_height = GV(26);
% number_pins = GV(27);
% volume_cooling_graphite_core = GV(28);
% dz_cond_grap = GV(29);
% radius_pin = GV(30);
% inner_radius = GV(31);
% Area_pin_internal_cond = GV(32);
% flow_outlet_core = GV(33);
% flow_inlet_core = GV(34);
% lower_fin_height = GV(35);
% upper_fin_height = GV(36);
% A_flow_outlet_grid = GV(37);
% P_flow_outlet_grid = GV(38);
% lower_cone_height = GV(39);
% upper_cone_height = GV(40);
% A_flow_inlet_grid = GV(41);
% P_flow_inlet_grid = GV(42);
% volume_cooling_lower_conical = GV(43);
% volume_cooling_upper_conical = GV(44);
% lower_pin_below_gr GV(45)
% upper_pin_above_gr
% lower_fin_height ...
% lower_cyl_height
% upper_cyl_height
% mass 104 ss region GV(50)
% mass 105 ss region
% dz_lower_ss_cond
% dz_upper_ss_cond
% A_s_lower_fin
% A_s_upper_fin GV(55)
% A_s_lower_cyl
% A_s_upper_cyl
% A_s_lower_cone
% A_s_upper_cone

%% neutronics constants %%%

% Group Lambdas
% U235
U235_L_1 = .0127;
U235_L_2 = .0317;
U235_L_3 = .115;
U235_L_4 = .311;
U235_L_5 = 1.40;
U235_L_6 = 3.87;
```

```
% U238
U238_L_1 = .0132;
U238_L_2 = .0321;
U238_L_3 = .139;
U238_L_4 = .358;
U238_L_5 = 1.41;
U238_L_6 = 4.02;

% Pu239
Pu239_L_1 = .0129;
Pu239_L_2 = .0313;
Pu239_L_3 = .135;
Pu239_L_4 = .333;
Pu239_L_5 = 1.36;
Pu239_L_6 = 4.04;

% total delayed neutron fraction U238 [Weaver, 1968]
B_238 = .0157;
% group i fraction is B * relative abundance
B_1_238 = B_238 * .013; %group 1
B_2_238 = B_238 * .137;
B_3_238 = B_238 * .162;
B_4_238 = B_238 * .388;
B_5_238 = B_238 * .225;
B_6_238 = B_238 * .075;

% total delayed neutron fraction U235 [Weaver, 1968]
B_235 = .0065;
B_1_235 = B_235 * .038;
B_2_235 = B_235 * .213;
B_3_235 = B_235 * .188;
B_4_235 = B_235 * .407;
B_5_235 = B_235 * .128;
B_6_235 = B_235 * .026;

% total delayed neutron fraction Pu239
B_Pu_239 = .0026;
B_1_Pu_239 = B_Pu_239 * .038;
B_2_Pu_239 = B_Pu_239 * .280;
B_3_Pu_239 = B_Pu_239 * .216;
B_4_Pu_239 = B_Pu_239 * .328;
B_5_Pu_239 = B_Pu_239 * .103;
B_6_Pu_239 = B_Pu_239 * .035;

% define fission cross sections for precursor ratios
%fission cross section (m^2) [NIST]
sig_f_238 = 3.3e-28;

%fission cross section (m^2)
sig_f_235 = 584.4e-28;

% fission cross section (m^2)
sig_f_Pu_239 = 747.4e-28;

% Boltzmann constant (m2 kg s-2 K-1)
k_b = 1.38064852e-23;

% Mass of neutron (kg)
m_nu = 1.674927471e-27;

% neutron velocity (m/s)
 velocity_neutron = sqrt( 2 * k_b * (R(12) + 273.15) / m_nu );
% velocity_neutron = 2197;

% U235 Beta factor (atoms/s)
Beta_factor_U235 = velocity_neutron * sig_f_235 * GV(6);

% U238 Beta factor (atoms/s)
Beta_factor_U238 = velocity_neutron * sig_f_238 * GV(5);
```

```
% Pu239 Beta factor (atoms/s)
Beta_factor_Pu239 = velocity_neutron * sig_f_Pu_239 * GV(16);

% Beta of the DNP mix
B_1_mix = ( Beta_factor_U238 * B_1_238 + Beta_factor_U235 * B_1_235 ...
    + Beta_factor_Pu239 * B_1_Pu_239) ...
    / ( Beta_factor_U238 + Beta_factor_U235 + Beta_factor_Pu239);
B_2_mix = ( Beta_factor_U238 * B_2_238 + Beta_factor_U235 * B_2_235 ...
    + Beta_factor_Pu239 * B_2_Pu_239) ...
    / ( Beta_factor_U238 + Beta_factor_U235 + Beta_factor_Pu239);
B_3_mix = ( Beta_factor_U238 * B_3_238 + Beta_factor_U235 * B_3_235 ...
    + Beta_factor_Pu239 * B_3_Pu_239) ...
    / ( Beta_factor_U238 + Beta_factor_U235 + Beta_factor_Pu239);
B_4_mix = ( Beta_factor_U238 * B_4_238 + Beta_factor_U235 * B_4_235 ...
    + Beta_factor_Pu239 * B_4_Pu_239) ...
    / ( Beta_factor_U238 + Beta_factor_U235 + Beta_factor_Pu239);
B_5_mix = ( Beta_factor_U238 * B_5_238 + Beta_factor_U235 * B_5_235 ...
    + Beta_factor_Pu239 * B_5_Pu_239) ...
    / ( Beta_factor_U238 + Beta_factor_U235 + Beta_factor_Pu239);
B_6_mix = ( Beta_factor_U238 * B_6_238 + Beta_factor_U235 * B_6_235 ...
    + Beta_factor_Pu239 * B_6_Pu_239) ...
    / ( Beta_factor_U238 + Beta_factor_U235 + Beta_factor_Pu239);

% Lambda of the DNP mix
lambda_1 = ( Beta_factor_U238 * U238_L_1 + Beta_factor_U235 * U235_L_1 ...
    + Beta_factor_Pu239 * Pu239_L_1) ...
    / ( Beta_factor_U238 + Beta_factor_U235 + Beta_factor_Pu239);
lambda_2 = ( Beta_factor_U238 * U238_L_2 + Beta_factor_U235 * U235_L_2 ...
    + Beta_factor_Pu239 * Pu239_L_2) ...
    / ( Beta_factor_U238 + Beta_factor_U235 + Beta_factor_Pu239);
lambda_3 = ( Beta_factor_U238 * U238_L_3 + Beta_factor_U235 * U235_L_3 ...
    + Beta_factor_Pu239 * Pu239_L_3) ...
    / ( Beta_factor_U238 + Beta_factor_U235 + Beta_factor_Pu239);
lambda_4 = ( Beta_factor_U238 * U238_L_4 + Beta_factor_U235 * U235_L_4 ...
    + Beta_factor_Pu239 * Pu239_L_4) ...
    / ( Beta_factor_U238 + Beta_factor_U235 + Beta_factor_Pu239);
lambda_5 = ( Beta_factor_U238 * U238_L_5 + Beta_factor_U235 * U235_L_5 ...
    + Beta_factor_Pu239 * Pu239_L_5) ...
    / ( Beta_factor_U238 + Beta_factor_U235 + Beta_factor_Pu239);
lambda_6 = ( Beta_factor_U238 * U238_L_6 + Beta_factor_U235 * U235_L_6 ...
    + Beta_factor_Pu239 * Pu239_L_6) ...
    / ( Beta_factor_U238 + Beta_factor_U235 + Beta_factor_Pu239);

% Beta mix
% Beff correction factor based on Beff/B ratio for U235 .007/,0065
Beta_correction = 1.076923076923077;

% Beta mix
Beta_mix = B_1_mix + B_2_mix + B_3_mix + B_4_mix + B_5_mix + B_6_mix;

% Beff mix
Beff_mix = Beta_mix * Beta_correction;

%prompt neutron lifetime (s) [UT SAR]
% l_star = 51.9e-6;

% 2 Ci AmBe source (atoms/s/m^3)
AmBe_source = 7.4e10;

% U238 spontaneous fission (atoms/s/m^3)
U238_spontaneous = 1.80e-2 * GV(12) * 1000; % factor in (n/g-s)

% U235 spontaneous fission (atoms/s/m^3)
U235_spontaneous = 7.43e-4 * GV(11) * 1000; % factor in (n/g-s)

% Pu239 spontaneous fission (atoms/s/m^3)
Pu_239_spontaneuos = 2.30e-2 * GV(17) * 1000;

%% Decay Heat Constants %%%
```

```
% Decay (1/s)
% U235
H_lambda_235(1) = 2.2138e1;
H_lambda_235(2) = 5.1587e-1;
H_lambda_235(3) = 1.9594e-1;
H_lambda_235(4) = 1.0314e-1;
H_lambda_235(5) = 3.3656e-2;
H_lambda_235(6) = 1.1681e-2;
H_lambda_235(7) = 3.5870e-3;
H_lambda_235(8) = 1.3930e-3;
H_lambda_235(9) = 6.2630e-4;
H_lambda_235(10) = 1.8906e-4;
H_lambda_235(11) = 5.4988e-5;
H_lambda_235(12) = 2.0958e-5;
H_lambda_235(13) = 1.0010e-5;
H_lambda_235(14) = 2.5438e-6;
H_lambda_235(15) = 6.6361e-7;
H_lambda_235(16) = 1.2290e-7;
H_lambda_235(17) = 2.7213e-8;
H_lambda_235(18) = 4.3714e-9;
H_lambda_235(19) = 7.5780e-10;
H_lambda_235(20) = 2.4786e-10;
H_lambda_235(21) = 2.2384e-13;
H_lambda_235(22) = 2.4600e-14;
H_lambda_235(23) = 1.5699e-14;

% U238
H_lambda_238(1) = 3.2881e00;
H_lambda_238(2) = 9.3805e-1;
H_lambda_238(3) = 3.7073e-1;
H_lambda_238(4) = 1.1118e-1;
H_lambda_238(5) = 3.6143e-2;
H_lambda_238(6) = 1.3272e-2;
H_lambda_238(7) = 5.0133e-3;
H_lambda_238(8) = 1.3655e-3;
H_lambda_238(9) = 5.5158e-4;
H_lambda_238(10) = 1.7873e-4;
H_lambda_238(11) = 4.9032e-5;
H_lambda_238(12) = 1.7058e-5;
H_lambda_238(13) = 7.0465e-6;
H_lambda_238(14) = 2.3190e-6;
H_lambda_238(15) = 6.4480e-7;
H_lambda_238(16) = 1.2649e-7;
H_lambda_238(17) = 2.5548e-8;
H_lambda_238(18) = 8.4782e-9;
H_lambda_238(19) = 7.5130e-10;
H_lambda_238(20) = 2.4188e-10;
H_lambda_238(21) = 2.2739e-13;
H_lambda_238(22) = 9.0536e-14;
H_lambda_238(23) = 5.6098e-15;

% Pu239
H_lambda_Pu239(1) = 1.0020e1;
H_lambda_Pu239(2) = 6.4330e-1;
H_lambda_Pu239(3) = 2.1860e-1;
H_lambda_Pu239(4) = 1.0040e-1;
H_lambda_Pu239(5) = 3.7280e-2;
H_lambda_Pu239(6) = 1.4350e-2;
H_lambda_Pu239(7) = 4.5490e-3;
H_lambda_Pu239(8) = 1.3280e-3;
H_lambda_Pu239(9) = 5.3560e-4;
H_lambda_Pu239(10) = 1.7300e-4;
H_lambda_Pu239(11) = 4.8810e-5;
H_lambda_Pu239(12) = 2.0060e-5;
H_lambda_Pu239(13) = 8.3190e-6;
H_lambda_Pu239(14) = 2.3580e-6;
H_lambda_Pu239(15) = 6.4500e-7;
H_lambda_Pu239(16) = 1.2780e-7;
H_lambda_Pu239(17) = 2.4660e-8;
H_lambda_Pu239(18) = 9.3780e-9;
```

```
H_lambda_Pu239(19) = 7.4500e-10;
H_lambda_Pu239(20) = 2.4260e-10;
H_lambda_Pu239(21) = 2.2100e-13;
H_lambda_Pu239(22) = 2.6400e-14;
H_lambda_Pu239(23) = 1.3800e-14;

% Power fraction
% U235
H_fraction_235(1) = 6.5057e-1;
H_fraction_235(2) = 5.1264e-1;
H_fraction_235(3) = 2.4384e-1;
H_fraction_235(4) = 1.3850e-1;
H_fraction_235(5) = 5.5440e-2;
H_fraction_235(6) = 2.2225e-2;
H_fraction_235(7) = 3.3088e-3;
H_fraction_235(8) = 9.3015e-4;
H_fraction_235(9) = 8.0943e-4;
H_fraction_235(10) = 1.9567e-4;
H_fraction_235(11) = 3.2535e-5;
H_fraction_235(12) = 7.5595e-6;
H_fraction_235(13) = 2.5232e-6;
H_fraction_235(14) = 4.9948e-7;
H_fraction_235(15) = 1.8531e-7;
H_fraction_235(16) = 2.6608e-8;
H_fraction_235(17) = 2.2398e-9;
H_fraction_235(18) = 8.1641e-12;
H_fraction_235(19) = 8.7797e-11;
H_fraction_235(20) = 2.5131e-14;
H_fraction_235(21) = 3.2176e-16;
H_fraction_235(22) = 4.5038e-17;
H_fraction_235(23) = 7.4791e-17;

% U238
H_fraction_238(1) = 1.2311e00;
H_fraction_238(2) = 1.1486e00;
H_fraction_238(3) = 7.0701e-1;
H_fraction_238(4) = 2.5209e-1;
H_fraction_238(5) = 7.1870e-2;
H_fraction_238(6) = 2.8291e-2;
H_fraction_238(7) = 6.8382e-3;
H_fraction_238(8) = 1.2322e-3;
H_fraction_238(9) = 6.8409e-4;
H_fraction_238(10) = 1.6975e-4;
H_fraction_238(11) = 2.4182e-5;
H_fraction_238(12) = 6.6356e-6;
H_fraction_238(13) = 1.0075e-6;
H_fraction_238(14) = 4.9894e-7;
H_fraction_238(15) = 1.6352e-7;
H_fraction_238(16) = 2.3355e-8;
H_fraction_238(17) = 2.8094e-9;
H_fraction_238(18) = 3.6236e-11;
H_fraction_238(19) = 6.4577e-11;
H_fraction_238(20) = 4.4963e-14;
H_fraction_238(21) = 3.6654e-16;
H_fraction_238(22) = 5.6293e-17;
H_fraction_238(23) = 7.1602e-17;

% Pu239
H_fraction_Pu239(1) = 2.0830e-1;
H_fraction_Pu239(2) = 3.8530e-1;
H_fraction_Pu239(3) = 2.2130e-1;
H_fraction_Pu239(4) = 9.4600e-2;
H_fraction_Pu239(5) = 3.5310e-2;
H_fraction_Pu239(6) = 2.2920e-2;
H_fraction_Pu239(7) = 3.9460e-3;
H_fraction_Pu239(8) = 1.3170e-3;
H_fraction_Pu239(9) = 7.0520e-4;
H_fraction_Pu239(10) = 1.4320e-4;
H_fraction_Pu239(11) = 1.7650e-5;
H_fraction_Pu239(12) = 7.3470e-6;
```

```
H_fraction_Pu239(13) = 1.7470e-6;
H_fraction_Pu239(14) = 5.4810e-7;
H_fraction_Pu239(15) = 1.6710e-7;
H_fraction_Pu239(16) = 2.1120e-8;
H_fraction_Pu239(17) = 2.9960e-9;
H_fraction_Pu239(18) = 5.1070e-11;
H_fraction_Pu239(19) = 5.7300e-11;
H_fraction_Pu239(20) = 4.1380e-14;
H_fraction_Pu239(21) = 1.0880e-15;
H_fraction_Pu239(22) = 2.4540e-17;
H_fraction_Pu239(23) = 7.5570e-17;

%% State dependent functions
% ALPHA T
% Temperature dependent alpha T of fuel from UT TRIGA Lab experiment
% dp/dT = -8.14475e-8 * T^2 - 2.53543e-5 * T - 1.56396e-4 ->
% alpha_t_T = -1.62895e-7 * T - 2.53543e-5
% alpha_fuel_T = -1.62895e-7 * R(12) - 2.53543e-5
% alpha_fuel_T = -1.65149e-7 * R(12) - 2.5056e-5;
% alpha_fuel_T = -6.9235e-5;
% alpha_fuel_T = -5.6196e-5;

% constant Alpha_T from SAR
% alpha_fuel_T = -1e-4;

% Curve fits to GA chart from SAR
% 6th order poly
% alpha_fuel_T = -1.9631e-21*R(12)^6 + 6.5610e-18*R(12)^5 - 8.210e-15*R(12)^4 ...
%    + 4.5051e-12*R(12)^3 - 7.8572e-10*R(12)^2 - 1.1061e-7*R(12) - 7.4965e-5;
% 4th order poly
  alpha_fuel_T = 1.4990e-16*R(12)^4 - 5.3734e-13*R(12)^3 + 6.5947e-10*R(12)^2 ...
    - 2.8159e-7*R(12) - 6.9571e-5;
% 2nd order poly
% alpha_fuel_T = 3.1802e-10*R(12)^2 - 2.1800e-7*R(12) - 7.2126e-5;

% volumetric heat capacity from Simnad (J/ m3 K)
cp_fuel_vol = (2.04 + 4.17e-3 * R(12) ) * 1e6;

% Convert to specific heat ( J / kg K )
cp_fuel = cp_fuel_vol / GV(15);

% find keff
current_keff = 1 / ( 1 - ( R(11) + (alpha_fuel_T * R(12) + GV(19) * R(13) ) ) );

% keff adjusted neutron lifetime (s)
 A = GV(18) / current_keff;
% A = l_star;

% Instantaneous Power (W)
% U235 contribution
P_inst_235 = R(1) * velocity_neutron * (200e6 * 1.602677e-19) * GV(6) ...
    * sig_f_235 * GV(8);

% U238 contribution (W)
P_inst_238 = R(1) * velocity_neutron * (200e6 * 1.602677e-19) * GV(5) ...
    * sig_f_238 * GV(8);

% Pu239 contribution (W)
P_inst_Pu239 = R(1) * velocity_neutron * (200e6 * 1.602677e-19) * GV(16) ...
    * sig_f_Pu_239 * GV(8);

% Total instantaneous (W)
P_inst = P_inst_235 + P_inst_238 + P_inst_Pu239;

% specific heat capacity of graphite (cal / g C) [Entegris, inc.]
cp_graphite_cal_91 = .10795e8 * R(91)^-3 - .61257e5 * R(91)^-2 ...
    + .30795e-4 * R(91) + .44391;
cp_graphite_91 = 4183.995381 * cp_graphite_cal_91; %Conversion to J / kg K

cp_graphite_cal_93 = .10795e8 * R(93)^-3 - .61257e5 * R(93)^-2 ...
```

```
    + .30795e-4 * R(93) + .44391;
cp_graphite_93 = 4183.995381 * cp_graphite_cal_93;

%% Moderator state equations

% Temperature Dependent Density (kg/m^3) [ddbst.de, 273K-648K]
A_cp = .14395;
B_cp = .0112;
C_cp = 649.727;
D_cp = .05107;
low_temp = R(94);
c_temp = R(101);
f_temp = R(12);

% density of outlet water/core area water (kg/m^3)
density_water_13 = A_cp / ( B_cp^(1 + (1 - (R(13) + 273.15)/C_cp)^D_cp));
density_water_92 = A_cp / ( B_cp^(1 + (1 - (R(92) + 273.15)/C_cp)^D_cp));
density_water_94 = A_cp / ( B_cp^(1 + (1 - (R(94) + 273.15)/C_cp)^D_cp));
density_water_102 = A_cp / ( B_cp^(1 + (1 - (R(102) + 273.15)/C_cp)^D_cp));
density_water_103 = A_cp / ( B_cp^(1 + (1 - (R(103) + 273.15)/C_cp)^D_cp));

% density of inlet water, bulk tank (kg/m^3)
density_water_0 =  A_cp / ( B_cp^(1 + (1 - (GV(2) + 273.15)/C_cp)^D_cp));

% Mass of water (kg)
mass_cooling_water_fuel = GV(20) * density_water_13;
mass_cooling_water_graphite_92 = GV(28) * density_water_92;
mass_cooling_water_graphite_94 = GV(28) * density_water_94;
mass_cooling_water_102 = GV(43) * density_water_102;
mass_cooling_water_103 = GV(44) * density_water_103;

% specific heat of inlet water (KJ/kg K)
cp_H2O_KJ_0 = 3.16744e-10 * GV(2)^4 - 1.05772e-7 * GV(2)^3 ...
    + 2.35330e-5 * GV(2)^2 - 1.47670e-3 * GV(2) + 4.20617e0;

% Convert to J/kg
cp_H2O_0 = cp_H2O_KJ_0 * 1000;

% Temperature dependent Specific heat (J/kgK) [steam tables]
cp_H2O_92 = ( 3.16744e-10 * R(92)^4 - 1.05772e-7 * R(92)^3 ...
    + 2.35330e-5 * R(92)^2 - 1.47670e-3 * R(92) + 4.20617e0 ) * 1000;

cp_H2O_13 = ( 3.16744e-10 * R(13)^4 - 1.05772e-7 * R(13)^3 ...
    + 2.35330e-5 * R(13)^2 - 1.47670e-3 * R(13) + 4.20617e0 ) * 1000;

cp_H2O_94 = ( 3.16744e-10 * R(94)^4 - 1.05772e-7 * R(94)^3 ...
    + 2.35330e-5 * R(94)^2 - 1.47670e-3 * R(94) + 4.20617e0 ) * 1000;

cp_H2O_102 = ( 3.16744e-10 * R(102)^4 - 1.05772e-7 * R(102)^3 ...
    + 2.35330e-5 * R(102)^2 - 1.47670e-3 * R(102) + 4.20617e0 ) * 1000;

cp_H2O_103 = ( 3.16744e-10 * R(103)^4 - 1.05772e-7 * R(103)^3 ...
    + 2.35330e-5 * R(103)^2 - 1.47670e-3 * R(103) + 4.20617e0 ) * 1000;

% Use the Plume formulas from Dartmouth Paper/UT LOCA to find current mass flow
% gravity (m/s^2)
gravity = 9.8066;


% Kinetic Viscosity [30C](m^2/s)
kinetic_viscosity_water = .801e-6;

% Dynamic viscosity [30C] (Ns/m^2)
dyn_vis_water = 7.98e-4;

% Expansion Coefficient (1/K)
B_water = .303e-3;

% Thermal conductivity [NIST](W/mK)
k_water_13 = -1.48445 + 4.12292 * ((R(13) + 273.15)/298.15) ...
```

```
    - 1.63866 * ((R(13) + 273.15)/298.15)^2;
k_water_92 = -1.48445 + 4.12292 * ((R(92) + 273.15)/298.15) ...
    - 1.63866 * ((R(92) + 273.15)/298.15)^2;
k_water_94 = -1.48445 + 4.12292 * ((R(94) + 273.15)/298.15) ...
    - 1.63866 * ((R(94) + 273.15)/298.15)^2;
k_water_102 = -1.48445 + 4.12292 * ((R(102) + 273.15)/298.15) ...
    - 1.63866 * ((R(102) + 273.15)/298.15)^2;
k_water_103 = -1.48445 + 4.12292 * ((R(103) + 273.15)/298.15) ...
    - 1.63866 * ((R(103) + 273.15)/298.15)^2;
k_fuel = 17.5730;  % [Simnad]
k_graphite = 112.4;

% Prandtl Number (Pr)
Pr_102 = dyn_vis_water * cp_H2O_102 / k_water_102;
Pr_92 = dyn_vis_water * cp_H2O_92 / k_water_92;
Pr_13 = dyn_vis_water * cp_H2O_13 / k_water_13;
Pr_94 = dyn_vis_water * cp_H2O_94 / k_water_94;
Pr_103 = dyn_vis_water * cp_H2O_103 / k_water_103;

% Thermal diffusivity (m^2/s)
% therm_diff_water = 0.143e-6;
therm_diff_water_13 = k_water_13 / ( density_water_13 * cp_H2O_13 );
therm_diff_water_92 = k_water_92 / ( density_water_92 * cp_H2O_92 );
therm_diff_water_94 = k_water_94 / ( density_water_94 * cp_H2O_94 );

% find this loops transient factor (m(T) * cp(T)) [J/K, J/C]
trans_factor_13 = mass_cooling_water_fuel * cp_H2O_13;
trans_factor_92 = mass_cooling_water_graphite_92 * cp_H2O_92;
trans_factor_94 = mass_cooling_water_graphite_94 * cp_H2O_94;

% find the mass flow rate of inlet (kg/s) rho(T) * A_opening * velocity
% m_dot = R(95) * density_water_13 * area_cooling_pin * number_pins;

%%  Enthalpy (J/kgK)
 % find current specific enthalpy h = cp * (T(t) - T_ref) + h_ref;
enthalpy_ref = 9007; % J/kg @ 0C saturated

% inlet to core from bulk tank
h_0 = cp_H2O_0 * GV(2) + enthalpy_ref;

% out of lower graphite
h_1 = cp_H2O_92 * R(92) + enthalpy_ref;

% out of fuel region
h_2 = cp_H2O_13 * R(13) + enthalpy_ref;

% out of upper graphite
h_3 = cp_H2O_94 * R(94) + enthalpy_ref;

% out of lower fin region
h_102 = cp_H2O_102 * R(102) + enthalpy_ref;

% out of upper fin region
h_103 = cp_H2O_103 * R(103) + enthalpy_ref;

%% Gas and SS expansion coefficients [ UT LOCA]
% Mass (kg)
mass_gas_fuel = .08375 * pi * ( (GV(30) - .000508)^2 - (GV(30) - .000508 - 1.310588235294116e-04)^2 ) * GV(9);
mass_gas_gr = .08375 * pi * ( (GV(30) - .000508)^2 - (GV(30) - .000508 - 1.310588235294116e-04)^2 ) * GV(26);
mass_clad_fuel = 7740 * pi * ( GV(30)^2 - (GV(30) - .000508)^2 ) * GV(9);
mass_clad_gr = 7740 * pi * ( GV(30)^2 - (GV(30) - .000508)^2 ) * GV(26);

% cp (J/kg K)
cp_gas = 14.53e3;
cp_clad = 500;

% dz for conduction (m)
inner_radius = 0.003175;
dz_gas = 1.310588235294116e-04/2;
dz_clad = 0.000508/2;
```

```
dz_fuel_unit = ( GV(30) - .000508 - 1.310588235294116e-04 - inner_radius ) /2;

% Gas heat transfer from KSU SAR (W/m^2 K) [Whaley]
h_gas = 2.84e3;

% Thermal conductivity (W/m K)
k_gas = h_gas * 1.310588235294116e-04; % k ~ h/dx
k_clad = 16.2;

% Area (m^2)
A_gas = pi * ( (GV(30) - .000508)^2 - (GV(30) - .000508 - 1.310588235294116e-04)^2 );
A_clad = pi * ( (GV(30) )^2 - (GV(30) - .000508)^2 );

% Surface areas (m^2)
surface_gas_out_grap = 2 * pi * (GV(30) - .000508) * GV(26);
surface_gas_out_fuel = 2 * pi * (GV(30) - .000508) * GV(9);
surface_gas_in_grap = 2 * pi * (GV(30) - .000508 - 1.310588235294116e-04) * GV(26);
surface_gas_in_fuel = 2 * pi * (GV(30) - .000508 - 1.310588235294116e-04) * GV(9);

%% Momentum balance
% P is a property vector to pass
Pf = [ density_water_13 density_water_92 density_water_94 GV(25) GV(33) ...
    GV(34) therm_diff_water_13 therm_diff_water_92 therm_diff_water_94 ...
    gravity GV(9) GV(26) GV(30) GV(10) GV(24) GV(7) density_water_102 ...
    density_water_103 GV(35) GV(36) GV(37) GV(38) GV(39) GV(40) GV(43) ...
    GV(44) density_water_0 GV(41:59) ];

% Balance function
% m1 - inlet to lower grid
% m2 - lower cone to graphite
% m3 - graphite to fuel
% m4 - fuel to upper graphite
% m5 - upper graphite to upper cone
% m6 - outlet upper grid

% mdot per pin
m_dot_pin = Coolant_properties_1_0( R, Pf );

% mdot
m_dot = m_dot_pin * GV(27);

if ~isreal(m_dot)
   isreal(m_dot)
   m_dot
   return;
end
% Heat transfer coefficient

% lower region

% Fin effects
% Dimensionless numbers
if ( R(104) - R(102)) > 0
   Gr_102_fin = gravity * B_water * GV(35)^3 *  ( R(104) - R(102)) / kinetic_viscosity_water^2;
else
   Gr_102_fin = 0;
end

Ra_102 = Pr_102 * Gr_102_fin;

Nu_102_fin = .59 * Ra_102^.25;

% fin heat transfer coefficient (W/m^2 K)
h_bar_102_fin = Nu_102_fin * k_clad / GV(35);

% efficiency (calculated from mean efficiency effect at 30C)
eff_lower_fin = .2130;

% Heat lost in region 104 from the fin (W)
% q_102_fin = eff_lower_fin * h_bar_102_fin * GV(54) * ( R(104) - R(102) );
```

```
% upper region

% Fin effects
% Dimensionless numbers
if ( R(105) - R(103) ) > 0
   Gr_103_fin = gravity * B_water * GV(35)^3 * ( R(105) - R(103) ) / kinetic_viscosity_water^2;
else
   Gr_103_fin = 0;
end

Ra_103 = Pr_103 * Gr_103_fin;

Nu_103_fin = .59 * Ra_103^.25;

% fin heat transfer coefficient (W/m^2 K)
h_bar_103_fin = Nu_103_fin * k_clad / GV(36);

% efficiency (calculated from mean efficiency effect at 30C)
eff_upper_fin = .1781;

% Cylindrical sections
% cylindrical Nusselt
Nu_102_cyl =  ( .825 + ( .387 * Ra_102^(1/6) ) / ( 1 + ( .492/Pr_102)^(9/16) )^(8/27))^2; %penn.edu
Nu_103_cyl =  ( .825 + ( .387 * Ra_103^(1/6) ) / ( 1 + ( .492/Pr_103)^(9/16) )^(8/27))^2; %penn.edu

% Cyl heat transfer coefficient (W/m^2 K)
h_bar_102_cyl = Nu_102_cyl * k_water_102 / GV(48);
h_bar_103_cyl = Nu_103_cyl * k_water_103 / GV(49);

% Conical sections
% Conical section Nusselt [Yaser]
Nu_102_cone = 2.0963 + .669 * Gr_102_fin^.25 * Pr_102^(1/3);
Nu_103_cone = 2.0963 + .669 * Gr_103_fin^.25 * Pr_103^(1/3);

% characteristic length (m)
L_102 = sqrt( GV(39)^2 + GV(30)^2);
L_103 = sqrt( GV(40)^2 + GV(30)^2);

% heat transfer coefficient (W/m^2 K)
h_bar_102_cone = Nu_102_cone * k_water_102 / L_102;
h_bar_103_cone = Nu_103_cone * k_water_103 / L_103;

%% heat transfer coefficient
% Grashof Number [Clarkson.edu]
% Gr_s = reduced_gravity * pin_height^3 / kinetic_viscosity_water^2

% Find the Ra_s [ Kaminski ]
% Ra_s = Gr_s * Pr_water;
if ( R(99) - R(92) ) > 0
   Ra_L_92 =  ( gravity * B_water ) / ( kinetic_viscosity_water * therm_diff_water_92 ) ...
      * ( R(99) - R(92) ) * GV(26)^3;
else
   Ra_L_92 = 0;
end

if ( R(100) - R(13) ) > 0
   Ra_L_13 =  ( gravity * B_water ) / ( kinetic_viscosity_water * therm_diff_water_13 ) ...
      * ( R(100) - R(13) ) * GV(9)^3;
else
   Ra_L_13 = 0;
end

if ( R(101) - R(94) ) > 0
   Ra_L_94 =  ( gravity * B_water ) / ( kinetic_viscosity_water * therm_diff_water_94 ) ...
      * ( R(101) - R(94) ) * GV(26)^3;
else
   Ra_L_94 = 0;
end
```

```
% Nusselt number
% external free convection vertical cylinder
Nu_L_92 = ( .825 + ( .387 * Ra_L_92^(1/6) ) / ( 1 + ( .492/Pr_92)^(9/16) )^(8/27))^2; %penn.edu
Nu_L_13 = ( .825 + ( .387 * Ra_L_13^(1/6) ) / ( 1 + ( .492/Pr_13)^(9/16) )^(8/27))^2; %penn.edu
Nu_L_94 = ( .825 + ( .387 * Ra_L_94^(1/6) ) / ( 1 + ( .492/Pr_94)^(9/16) )^(8/27))^2; %penn.edu

% Heat transfer coefficient ( W/ m^2 K )
h_bar_92 = Nu_L_92 * k_water_92 / GV(26);
h_bar_13 = Nu_L_13 * k_water_13 / GV(9);
h_bar_94 = Nu_L_94 * k_water_94 / GV(26);

%% Delayed Power State Function (W)
% U235 delayed (W)
P_d_235 =  ( P_inst_235 / 200 ) * sum( H_fraction_235 ./ H_lambda_235 );

% U238 delayed (W)
P_d_238 =  ( P_inst_238 / 200 ) * sum( H_fraction_238 ./ H_lambda_238 );

% Pu239 delayed (W)
P_d_Pu_239 = ( P_inst_Pu239 / 200 ) * sum( H_fraction_Pu239 ./ H_lambda_Pu239 );

P_delay = P_d_235 + P_d_238 + P_d_Pu_239;

%% Total power by isotope (W)
% U235
P_i_235 = P_inst_235 + P_d_235;

% U238
P_i_238 = P_inst_238 + P_d_238;

% Pu239
P_i_Pu239 = P_inst_Pu239 + P_d_Pu_239;

P_eff = P_inst - P_delay + sum(R(21:66));

%% Differentials
% Reactor Kinetics
% overall in hour
R_dot(1) = ( R(11) + alpha_fuel_T * ( R(12) - GV(1) ) ...
   + GV(19) * ( R(13) - GV(2) ) - Beff_mix ) / A * R(1) ...
   + lambda_1 * R(2) ...
   + lambda_2 * R(3) ...
   + lambda_3 * R(4) ...
   + lambda_4 * R(5) ...
   + lambda_5 * R(6) ...
   + lambda_6 * R(7) ...
   + AmBe_source ...
   + U238_spontaneous + U235_spontaneous + Pu_239_spontaneuos; % ...
   % - R(1) * velocity_neutron * N_Sm * sig_a_Sm;
   % - R(1) * velocity_neutron * N_Zr * sig_a_Zr;

% DNP groups
R_dot(2) = (B_1_mix / A) * R(1) - lambda_1 * R(2);
R_dot(3) = (B_2_mix / A) * R(1) - lambda_2 * R(3);
R_dot(4) = (B_3_mix / A) * R(1) - lambda_3 * R(4);
R_dot(5) = (B_4_mix / A) * R(1) - lambda_4 * R(5);
R_dot(6) = (B_5_mix / A) * R(1) - lambda_5 * R(6);
R_dot(7) = (B_6_mix / A) * R(1) - lambda_6 * R(7);

% Source strength
R_dot(8) = 0;
R_dot(9) = 0;
R_dot(10) = 0;

% Reactivities
% event reactivity
switch SV(1)
   case 'Pulse'
      R_dot(11) = 0;
```

```
    case 'Rod Withdraw'
        if R(11) < GV(3) * Beff_mix
            R_dot(11) = GV(4);

        elseif R(11) >= GV(3) * Beff_mix
            R_dot(11) = 0;

        else
            display (' Error in Reactivity Addition ');
        end

    otherwise
        R_dot(11) = 0;
end

% Fuel Temperature
R_dot(12) = ( 1 / ( GV(13) * cp_fuel ) ) * ( P_eff ...
    + ( R(91) - R(12) ) / ( GV(22)/k_fuel + GV(29)/k_graphite ) * GV(32) ...
    + ( R(93) - R(12) ) / ( GV(22)/k_fuel + GV(29)/k_graphite ) * GV(32) ...
    + ( R(97) - R(12) ) / ( dz_fuel_unit/k_fuel + dz_gas/k_gas ) *  surface_gas_in_fuel );

% Moderator Temperature
R_dot(13) = (1 / trans_factor_13) * ( h_bar_13 * GV(7) * (R(100) - R(13)) ...
        + m_dot(3) * ( h_1 + gravity * (GV(26) + GV(45)) ) ...
        - m_dot(4) * ( h_2 + gravity * (GV(26) + GV(9) + GV(45)) ) );

% Changes in Beta effective
% [ current model will be 0 ]
R_dot(14) = 0;
R_dot(15) = 0;
R_dot(16) = 0;
R_dot(17) = 0;
R_dot(18) = 0;
R_dot(19) = 0;
R_dot(20) = 0;

%% Decay Heat calculations
% Delayed heat from U235 antecedes
R_dot(21) = H_fraction_235(1) * P_i_235 / 200 - H_lambda_235(1) * R(21);
R_dot(22) = H_fraction_235(2) * P_i_235 / 200 - H_lambda_235(2) * R(22);
R_dot(23) = H_fraction_235(3) * P_i_235 / 200 - H_lambda_235(3) * R(23);
R_dot(24) = H_fraction_235(4) * P_i_235 / 200 - H_lambda_235(4) * R(24);
R_dot(25) = H_fraction_235(5) * P_i_235 / 200 - H_lambda_235(5) * R(25);
R_dot(26) = H_fraction_235(6) * P_i_235 / 200 - H_lambda_235(6) * R(26);
R_dot(27) = H_fraction_235(7) * P_i_235 / 200 - H_lambda_235(7) * R(27);
R_dot(28) = H_fraction_235(8) * P_i_235 / 200 - H_lambda_235(8) * R(28);
R_dot(29) = H_fraction_235(9) * P_i_235 / 200 - H_lambda_235(9) * R(29);
R_dot(30) = H_fraction_235(10) * P_i_235 / 200 - H_lambda_235(10) * R(30);
R_dot(31) = H_fraction_235(11) * P_i_235 / 200 - H_lambda_235(11) * R(31);
R_dot(32) = H_fraction_235(12) * P_i_235 / 200 - H_lambda_235(12) * R(32);
R_dot(33) = H_fraction_235(13) * P_i_235 / 200 - H_lambda_235(13) * R(33);
R_dot(34) = H_fraction_235(14) * P_i_235 / 200 - H_lambda_235(14) * R(34);
R_dot(35) = H_fraction_235(15) * P_i_235 / 200 - H_lambda_235(15) * R(35);
R_dot(36) = H_fraction_235(16) * P_i_235 / 200 - H_lambda_235(16) * R(36);
R_dot(37) = H_fraction_235(17) * P_i_235 / 200 - H_lambda_235(17) * R(37);
R_dot(38) = H_fraction_235(18) * P_i_235 / 200 - H_lambda_235(18) * R(38);
R_dot(39) = H_fraction_235(19) * P_i_235 / 200 - H_lambda_235(19) * R(39);
R_dot(40) = H_fraction_235(20) * P_i_235 / 200 - H_lambda_235(20) * R(40);
R_dot(41) = H_fraction_235(21) * P_i_235 / 200 - H_lambda_235(21) * R(41);
R_dot(42) = H_fraction_235(22) * P_i_235 / 200 - H_lambda_235(22) * R(42);
R_dot(43) = H_fraction_235(23) * P_i_235 / 200 - H_lambda_235(23) * R(43);

% Delayed heat from U238 antecedes
R_dot(44) = H_fraction_238(1) * P_i_238 / 200 - H_lambda_238(1) * R(44);
R_dot(45) = H_fraction_238(2) * P_i_238 / 200 - H_lambda_238(2) * R(45);
R_dot(46) = H_fraction_238(3) * P_i_238 / 200 - H_lambda_238(3) * R(46);
R_dot(47) = H_fraction_238(4) * P_i_238 / 200 - H_lambda_238(4) * R(47);
R_dot(48) = H_fraction_238(5) * P_i_238 / 200 - H_lambda_238(5) * R(48);
R_dot(49) = H_fraction_238(6) * P_i_238 / 200 - H_lambda_238(6) * R(49);
R_dot(50) = H_fraction_238(7) * P_i_238 / 200 - H_lambda_238(7) * R(50);
```

```
R_dot(51) = H_fraction_238(8) * P_i_238 / 200 - H_lambda_238(8) * R(51);
R_dot(52) = H_fraction_238(9) * P_i_238 / 200 - H_lambda_238(9) * R(52);
R_dot(53) = H_fraction_238(10) * P_i_238 / 200 - H_lambda_238(10) * R(53);
R_dot(54) = H_fraction_238(11) * P_i_238 / 200 - H_lambda_238(11) * R(54);
R_dot(55) = H_fraction_238(12) * P_i_238 / 200 - H_lambda_238(12) * R(55);
R_dot(56) = H_fraction_238(13) * P_i_238 / 200 - H_lambda_238(13) * R(56);
R_dot(57) = H_fraction_238(14) * P_i_238 / 200 - H_lambda_238(14) * R(57);
R_dot(58) = H_fraction_238(15) * P_i_238 / 200 - H_lambda_238(15) * R(58);
R_dot(59) = H_fraction_238(16) * P_i_238 / 200 - H_lambda_238(16) * R(59);
R_dot(60) = H_fraction_238(17) * P_i_238 / 200 - H_lambda_238(17) * R(60);
R_dot(61) = H_fraction_238(18) * P_i_238 / 200 - H_lambda_238(18) * R(61);
R_dot(62) = H_fraction_238(19) * P_i_238 / 200 - H_lambda_238(19) * R(62);
R_dot(63) = H_fraction_238(20) * P_i_238 / 200 - H_lambda_238(20) * R(63);
R_dot(64) = H_fraction_238(21) * P_i_238 / 200 - H_lambda_238(21) * R(64);
R_dot(65) = H_fraction_238(22) * P_i_238 / 200 - H_lambda_238(22) * R(65);
R_dot(66) = H_fraction_238(23) * P_i_238 / 200 - H_lambda_238(23) * R(66);

R_dot(68) = H_fraction_Pu239(1) * P_i_Pu239 / 200 - H_lambda_Pu239(1) * R(68);
R_dot(69) = H_fraction_Pu239(2) * P_i_Pu239 / 200 - H_lambda_Pu239(2) * R(69);
R_dot(70) = H_fraction_Pu239(3) * P_i_Pu239 / 200 - H_lambda_Pu239(3) * R(70);
R_dot(71) = H_fraction_Pu239(4) * P_i_Pu239 / 200 - H_lambda_Pu239(4) * R(71);
R_dot(72) = H_fraction_Pu239(5) * P_i_Pu239 / 200 - H_lambda_Pu239(5) * R(72);
R_dot(73) = H_fraction_Pu239(6) * P_i_Pu239 / 200 - H_lambda_Pu239(6) * R(73);
R_dot(74) = H_fraction_Pu239(7) * P_i_Pu239 / 200 - H_lambda_Pu239(7) * R(74);
R_dot(75) = H_fraction_Pu239(8) * P_i_Pu239 / 200 - H_lambda_Pu239(8) * R(75);
R_dot(76) = H_fraction_Pu239(9) * P_i_Pu239 / 200 - H_lambda_Pu239(9) * R(76);
R_dot(77) = H_fraction_Pu239(10) * P_i_Pu239 / 200 - H_lambda_Pu239(10) * R(77);
R_dot(78) = H_fraction_Pu239(11) * P_i_Pu239 / 200 - H_lambda_Pu239(11) * R(78);
R_dot(79) = H_fraction_Pu239(12) * P_i_Pu239 / 200 - H_lambda_Pu239(12) * R(79);
R_dot(80) = H_fraction_Pu239(13) * P_i_Pu239 / 200 - H_lambda_Pu239(13) * R(80);
R_dot(81) = H_fraction_Pu239(14) * P_i_Pu239 / 200 - H_lambda_Pu239(14) * R(81);
R_dot(82) = H_fraction_Pu239(15) * P_i_Pu239 / 200 - H_lambda_Pu239(15) * R(82);
R_dot(83) = H_fraction_Pu239(16) * P_i_Pu239 / 200 - H_lambda_Pu239(16) * R(83);
R_dot(84) = H_fraction_Pu239(17) * P_i_Pu239 / 200 - H_lambda_Pu239(17) * R(85);
R_dot(85) = H_fraction_Pu239(18) * P_i_Pu239 / 200 - H_lambda_Pu239(18) * R(85);
R_dot(86) = H_fraction_Pu239(19) * P_i_Pu239 / 200 - H_lambda_Pu239(19) * R(86);
R_dot(87) = H_fraction_Pu239(20) * P_i_Pu239 / 200 - H_lambda_Pu239(20) * R(87);
R_dot(88) = H_fraction_Pu239(21) * P_i_Pu239 / 200 - H_lambda_Pu239(21) * R(88);
R_dot(89) = H_fraction_Pu239(22) * P_i_Pu239 / 200 - H_lambda_Pu239(22) * R(89);
R_dot(90) = H_fraction_Pu239(23) * P_i_Pu239 / 200 - H_lambda_Pu239(23) * R(90);

%% Input reactivity
switch SV(1)
   case 'Pulse'
      R_dot(67) = 0;

   case 'Rod Withdraw'
      if R(67) < GV(3) * Beff_mix
         R_dot(67) = GV(4);

      elseif R(67) >= GV(3) * Beff_mix
         R_dot(67) = 0;
      else
         display (' Error in Reactivity Addition ');
      end

   otherwise
      R_dot(67) = 0;
end

%% Temperature expansion
% Lower Graphite
R_dot(91) = ( 1 / ( GV(23) * cp_graphite_91 ) ) ...
   * ( ( R(91) - R(12) ) / ( GV(22)/k_fuel + GV(29)/k_graphite ) * GV(32) ...
      + ( R(96) - R(91) ) / ( dz_fuel_unit/k_fuel + dz_gas/k_gas ) * surface_gas_in_grap );

% upper graphite
R_dot(93) = ( 1 / ( GV(23) * cp_graphite_93 ) ) ...
   * ( ( R(93) - R(12) ) / ( GV(22)/k_fuel + GV(29)/k_graphite ) * GV(32) ...
      + ( R(98) - R(93) ) / ( dz_fuel_unit/k_fuel + dz_gas/k_gas ) * surface_gas_in_grap );
```

```
% Lower water channel
R_dot(92) = 1/trans_factor_92 * ( h_bar_92 * GV(24) * (R(99) - R(92)) ...
      + m_dot(2) * ( h_0 + gravity * ( GV(45) ) )...
      - m_dot(3) * ( h_1 + gravity * (GV(26) + GV(45)) ) );

% Upper water channel
R_dot(94) = 1/trans_factor_94 * ( h_bar_94 * GV(24) * (R(101) - R(94)) ...
      + m_dot(4) * ( h_2 + gravity * (GV(26) + GV(9) + GV(45)) )...
      - m_dot(5) * ( h_3 + gravity * ( 2 * GV(26) + GV(9) + GV(45)) ) );

% Velocity
if ( R(13) - R(94) ) > 0
   R_dot(95) = sqrt( therm_diff_water_13 * ( R(13) - R(94) ) * ( GV(22) + GV(29))  );
else
   R_dot(95) = 0;
end

%% Gas/ SS expansion
% Lower gas
R_dot(96) = 1/(mass_gas_gr * cp_gas) * ( ( R(97) - R(96) )* k_gas * A_gas / ( GV(22) + GV(29)) ...
      + (  R(91) - R(96) ) / ( dz_fuel_unit/k_graphite + dz_gas/k_gas ) * surface_gas_in_grap ...
      + (  R(99) - R(96) ) / ( dz_clad/k_clad + dz_gas/k_gas ) * surface_gas_out_grap );

% Mid gas
R_dot(97) = 1/(mass_gas_fuel * cp_gas) * ( ( R(96) - R(97) )* k_gas * A_gas / ( GV(22) + GV(29)) ...
      + (  R(12) - R(97) ) / ( dz_fuel_unit/k_fuel + dz_gas/k_gas ) * surface_gas_in_fuel ...
      + (  R(100) - R(97) ) / ( dz_clad/k_clad + dz_gas/k_gas ) * surface_gas_out_fuel ...
      + (  R(98) - R(97) ) * k_gas * A_gas / ( GV(22) + GV(29)) );

% Upper gas
R_dot(98) = 1/(mass_gas_gr * cp_gas) * ( ( R(97) - R(98) ) * k_gas * A_gas / ( GV(22) + GV(29)) ...
      + (  R(93) - R(98) ) / ( dz_fuel_unit/k_graphite + dz_gas/k_gas ) * surface_gas_in_grap ...
      + (  R(101) - R(98) ) / ( dz_clad/k_clad + dz_gas/k_gas ) * surface_gas_out_grap );

% Lower Clad (SS304)
R_dot(99) = 1/(mass_clad_gr * cp_clad) * ( ( R(100) - R(99) )* k_clad * A_clad / ( GV(22) + GV(29)) ...
   + (  R(96) - R(99) ) / ( dz_clad/k_clad + dz_gas/k_gas ) * surface_gas_out_grap  ...
   - h_bar_92 * GV(24) * (R(99) - R(92)) );

% Mid Clad (SS304)
R_dot(100) = 1/(mass_clad_fuel * cp_clad) * ( ( R(99) - R(100) )* k_clad * A_clad / ( GV(22) + GV(29)) ...
   + ( R(101) - R(100) )* k_clad * A_clad / ( GV(22) + GV(29)) ...
   + (  R(97) - R(100) ) / ( dz_clad/k_clad + dz_gas/k_gas ) * GV(7) ...
   - h_bar_13 * GV(7) * (R(100) - R(13)) );

% Upper Clad (SS304)
R_dot(101) = 1/(mass_clad_gr * cp_clad) * ( ( R(100) - R(101) )* k_clad * A_clad / ( GV(22) + GV(29)) ...
   + (  R(98) - R(101) ) / ( dz_clad/k_clad + dz_gas/k_gas ) * surface_gas_out_grap ...
   - h_bar_94 * GV(24) * (R(101) - R(94)) );

%% Momentum balance expansion
% Coolant area between lower graphite and lower grid plate (inlet)
R_dot(102) = 1 / (mass_cooling_water_102 * cp_H2O_102) * ( ...
   + m_dot(1) * ( h_0 ) - m_dot(2) * ( h_102 + gravity * GV(45) ) ...
   + eff_lower_fin * h_bar_102_fin * GV(54) * (R(104) - R(102)) ...
   + h_bar_102_cyl * GV(56) * (R(104) - R(102)) ...
   + h_bar_102_cone * GV(58) * (R(104) - R(102)) );

% Coolant area above upper and below upper grid plate (outlet)
R_dot(103) = 1 / (mass_cooling_water_103 * cp_H2O_103) * ( ...
   + m_dot(5) * ( h_3 + gravity * ( 2 * GV(26) + GV(9)) ) ...
   - m_dot(6) * ( h_103 + gravity * ( 2 * GV(26) + GV(9) + GV(45))) ...
   + eff_upper_fin * h_bar_103_fin * GV(55) * ( R(105) - R(103)) ...
   + h_bar_103_cyl * GV(57) * (R(105) - R(103)) ...
   + h_bar_103_cone * GV(59) * (R(105) - R(103)) );

% SS region of pin below lower graphite sitting on lower grid plate
R_dot(104) = 1 / (GV(50) * cp_clad) * ( (R(91) - R(104)) / ( GV(52)/k_clad + GV(29)/k_graphite) ...
   - eff_lower_fin * h_bar_102_fin * GV(54) * ( R(104) - R(102)) ...
```

```
    - h_bar_102_cyl * GV(56) * (R(104) - R(102)) ...
    - h_bar_102_cone * GV(58) * (R(104) - R(102)) );

% SS region of pin above upper graphite up to upper grid plate
R_dot(105) = 1 / (GV(51) * cp_clad) * ( (R(93) - R(105)) / ( GV(53)/k_clad + GV(29)/k_graphite) ...
    - eff_upper_fin * h_bar_103_fin * GV(55) * ( R(105) - R(103)) ...
    - h_bar_103_cyl * GV(57) * (R(105) - R(103)) ...
    - h_bar_103_cone * GV(59) * (R(105) - R(103)) );

%% Output
R_dot = R_dot';

if ~isreal(R)
    low_temp
    c_temp
    f_temp
    cp_H2O_94
    cp_H2O_92
    cp_H2O_13
    r92 = R(92)
    r94 = R(94)
    r13 = R(13)
    r99 = R(99)
    r100 = R(100)
    r101 = R(101)
    r102 = R(102)
    r103 = R(103)
    r104 = R(104)
    r105 = R(105)
    m_dot
    h_bar_13
    h_bar_92
    h_bar_94
    return;
end
```

Coolant Flow Development Program

```
function [ m_dot ] = Coolant_properties_1_0( R, P )
%% Coolant property momentum balance
% Author: Greg Kline
% Date: 8 Jul 2016
% Revision 1.0
%
% In order to better quantify the effects of thermal hydraulics, a
% conservation of momentum over conv of mass
% flow is considered driven by the fuel water region to upper graphite
% this gives a state dependent velocity across that border,
% using this, the additional velocities are found using
% pAw^2(in) - pAw^2(out) = F_darcy friction + F_gravity + F_misc
% while prudency may demand using an average or inlet velocity in the
% Darcy factor, the calculation intensity is minimized using the known

%% ODE Variables
% R(1) - neutron density
% R(2) - group 1 concentration
% R(3) - group 2 concentration
% R(4) - group 3 concentration
% R(5) - group 4 concentration
% R(6) - group 5 concentration
% R(7) - group 6 concentration
% R(8) - AmBe source concentration
% R(9) - spontaneous U238 concentration
% R(10) - spontaneous U235 concentration
% R(11) - reactivity at t
% R(12) - temperature of the fuel at t
% R(13) - temperature of the moderator at t
% R(14) - B_eff_mix at t
% R(15) - B_mix_1 at t
```

```
% R(16) - B_mix_2 at t
% R(17) - B_mix_3 at t
% R(18) - B_mix_4 at t
% R(19) - B_mix_5 at t
% R(20) - B_mix_6 at t
% R(21) - H_U2325_group_1 decay heat matrix
% R(22) - H_U2325_group_2
% R(23) - H_U2325_group_3
% R(24) - H_U2325_group_4
% R(25) - H_U2325_group_5
% R(26) - H_U2325_group_6
% R(27) - H_U2325_group_7
% R(28) - H_U2325_group_8
% R(29) - H_U2325_group_9
% R(30) - H_U2325_group_10
% R(31) - H_U2325_group_11
% R(32) - H_U2325_group_12
% R(33) - H_U2325_group_13
% R(34) - H_U2325_group_14
% R(35) - H_U2325_group_15
% R(36) - H_U2325_group_16
% R(37) - H_U2325_group_17
% R(38) - H_U2325_group_18
% R(39) - H_U2325_group_19
% R(40) - H_U2325_group_20
% R(41) - H_U2325_group_21
% R(42) - H_U2325_group_22
% R(43) - H_U2325_group_23
% R(44) - H_U2328_group_1
% R(45) - H_U2328_group_2
% R(46) - H_U2328_group_3
% R(47) - H_U2328_group_4
% R(48) - H_U2328_group_5
% R(49) - H_U2328_group_6
% R(50) - H_U2328_group_7
% R(51) - H_U2328_group_8
% R(52) - H_U2328_group_9
% R(53) - H_U2328_group_10
% R(54) - H_U2328_group_11
% R(55) - H_U2328_group_12
% R(56) - H_U2328_group_13
% R(57) - H_U2328_group_14
% R(58) - H_U2328_group_15
% R(59) - H_U2328_group_16
% R(60) - H_U2328_group_17
% R(61) - H_U2328_group_18
% R(62) - H_U2328_group_19
% R(63) - H_U2328_group_20
% R(64) - H_U2328_group_21
% R(65) - H_U2328_group_22
% R(66) - H_U2328_group_23
% R(67) - control rod reactivity at t
% R(68-90) - Pu239
% R(91) - Lower graphite temp
% R(92) - Lower moderator temp
% R(93) - Upper graphite temp
% R(94) - Upper mod temp
% R(95) - velocity
% R(96) - Lower Gas region
% R(97) - Mid Gas region
% R(98) - Upper Gas region
% R(99) - Lower SS Region
% R(100) - Mid SS region
% R(101) - Upper SS region
% R(102) - Lower pin grid plate coolant region
% R(103) - Upper pin grid plate coolant region
% R(104) - Pin lower fins and region
% R(105) - Pin upper fins and region

%% P = [
```

```
%   1  density_water_13
%   2  density_water_92
%   3  density_water_94
%   4  area_cooling_pin
%   5  flow_outlet_core
%   6  flow_inlet_core
%   7  therm_diff_water_13
%   8  therm_diff_water_92
%   9  therm_diff_water_94
%   10  gravity
%   11  pin_height
%   12  graphite_height
%   13  radius_pin
%   14  inner_hex
%   15  surface_area_graphite
%   16  surface_area_fuel
%   17  density_water_102
%   18  density_water_103
%   19  low_fin_height
%   20  upper_fin_height
%   21  A_flow_outlet_grid
%   22  P_flow_outlet_grid
%   23  lower_cone_height
%   24  upper_cone_height
%   25  volume_cooling_lower_conical
%   26  volume_cooling_upper_conical
%   27  density_water_0
%   28  A_flow_inlet_grid
%   29  P_flow_inlet_grid
%   30  volume_cooling_lower_conical
%   31  volume_cooling_upper_conical ...
%   32  lower_pin_below_gr
%   33  upper_pin_above_gr
%   34  lower_fin_height ...
%   35  lower_cyl_height
%   36  upper_cyl_height
%   37  mass 104 ss region GV(50)
%   38  mass 105 ss region
%   39  dz_lower_ss_cond
%   40  dz_upper_ss_cond
%      ];

%% Designators
% 0 - inlet under or at lower grid plate
% 1 - boundary between lower finned portion and lower edge of graphite
% 2 - boundary between lower graphite and fuel
% 3 - boundary between fuel and upper graphite
% 4 - boundary between upper graphite and upper fuel area
% 5 - outlet or just above upper grid plate

% Darcy Roughness for SS (m) [Eng. toolbox]
e = .01524e-3;

% Kinetic Viscosity (m^2/s)
v = .279e-6;

% gravity (m/s^2)
gravity = 9.8066;

% Find hydraulic diameter (m)
% Wetted perimeter (m)
Pewit = 12 * P(14) / sqrt(3) + 2 * pi * P(13);

% flow area (m^2)
A_cool_1 = P(4);
A_cool_2 = P(4);
A_cool_3 = P(4);
A_cool_4 = P(4);

% Hydraulic diameter (m)
```

```
D_H_0 = 4 * P(28) / P(29);
D_H_1 = 4 * A_cool_1 / Pewit;
D_H_2 = 4 * A_cool_2 / Pewit;
D_H_3 = 4 * A_cool_3 / Pewit;
D_H_4 = 4 * A_cool_4 / Pewit;
D_H_5 = 4 * P(21) / P(22);

% Surface areas of cones
A_cone_lower = pi * P(13) * sqrt( P(23)^2 + P(13)^2 );
A_cone_upper = pi * P(13) * sqrt( P(24)^2 + P(13)^2 );
A_s_102 = 2 * pi * P(13) * P(35);
A_s_103 = 2 * pi * P(13) * P(36);

% w3 is tracked using Boussinesq assumption and the presumption the area
% with the heat flux from fuel drives the overall velocity
% using v3 and a momentum balance, v2 can be found

%%%%% Code to use inlet.  fails at 0 m/s %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%     function [ y ] = w3_fun ( x )
%         y = P(2) * P(4) * x^2 ...
%             - P(1) * P(4) * R(95)^2 ...
%             - P(2) * gravity * P(4) * P(11) ...
%             - .5 * ( P(11) / D_H_2 ) * P(2) * x^2 * P(16) ...
%             / ( log10( (e/(3.7*D_H_2)) + 5.74/(x*P(11)/v) ) )^2;
%
%     end
%
%     w2 = fzero(@w3_fun,.99*R(95))
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%

% Find the Darcy force around fuel (N)
R(95);

%% Fuel section
% fuel area Darcy
F_darcy_13 = .5 * ( P(11) / D_H_2 ) * P(2) * R(95)^2 * P(16) ...
             * ( log10( (e/(3.7*D_H_2)) + 5.74/((R(95)*P(11)/v)^.9) ) )^-2;

% Find the Gravity force around fuel (N)
F_grav_13 = P(1) * gravity * A_cool_2 * P(11);

% find the velocity at 2 (m/s)
if P(1) * A_cool_3 * R(95)^2 - F_darcy_13 - F_grav_13 > 0
    w2 = sqrt( (- F_darcy_13 - F_grav_13 + P(1) * A_cool_3 * R(95)^2) ...
                / ( P(2) * A_cool_2 ) );
else
    w2 = 0;
end

%% Upper graphite
% Find the Darcy friction of upper graphite (N)
F_darcy_94 = .5 * ( P(12) / D_H_3 ) * P(1) * R(95)^2 * P(15) ...
             * ( log10( (e/(3.7*D_H_3)) + 5.74/((R(95)*P(12)/v)^.9) ) )^-2;

% Find gravity force of upper graphite (N)
F_grav_94 = P(2) * gravity * A_cool_3 * P(12);

% find the velocity at 4 (m/s)
w4 = sqrt( (P(1) * A_cool_3 * R(95)^2 + F_darcy_94 + F_grav_94) ...
             / ( P(2) * A_cool_4 ) );

%% Lower graphite
% Find the Darcy friction of lower graphite region (N)
F_darcy_92 = .5 * ( P(12) / D_H_2 ) * P(17) * w2^2 * P(15) ...
             * ( log10( (e/(3.7*D_H_2)) + 5.74/((w2*P(12)/v)^.9) ) )^-2;

% Gravity force of 92
F_grav_92 = P(3) * gravity * A_cool_1 * P(12);
```

```
% find the velocity at 1
if P(2) * A_cool_3 * w2^2 - F_darcy_92 > 0
    w1 = sqrt( (-F_darcy_92 - F_grav_92 + P(2) * A_cool_3 * w2^2) ...
                / ( P(17) * A_cool_1 ) );
else
    w1 = 0;
end

%% Lower conical region
% Drag force on the conical region (drag coefficient from rocketry)
F_drag_lower_cone = .5 * P(17) * w1^2 * .2 * A_cone_lower;

% Darcy friction for the small section
F_darcy_102 = .5 * ( P(32) / D_H_1 ) * P(17) * w1^2 * A_s_102 ...
            * ( log10( (e/(3.7*D_H_1)) + 5.74/((w1*P(17)/v)^.9) ) )^-2;

% Gravity force
F_grav_102 = P(17) * gravity * P(25);

% find the velocity at 1
if P(17) * A_cool_3 * w2^2 - F_drag_lower_cone - F_darcy_102 > 0
    w0 = sqrt( (-F_drag_lower_cone - F_darcy_102 - F_grav_102 + P(17) * A_cool_3 * w2^2) ...
                / ( P(27) * P(28) ) );
else
    w0 = 0;
end

%% Upper conical region
% Drag force on the conical region (drag coefficient from rocketry)
F_drag_upper_cone = .5 * P(18) * w4^2 * .2 * A_cone_upper;

% Darcy friction for the small section
F_darcy_103 = .5 * ( P(33) / D_H_4 ) * P(3) * w4^2 * A_s_103 ...
            * ( log10( (e/(3.7*D_H_4)) + 5.74/((w4*P(3)/v)^.9) ) )^-2;

% Gravity force
F_grav_103 = P(18) * gravity * P(26);

% find the velocity at 1
w5 = sqrt( (P(3) * A_cool_4 * w4^2 + F_drag_upper_cone + F_darcy_103 + F_grav_103) ...
            / ( P(18) * P(21) ) );

%% Build mass flow rate vectors
if R(95) > 0
    m_dot(1) = P(27) * P(28) * w0;
    m_dot(2) = P(17) * A_cool_1 * w1;
    m_dot(3) = P(2) * A_cool_2 * w2;
    m_dot(4) = P(1) * A_cool_3 * R(95);
    m_dot(5) = P(3) * A_cool_4 * w4;
    m_dot(6) = P(18) * P(21) * w5;

else
    m_dot = [ 0 0 0 0 0 0 ];
end

end
```