TRANSIENT SIMULATIONS OF THE SLOWPOKE-2 REACTOR USING THE G4-STORK CODE

TRANSIENT SIMULATIONS OF THE SLOWPOKE-2 REACTOR USING THE G4-STORK CODE

By ANDREW TAN, B. SC.

A Thesis Submitted to the School of Graduate Studies in Partial Fulfilment of the Requirements for the Masters Degree of Applied Science

McMaster University MASTERS OF APPLIED SCIENCE (2015) Hamilton, Ontario (Engineering Physics)

TITLE: Transient Simulations of the SLOWPOKE-2 Reactor Using the G4-STORK code

AUTHOR: Andrew Tan, B.Sc. (University of Guelph)

SUPERVISOR: Professor Adriaan Buijs

NUMBER OF PAGES: ix, 90

Abstract

The goal of this thesis is to study the transient behaviour of the SLOWPOKE-2 reactor using Monte-Carlo simulations with the G4-STORK code. G4-STORK is a 3-dimensional Monte-Carlo code derived from the GEANT4 physics simulation toolkit. Methods were developed for the proper treatment of delayed neutrons and a lumped capacitance model was used to track the time-dependent fuel properties (temperature, density) based on the fission power. By validating the methods in G4-STORK with experimental measurements we hope to extend our understanding of reactor transients as well as further develop our methods to model the transients of the next generation reactor designs. A SLOWPOKE-2 reactor such as the one at RMC was chosen for simulation due to its compact size, and well-known transient response of control rod removal and measured temperature feedback. Static simulations in G4-STORK find a neutron flux of order 10^{12} cm^{-2} s^{-1} which agrees with experiment and a control rod worth of (4.9 ± 2.0) mk compared to the experimentally measured worth of 5.45 mk. Transient simulations from rod pluck-out find similar trends to the experimental findings as our results suggest a negative temperature feedback due to the doppler broadening of the U-238 absorption spectrum which contributes to the overall safety mechanism seen in the SLOWPOKE reactor. It is determined that the methods in G4-STORK provide a reasonable ability to simulate reactor transients and it is recommended that a full-core thermal-hydraulics model be coupled to G4-STORK to achieve a higher level of accuracy.

Contents

List of Figures

Declaration of Academic Achievement

Chapter 1

Introduction

The ability to accurately predict and model the power production of a nuclear reactor during a transient is of great importance in nuclear safety. Control rod movements and loss of coolant events occur during normal operation as well as accident scenarios and can quickly lead to unpredictable behaviour. Improving our current understanding of these scenarios in terms of how and why they evolve can help increase the safety of our current reactors. It is during the transient evolution of a reactor where predicting the changing rate of power, and the fuel, coolant and moderator properties are of interest. The evolution however, is quite complex and occurs in a very short amount of time. Thus it is necessary to design and test new methods to simulate these transients.

Computer simulations have played a vital role in the design, operation and safety in nuclear reactors since the beginning of the nuclear age. They provide a cost effective and fair alternative to experiment. With the progression of computational power researchers have the ability to model and solve problems of greater complexity and scale. Nuclear transients which involve many coupled systems rapidly changing can be simulated with today's technology.

1.1 Computational Reactor Physics

Computational physics is the study and application of numerical analysis to solve problems in physics. It is used in nearly every branch of physics but has major application in mechanics, fluid dynamics and electrodynamics. Computational physics is the bridge between theoretical and experimental physics. Its concern is not only

the development of more accurate models but also the testing and verification of its results. With strides being made in the processing power of CPUs, high performance parallel computing and more recently the use of graphical processing units (GPUs) to deal with special vectorized problems, the use of computation is growing in nearly every field.

Computational reactor physics is the subset which concerns itself with the physics of reactors. A wide variety of codes have been developed to solve for the key quantities such as reactivity and flux and to do so they must solve the key equations pertaining to neutron transport in the reactor. In a nuclear reactor the changes in the operation and behaviour of the reactor stem from the changes in the neutron distribution. The neutron response comes from changes in their environment or more specifically the interacting geometry and materials. These two systems feed back into each other where the neutrons change the environment and the environment changes the neutron distribution. These systems also operate on various timescales, where the neutron lifetimes in a reactor are short compared to the timescales seen in the movement of control rods, thermal-hydraulics of the reactor and the isotopic re-composition of the fuel. This is a complex problem that requires the ingenuity of scientists and developers with the processing speed of computers to solve [1, p. 1].

Nuclear simulation codes can be divided into two groups: stochastic and deterministic. Both methods produce results that are used in industry but there are trade-offs between the two that may make one more suitable to use for a particular application. In a deterministic code the solvers typically use approximations or simplifications to solve for an exact solution to the problem. The stochastic codes use a full description of the problem but produce an approximate solution to the problem. Additionally, the stochastic codes are more computationally expensive because they require the generation of pseudo-random numbers.

Traditionally, the focus of nuclear reactor codes has been on static cases, in full-power operation where the reactor power is stable. In these cases it can be assumed without a significant loss of accuracy that the system is time-independent. These static calculations are useful for the process of designing a reactor as well as developing refueling schemes. Some methods to establish limited time-dependency include *quasistatic* methods where the time-independent calculation is combined with a time-dependent model. Such a case would be combining a time-dependent fuel burn-up code with a static calculation at each burn-up in an iterative fashion. The

calculation of neutron flux and power serves as a input into the calculation of fuel burn-up at a future time. The new burn-up then serves as input into the new flux and power calculation and the process is repeated. Such a method may be sufficient to characterize the reactor over normal controlled conditions but lacks the detail and sophistication to capture the behaviour of accident scenarios. Typically deterministic codes are employed when solving static cases of large reactors.

To model the quick changes seen in accident scenarios the use of Monte Carlo methods are a preferred choice. Accidents must be modeled as a dynamic system as they are transient. The basis of Monte Carlo is simple, reproduce data through iterative random sampling. These stochastic methods can be used to solve complex systems without the need of approximations or simplifications in the model such as spatial and energy discretization. The drawback of the method however is the computational effort required. Each neutron in the simulation must be followed with each step sampled for. It should also be noted that the process of randomly sampling is subject to statistical errors which can only be eliminated with an infinite number of particles. Thus the at the cost of computational time Monte Carlo serves as an accurate and sensible solution to modeling dynamic systems in reactor physics.

1.2 G4-STORK Reactor Kinetics Code

The G4-STORK (STOchastic Reactor Kinetics) code developed at McMaster University is a time-dependent Monte Carlo nuclear simulation code derived from CERN's GEANT4 (GEometry ANd Tracking) toolkit. Written in C++, the basis of simulation in G4-STORK is a user defined world where it is up to the user to define the geometry and materials of the world, particles and the physical interactions they undergo. Currently G4-STORK has been validated against MCNP, DRAGON and TART 2005 for criticality calculations [2].

1.2.1 History and Development

G4-STORK was first developed by M.A.Sc. student Liam Russell at McMaster University. The intent was to create a platform to simulate time-dependent neutron populations in a changing environment using the GEANT4 toolkit as the basic framework. Using a finite geometry of a sphere U-235, G4-STORK was verified against

TART and MCNP in terms of reactivity. It was also verified against DRAGON using a CANDU6 lattice cell.

In order to increase the computational abilities of the code and utilize multi-core processing the code was also adapted to run in parallel using the TOP-C (Task Oriented Parallel C/C++) package. TOP-C is a free, open-source library designed to allow users to easily parallelize their C/C++ applications. TOP-C employs the master-slave process hierarchy. Tasks are managed by the master and delegated to the slaves, after a task is completed the output is checked and the master decides on the next action [3, 4].

Further development was made in terms of the reactor geometries available to simulate. Currently the G4-STORK code has simulation worlds built for the CANDU6 lattice, ZED-2 reactor, McMaster Nuclear Reactor, SLOWPOKE reactor and the Super-Critical Water Reactor (SCWR) concept of AECL (now CNL). A visualization of the SLOWPOKE reactor is shown in Figure 1.1.

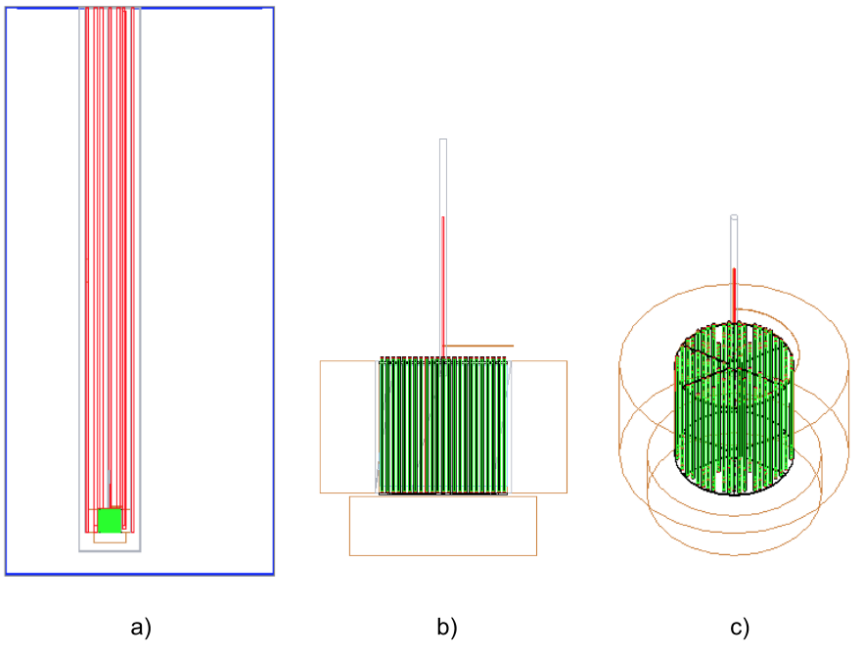

Figure 1.1: Visualization of the SLOWPOKE reactor in GEANT4. **a)** Reactor core *(green)* with irradiation tubes *(red)* submerged in the pool. **b)** Side view of the core, with fuel rods *(green)* and beryllium reflector *(yellow)*. **c)** Angled view with control rod *(red)*

1.2.2 Features

One of the major features that makes G4-STORK unique from other codes is that it is user-extensible. It is up to the user to define the physics, particles and materials and geometry that make up the simulation world giving the user complete control of the simulation. The user is also capable of creating their own processes and methods to add to the simulation which can provide a means of testing new methods.

G4-STORK features dynamic geometry changes during simulation. This means that the user can simulate reactor operations such as control rod movements or even an accident scenario. This is a powerful feature in simulating reactor transients.

Doppler broadening in G4-STORK is done on the fly. G4-STORK uses the G4NDL libraries which are held at 0K. By doppler broadening up to the specified temperature, G4-STORK has the capability to model temperature feedback effects.

To allow reasonable computational cost, G4-STORK is equipped with a renormalization feature. Since the neutron population in a reactor can exponentially grow or shrink in time the population may be renormalized to a set population. The renormalization is unbiased as it randomly selects neutrons from the resulting population to continue in the simulation [4].

1.2.3 Current and Future Progress

Several reactors have been added to G4-STORK's arsenal since its creation. The ZED-2 reactor was developed and tested by M.A.Sc. student Salma Mahzooni. Testing involved the subcritical experiments done at AECL [5]. The McMaster Nuclear Reactor was developed by undergraduate student Alexander Otto where reactivity measurements were made against control rod movement [6].

To allow the model of transient behaviour in G4-STORK a delayed neutron model which simulates time-dependent precursors and a thermal model to simulate the time-dependency of fuel properties during power changes have been developed as part of this thesis and are further described in later sections.

There has been some progress made in terms of improving the speed of G4-STORK. The doppler broadening process employed in G4-STORK typically involves an iterative calculation the size of which is proportional to the difference of the temperature of the library used and the temperature of the material of interest. In our case the library starts at 0K. A speed-up was contributed by M.A.Sc candidate Wesley Ford who

has produced pre-doppler broadened libraries of the G4NDL cross-sections at various temperatures. These libraries result in the code spending less time broadening the libraries as it may start the calculation from a closer temperature [2].

Currently development is underway to allow G4-STORK to use Graphical Processing Units (GPUs) for computation. A large improvement in computational expense can be found by using GPUs to perform traditional calculations done on CPUs. The advantage of GPUs is their ability to handle large vectorized computations. By altering our method of parallelism by vectorizing a subset of computations the workload can be processed with the GPU and a reduction in computational expense can be achieved. With a significant speed up G4-STORK will be able to simulate more neutrons and for a longer period of real time.

To further valid the ability of G4-STORK, comparisons between the code and the well established MCNP code are being made using the Super-Critical Water Reactor. This project is led by Wesley Ford. To further investigate the differences between the codes he has developed a data library conversion package between the G4NDL and MNCP library format.

1.2.4 The GEANT4 Toolkit

The GEometry ANd Tracking 4 (GEANT4) code is an open source simulation platform designed for the purpose of modelling particles and their interactions with matter. GEANT4 is collaborated from scientists and software engineers around the world. The current version was written in C++, derived from its predecessor GEANT3. In 1993, two separate studies were done to determine how modern computing techniques could be applied to computational particle physics. It was concluded that using object-oriented programming should be the next evolution in GEANT3 and thus GEANT4 was developed. Spearheaded by CERN, the first release of GEANT4 was in 1998 and to further maintain and develop the code the GEANT4 consortium was formed in 1999. The GEANT4 consortium is formed of many groups including the European Organization for Nuclear Research (CERN), the SLAC National Accelerator Laboratory, Massachusetts Institute of Technology (MIT), TRIUMF, University of Alberta and KEK.

In contrast to other codes, GEANT4 is designed as a simulation toolkit, meaning the user can build a specific simulation using the classes provided. GEANT4 has

seen applications in many fields other than particle physics: medical physics, space engineering, and accelerator design [7].

1.3 The SLOWPOKE-2 Reactor

The SLOWPOKE-2 (Safe LOW POwer Kritical Experiment) reactor is a low-power, in-pool type research reactor and features a light-water moderated low-enriched uranium core which runs at a nominal power of 20 kW. It features a single control rod placed at its center with a 216 pin configuration using Low Enriched Uranium (LEU) fuel. Designed by Atomic Energy of Canada Limited (AECL, now Canadian Nuclear Laboratories, CNL) in the late 1960s the SLOWPOKE was intended to provide a neutron source for teaching institutions and experiments. The low operating power of the SLOWPOKE means that it is a low-risk facility. SLOWPOKEs have been in operation for over 30 years and have a proven track record of safety.

A self-regulating feedback mechanism that is present in the SLOWPOKE reactor provides it with such a high safety level. With an undermoderated lattice, any decrease in coolant density will provide a reduction in reactivity. Additionally in the LEU fuelled core, an increased fuel temperature provides a negative feedback because of the doppler broadening of the U-238 absorption spectrum. Due to these mechanisms the SLOWPOKE has been licensed to run unsupervised and without the need of an emergency shutdown system.

The core dimensions measure 22.1 cm in diameter and 22.7 cm in depth. The small size and high neutron flux on the order of 10^{12} $cm^{-2}s^{-1}$ means the SLOWPOKE is tightly coupled neutronically and is a sensible candidate for computer simulation. [8, 9]

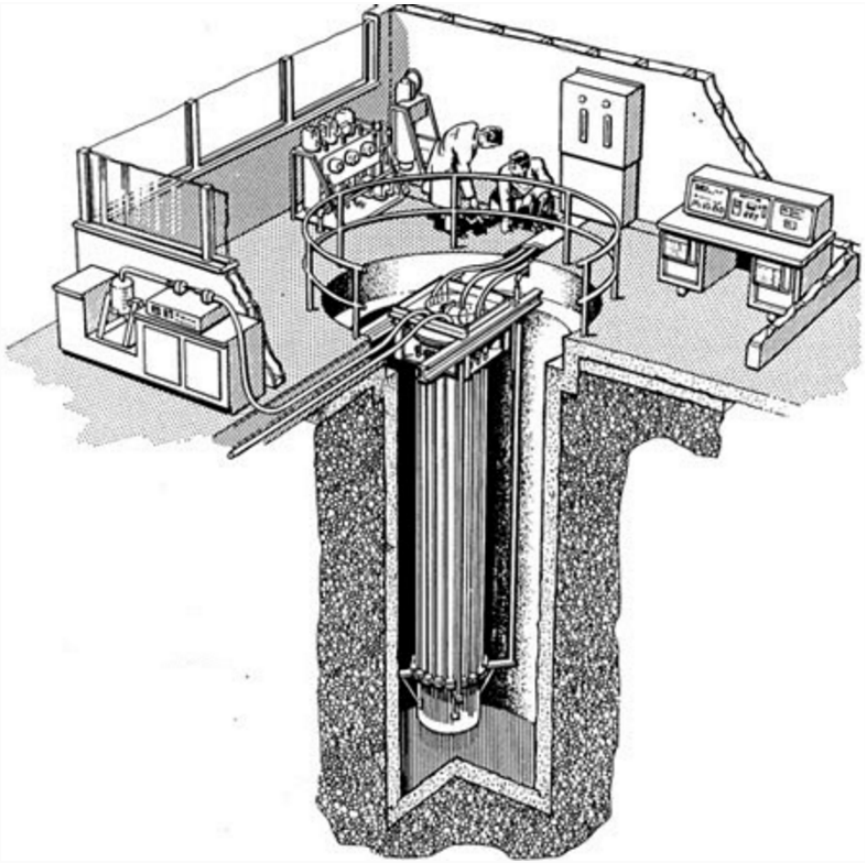

Figure 1.2: The SLOWPOKE reactor [8].

1.3.1 SLOWPOKE-2 Reactor Model

The following list provides a description of the components used in the SLOWPOKE-2 reactor model.

Reflector: The beryllium reflector surrounds the core and is comprised of a surrounding annulus with an additional plate placed underneath.

Fuel Pins: LEU fuel was used as the fuel. They were placed inside the a zirconium sheathe surrounded with air.

Irradiation Sites: Five identical irradiation sites were used, they were modeled using straight aluminum tubes.

Coolant/Moderator: Light water is used as the coolant and flows via natural convection in the container through the core.

Fuel Sheathe: The fuel sheathe is made of zirconium and held in place by the grid.

Reactor Container and Pool: The reactor container was modelled as an aluminum cylinder with enclosing disk at the bottom. It was placed off-center in the reactor pool filled with light water.

D2O container: Unique to the RMC SLOWPOKE, an aluminum container filled with D2O was placed alongside the reactor core. It is shaped in the form of a partial annulus spanning only $\pi/3$ radians.

1.4 Objectives

The purpose of this thesis is to extend the capabilities of the GEANT4 toolkit to applications for reactor physics modelling. In the particular, the case of transient modes of behaviour in the reactor. This includes the time-dependency of the reactor fuel properties, calculation of neutron flux and proper treatment of delayed neutrons all of which are linked to create a more detailed simulation. The reactor of interest is the SLOWPOKE-2 reactor from the Royal Military College in Kingston, Ontario. This reactor was chosen based on its small and simplistic design. With only one control rod placed in its center and experimental measurements of its control rod worth and

transient response to rod pluck out, this thesis attempts to reproduce these results in order to test and verify the methods used.

The motivation of such a task is for the continued development of next generation simulation tools. These tools will be the essential instruments when designing the next generation energy systems. Fusion reactors, antimatter and nuclear electric propulsion systems, small modular reactors are some of the complex systems that need the aid of simulation tools for design and testing. The GEANT4 platform is a prime candidate to develop into the next level of simulation tools. GEANT4 starts at the very fundamental level of particles and their interactions with matter. With support from CERN, SLAC, Fermilab and many other physics research groups the models used in GEANT4 represent the most current and established physics.

The development of the G4-STORK project is an extension of the GEANT4 into the realm of nuclear reactor simulation. With further progression of the code, refinement of methods, testing and verification against other codes such as MCNP as well as experiment, it is ultimately the goal of the G4-STORK project to develop a fully time-dependent reactor simulation code with complete feedback between the constituent particles and materials in the reactor.

1.5 Contribution

This thesis outlines the contribution of the development and testing of transient methods in G4-STORK, specifically in the SLOWPOKE-2 reactor. Several models were developed in order to create a more realistic simulation. Firstly, the proper treatment of delayed neutrons. To simulate a time-dependent model of delayed neutrons a method was implemented which models the creation and decay of precursors. Hence the history and current status of the reactor will dictate the behaviour of the delayed neutrons and precursors. The previous methods in G4-STORK relied on the approximation that the reactor was near-critical or required extremely long simulations to produce a delayed neutron source. With our new method we no longer require these as we can estimate a precursor population based on fission rate and allow the precursors to naturally decay.

The SLOWPOKE-2 reactor is known for its inherent safety and its self-regulation when the power increases due to rod removal which is shown in Figure 1.3. The negative temperature feedback in the SLOWPOKE-2 contributes to this safety mechanism and

thus it is of interest to simulate. A model was created to update the temperature, density and volume of the fuel rods according to the time-evolving reactor power. This model uses a simple lumped capacitance method that dissipates heat into the surrounding coolant. Our aim of this model was to illustrate the ability to simulate the temperature feedback effect rather than absolute certainty in the thermal properties of our fuel. Since we are neglecting to track property changes of the other materials in the reactor and cannot simulate the corresponding timescales, we do not expect to have a complete agreement with the experimental results. Figure 1.4 shows the measured negative feedback that occurs when the average core temperature of the SLOWPOKE increases.

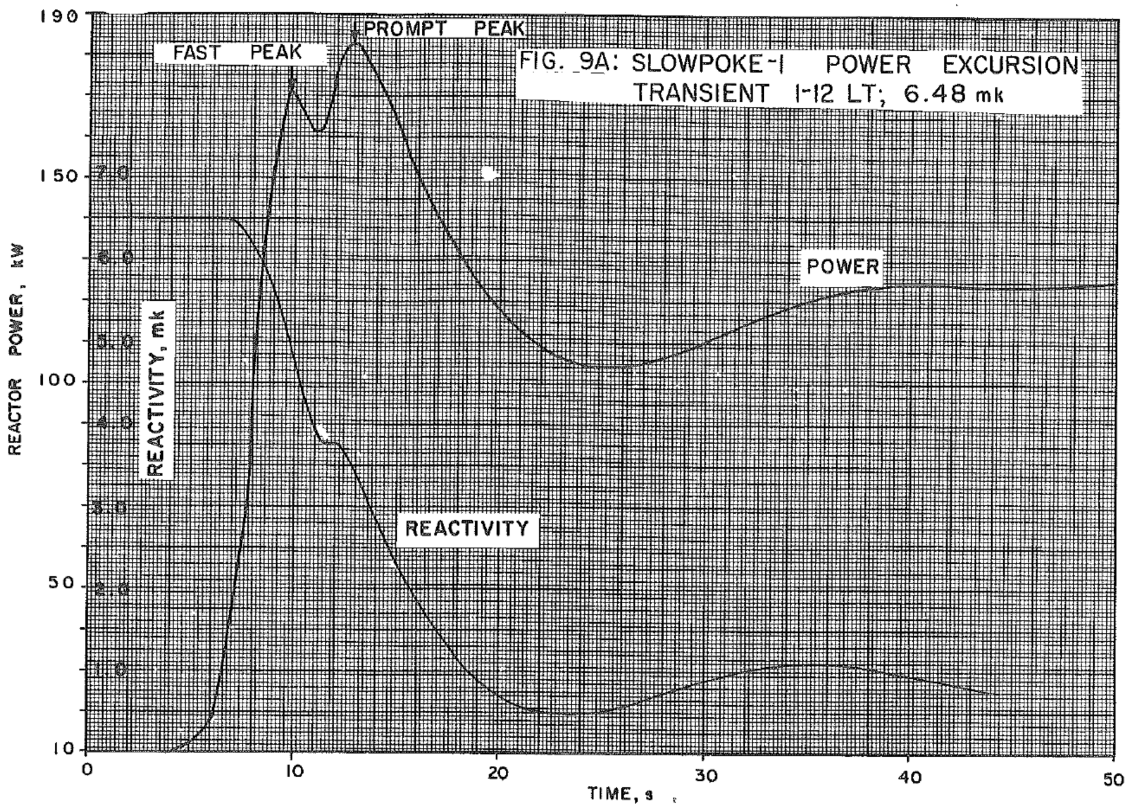

Figure 1.3: Experimental measurements of the power and reactivity of the SLOWPOKE transient [10].

With these additions to the G4-STORK code several tests were performed on the SLOWPOKE-2 reactor. In particular we focused on the simulated value of the control rod worth. The reactivity response of increasing the fuel temperature and lastly

we show the simulations of a rod pull out transient. These tests were conducted to investigate the current capabilities of the G4-STORK code. Through this investigation we hope to make further suggestions and recommendations into the future works that should be added to the code to increase its accuracy in simulating reactor transients.

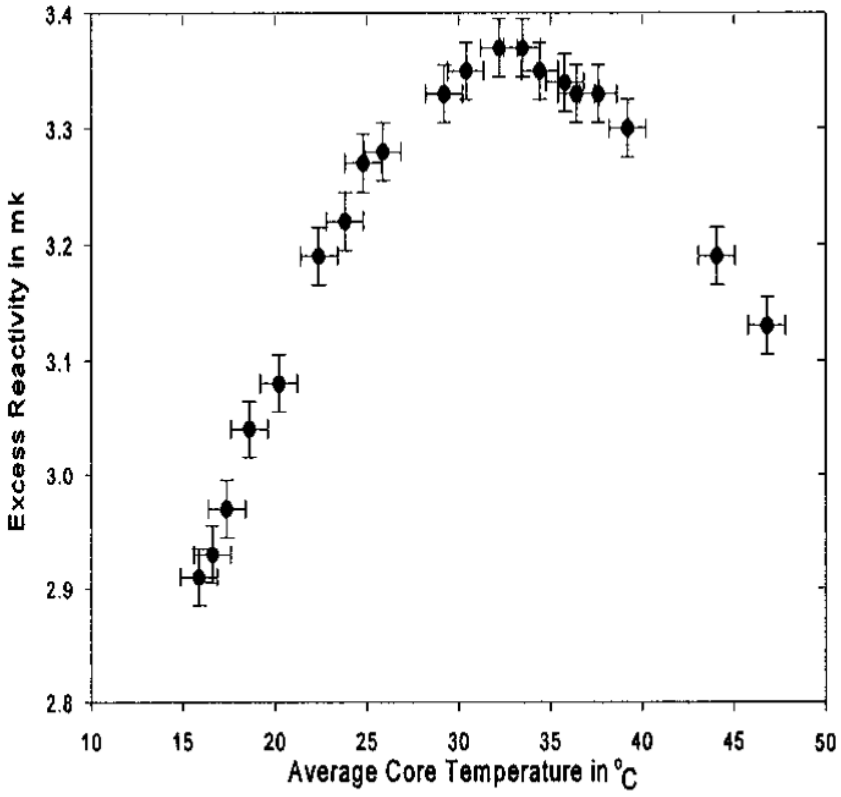

Figure 1.4: Excess reactivity measured as a function of average core temperature [11].

1.6 Related Research

The development of other time-dependent Monte Carlo codes has been done with similar objectives in modelling reactor transients. Utilizing the general purpose transport code Tripoli, Sjenitzer and Hoogenboom have developed a Dynamic Monte Carlo method and provide an alternative approach to simulating delayed neutrons. This is a promising method however it relies on the use of precursor and delayed neutron weighting. Essentially precursors are forced to decay into the simulation and the resulting delayed neutrons are weighted accordingly to account for this. In contrast,

G4-STORK offers a simpler approach, but one that closer emulates reality. The same group has also coupled a thermal-hydraulics code to their model, with validation against a diffusion code and pin power solver they show it is possible to perform transient analysis with Monte Carlo. However more research into the cause of discrepancies must still be done [12].

Another Monte Carlo approach to modelling transients was developed at École Polytechnique de Montréal using the OpenMC tools. Similar methods are employed to that of G4-STORK, dynamic geometry changes and delayed neutrons models are implemented. Validation of the CANDU6 lattice against the Serpent code has been done for a LOCA scenerio. Their models however lack any heat transfer effects or doppler broadening methods which we provide in our code. It is mentioned that this is a source of error in their simulations and is to be addressed in their future work [13].

The Monte-Carlo N-Particle (MCNP5) code is a three dimensional Monte Carlo code developed at Los Alamos National Labs (LANL). MCNP5 is a particle transport code that can be used for neutrons, photons and electrons and has applications in the nuclear field [14]. Both MCNP5 and G4-STORK are used for criticality calculations and general reactor physics simulations. There are differences however in that MCNP5 uses the ENDF (Evaluated Nuclear Data File) data libraries for their neutron transport simulations compared to the G4ENDL (Geant4 Evaluated Nuclear Data Library) data used in GEANT4/G4-STORK. In terms of reactor simulation MCNP5 will track its neutrons through the reactor until that neutron is killed, either by fission or capture. A unit of simulation called a 'cycle' will follow a starting neutron distribution until they are all killed. The resulting neutrons that are created during the current cycle are saved and used as the starting neutron distribution for the next cycle. This generational method of following neutrons however is inherently time-independent as each neutron has a unique lifetime. This method would be insufficient for simulating fast reactor transients as well as sub-critical or super-critical systems. For near-critical static cases this method has been proven to provide reliable and accurate results but it should be used with caution when regarding reactor kinetics.

1.6.1 SLOWPOKE Reactor Simulations

Several others have performed simulations of the SLOWPOKE-2. A model called SLOWKIN was developed at École Polytechnique de Montréal. The model was based on their own SLOWPOKE-2 reactor and featured a lumped parameter model to

predict the temperatures of the reactor materials. Their in-house codes DRAGON and DONJON were used for the calculation of the neutronics, however the calculations were based on the diffusion approximation and only utilized a discretization of six energy groups [9].

The Monte Carlo method has also been applied to the SLOWPOKE reactor. The SLOWPOKE-2 at the Royal Military College has been modelled using the Monte Carlo N-Particle (MCNP4A) code. In this investigation a simplified lattice cell was used in the calculations as opposed to a full-core. To simulate the changes in temperature the densities and volume of the fuel was updated as well as the density of the coolant. This method fails to address the effect that temperature broadened cross sections would have on the system. In all cases only static calculations were used thus a true transient model was not established [11].

Chapter 2

Background and Theory

2.1 Neutron Transport

Neutron transport is the study of the motions of neutrons within, and interactions with materials. It is one of the essential ingredients in understanding the nuclear reactor physics and one that every reactor physicist is well-versed in. Reactor scientist and engineers use neutron transport to model how neutrons will behave in a certain environment.

2.1.1 Neutron Interactions

The interactions of neutrons with materials can be divided into absorption and scattering. The scattering process occurs when neutrons interact with nuclei without being absorbed by the nuclei. Instead the neutron may conserve or lose some of its kinetic energy and change momentum.

When describing the interactions of a projectile with a target the following notation is used:

$$A + b \rightarrow c + D, \tag{2.1}$$

where A and b refer to the reactant nucleus and particle and D and c are the product nucleus and particle. We can condense this form into the following:

$$A(b, c)D. \tag{2.2}$$

When describing the following neutron *(n)* interactions we will omit the identities of the nuclei.

- *Elastic Scattering (n,n)* - Both kinetic energy and momentum are conserved. No additional particles are created.
- *Inelastic Scattering (n,n')* - Some kinetic energy is lost and only momentum is conserved. Some radiation may be emitted due to the deposited energy into the nucleus.
- *Fission (n,f)* - The resulting nucleus is unstable and almost spontaneously decays into several fragments including neutrons. There are several fissile isotopes in nature including U-235 and Pu-239.
- *Radiative Capture (n,γ)* - The absorption of the neutron raises the nucleus into an excited state where it emits gamma radiation in order to lower itself into a stable state.

There are additional absorption reactions that may release alpha particles and neutrons. These *(n,n*)* are important for the sustained fission reaction as they contribute to the overall neutron economy [15, p. 15].

2.1.2 Neutron Cross Sections

The probability of a neutron undergoing a particular interaction is given by its cross section. The cross section can also be conceptualized as the "effective area" that the particle sees. To further quantify this consider a beam of neutrons of uniform speed and direction at a normal incident to a thin slab of material. We can quantify the reaction rate R as such:

$$R\left(\frac{\#}{cm^2 s}\right) = \sigma\left(cm^2\right) I\left(\frac{\#}{cm^2 s}\right) N_A\left(\frac{\#}{cm^2}\right), \tag{2.3}$$

where σ is the microscopic cross section, I is the intensity and N_A is Avogadro's number. From the units we see that the proportionality constant σ is an area. Classically we can think of the cross section as the target nuclei's area impeding the incident beam. A nuclear radius is roughly 10^{-12}cm meaning the geometrical area would be on the order of 10^{-24} cm^2, this magnitude is referred to as a barn and nuclear cross sections are

expressed in this unit. Due to resonance which is a product of the quantum-mechanical nature of the nuclei cross sections can be much larger or smaller than a barn [15, p. 17-19].

Cross sections are also available in terms of which reactions may take place. We can define a total cross section as the sum of the absorption and scattering cross-sections:

$$\sigma_t = \sigma_a + \sigma_s. \tag{2.4}$$

The absorption and scattering terms can be split even further into elastic, inelastic, fission and radiative capture:

$$\sigma_t = \sigma_f + \sigma_r + \sigma_e + \sigma_{in}. \tag{2.5}$$

Given a target and a neutron of energy E the cross section governing the reaction is strongly dependent on E, that is $\sigma = \sigma(E)$. For example the fission cross section of U235 is low at high energies and high at low energies. This feature is essential to the design of nuclear reactors.

Simple models such as the Ramsauer Model provide estimates to the energy dependence [16]. The idea is that the neutron's effective size is proportional to its DeBroglie wavelength which is a function of energy:

$$\lambda(E) = \frac{h}{\sqrt{2mE}}. \tag{2.6}$$

Taking the wavelength λ to contribute to the effective radius we can make the proportionality relation:

$$\sigma(E) = \left(R + \lambda(E)\right)^2 \pi, \tag{2.7}$$

where R is the nuclei's effective radius.

2.1.3 Resonance Cross Sections

Another important feature of the energy dependence is resonance peaks. These are narrow peaks covering a small energy range. Thus for localized energy regions these peaks spike many times higher than their surroundings. There are so-called resonance regions where many of these peaks exist. These regions play an important role in reactor control. Looking at Figure 2.1 we can see a resonance region at relatively low

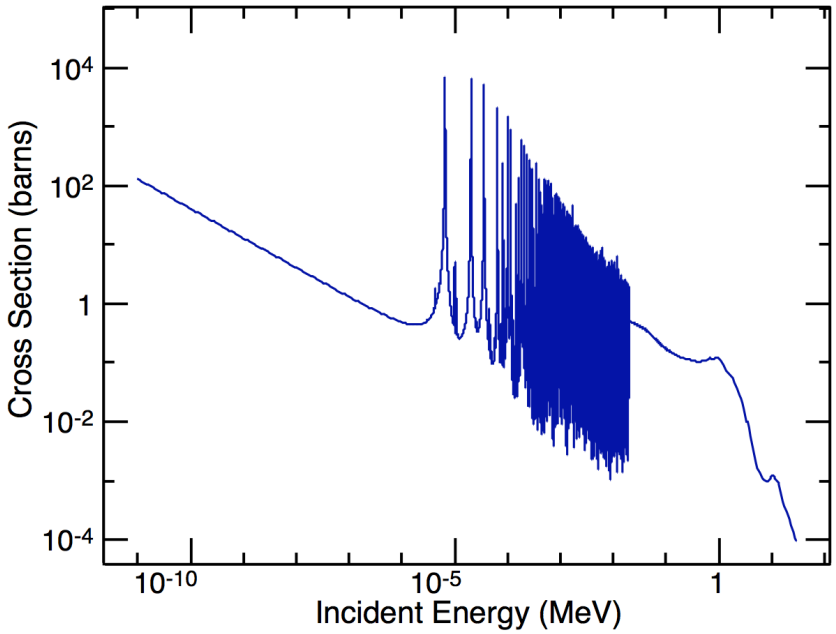

Figure 2.1: Radiative capture cross section of U-238

energies. When the energies of the incident neutron (center-of-mass), nucleus and its neutron binding energy combine to match that of an energy level corresponding to the formed nuclei via neutron capture, the probability drastically increases that this event will occur. The lowest-energy excited states are less than 1 eV above ground state and can go up to around 100 keV for heavy mass fuel nuclei (fissile and fertile). For intermediate-mass nuclei this starts at about 10 eV and for lighter-mass nuclei at 10 keV. Typically the heavier-mass isotopes have many low-energy excited states which give rise to resonances in the neutron absorption and scattering cross sections [17, p. 13-14].

2.1.4 Doppler Broadening

The temperature coefficient of most nuclear reactors is negative due to the effect known as *Doppler Broadening*. As the temperature increases the vibrating motions of the nucleus increase. Thus for an incoming neutron the relative velocity $\vec{v}$ in the rest frame of the nucleus is

$$\vec{v} = \vec{v_{lab}} - \vec{V}, \tag{2.8}$$

where $\vec{v_{lab}}$ is the lab frame velocity and $\vec{V}$ is the nucleus' velocity due to vibrations. Assuming that the nucleus has the same behaviour as an ideal gas in equilibrium at a temperature T, we can approximate its velocity using a Maxwell-Boltzmann distribution. This gives the relation

$$V = \sqrt{\frac{kT}{M}}, \tag{2.9}$$

where k is the Boltzmann constant and M the mass of the nucleus. Thus for higher temperatures the nucleus is moving with greater energy meaning that incoming neutrons with energies that were below or above the resonance energy in the lab frame now have enough energy in the rest frame to form these compound nuclei. This effectively *broadens* these peaks [17, p. 117-122].

In terms of computer simulation, data libraries are imported at lower temperature (typically 0K). To be of use at higher temperatures they must undergo doppler broadening in order to sufficiently match simulation conditions. Some approaches to doing this involve broadening the data library before it is used for simulation, codes such as NJOY have been developed to do this. Other methods involved on-the-fly doppler broadening where the data libraries are reevaluated during the simulation.

2.1.5 Neutron Transport Equation

The neutron transport equation is the fundamental equation describing reactor physics:

$$\begin{aligned}\frac{1}{\nu}\frac{\partial\psi}{\partial t} + \hat{\Omega}\cdot\nabla\psi + \Sigma_t(\mathbf{r},E)\psi(\mathbf{r},E,\hat{\Omega},t) \\ = \int_{4\pi} d\hat{\Omega}' \int_0^\infty dE'\nu'\Sigma_s(E'\to E,\hat{\Omega}'\to\hat{\Omega})\psi(\mathbf{r},E',\hat{\Omega}',t) + s(\mathbf{r},E,\hat{\Omega},t),\end{aligned} \tag{2.10}$$

where:
$\psi(\mathbf{r},E,\hat{\Omega},t)$ is the angular neutron flux,
ν is the neutron speed,
$\Sigma_t(\mathbf{r},E)$ is the total interaction cross-section,
$\Sigma_s(E'\to E,\hat{\Omega}'\to\hat{\Omega})$ is the scattering cross-section,
$s(\mathbf{r},E,\hat{\Omega},t) -$ are the neutron source terms.
There are several important features of this equation. It is a linear equation with unknown dependent variable $\psi(\mathbf{r},E,\hat{\Omega},t)$ and seven independent variables ($\mathbf{r} =$

$x, y, z; E; \hat{\Omega} = \theta, \phi; t)$. This equation is known as an "integrodifferential" equation due to the fact that it contains both derivatives in space and time and integrals over angle and energy.

To extend the previous equation to describe nuclear fission we include a component in the source term to account for fission neutrons. The fission source term S_f is formulated as:

$$S_f(\mathbf{r}, E', \hat{\Omega}', t) = \frac{\chi(E)}{4\pi} \int_{4\pi} d\hat{\Omega}' \int_0^\infty dE' v(E') \Sigma_f(E') \psi(\mathbf{r}, E', \hat{\Omega}', t) \tag{2.11}$$

where:
$\Sigma_f(E')$ is the fission interaction cross section,
$v(E')$ is the average number of fission neutrons produced from an event,
$\chi(E)$ is the fission neutron energy spectrum.
This source term covers the neutrons that are created at the time of fission which are known as *prompt* neutrons. Another source we must account for are the delayed neutrons which can appear much later in time. This delayed source S_d is described as:

$$S_d(\mathbf{r}, t) = \sum_k \chi_k(E) \lambda_k C_k(\mathbf{r}, t), \tag{2.12}$$

where:
$\chi_k(E)$ is the delayed neutron energy spectrum,
λ_k is the decay constant of the k group,
$C_k(\mathbf{r}, t)$ is the precursor concentration of the k group.
In general the source term is made of the prompt and delayed neutron source in addition to any other external source:

$$s = S_f + S_d + Q, \tag{2.13}$$

where Q are any other neutron sources [15, p. 111-117].

2.2 Reactor Kinetics

To operate at a steady state the reactor must be in a complete neutron balance, that is the rate of neutrons being produced via fission must equal the rate that the neutrons are being lost via leakage and absorption. There are several ways for the balance to

shift, normal power changes during operation would have the control rod move and upset the balance. Long term changes could be the depletion of fuel or buildup of absorbing isotopes. More serious changes could result from loss of coolant, broken coolant pump or accidental control rod ejection.

Reactor kinetics is the study of predicting the behaviour of a time-dependent neutron population. It is vital to be able to understand the factors influencing sudden changes in neutron balance and thus reactor stability.

2.2.1 Criticality

The fission chain reaction is the driving force which creates power in a reactor. Neutrons emitted from fission go on to induce fission in other fissile nuclei and so on. This chain reaction is described quantitatively with the *k-multiplication factor*.

$$k = \frac{\text{Number of fissions in one generation}}{\text{Number of fissions in previous generation}}. \tag{2.14}$$

From a "life-cycle" point of view this definition makes sense as it describes the ratio of successive fission generations. This definition fails however when considering the generation lifetime. An individual neutron can take a wide variety of paths before it dies. Some neutrons may immediately fission after birth or they may slow down in the reactor before fissioning. Neutrons can also be unproductive in the generational sense if they are absorbed into the materials or leaked out of the reactor. Thus, a more practical definition of k-multiplication factor can be defined:

$$k_{\text{eff}} = \frac{\text{Rate of Production}}{\text{Rate of Loss}} = \frac{P(t)}{L(t)}. \tag{2.15}$$

From this perspective we look at the "neutron balance" of the reactor. It is explicitly noted that the production and loss rates are functions of time. Changes in the material such as fuel temperature and fuel depletion would have a direct effect on theses rates.

When talking about the k-multiplication factor the term criticality is often used to characterize the behaviour in the reactor. There are essentially three modes of criticality. *Sub-critical* means the neutron population is shrinking, *super-critical* means the population is growing and *critical* means the population is stable. The degree to which the reactor is critical can be described with the k-multiplication factor [15,

p. 74-77].

$$k = \begin{cases} < 1, & \text{sub-critical} \\ 1, & \text{critical} \\ > 1, & \text{super-critical} \end{cases} \tag{2.16}$$

2.2.2 Reactor Period

The change in neutron population can be described with the following equation:

$$\frac{dN(t)}{dt} = P(t) - L(t). \tag{2.17}$$

Rearranging this equation,

$$\frac{dN(t)}{N(t)} = \frac{L(t)}{N(t)} \left(\frac{P(t)}{L(t)} - 1 \right), \tag{2.18}$$

and before solving we define the average neutron lifetime l,

$$l = \frac{N(t)}{L(t)}, \tag{2.19}$$

and substitute $k = P(t)/L(t)$ to obtain

$$N(t) = N_0 \exp\left(\left[\frac{k-1}{l} \right] t \right). \tag{2.20}$$

From this equation it's clear that the neutron population will evolve with exponential growth or decay through time. It is useful to define the following as the reactor period:

$$\tau = \frac{l}{k-1}. \tag{2.21}$$

Thus when following a growing or decaying neutron population in time, one can use the reactor period to find the criticality of the system [15, p. 77-78].

2.2.3 Reactivity

Reactivity (ρ) is a measurement that represents the deviation of the reactor from critical state ($k = 1$):

$$\rho(t) = \frac{k(t) - 1}{k(t)}. \tag{2.22}$$

Here we see that both the reactivity and k-multiplication factor may be function of time.

2.2.4 Precursors and Delayed Neutrons

The neutrons appearing nearly instantaneously are called *prompt* neutrons and for the application of our simulations are set to appear at the time of fission. There are other neutrons which may appear milliseconds to even minutes from fission; they are called *delayed* neutrons. After a neutron is absorbed into a nucleus the resulting isotope may have a higher energy than the binding energy of a neutron, this isotope may then break into fragments releasing one or several neutrons. There is also an option for the newly formed nucleus to beta decay into other entities before fragmenting into smaller pieces and neutrons. The beta decay may happen after some time hence why the resulting neutrons are called delayed.

The creation of delayed neutrons is of great importance in reactor control. For example if we had a reactor with solely prompt neutrons controlling the reaction the average neutron lifetime l may be on the order of 0.1 ms for a light water reactor using enriched uranium. For a small reactivity change of +1 mk from critical ($k = 1.001$) we can calculate the increase in neutron population in 1 second using equation 2.20.

$$\frac{N(t = 1s)}{N_0} = e^{\left(\frac{1.001-1}{0.0001}1\right)} = e^{10} \approx 2.2 \times 10^4. \tag{2.23}$$

Thus in 1 second the neutron population and therefore power increase by a factor of over 10000. With delayed neutrons added to the calculation we can take the average neutron lifetime to be much longer $l \approx 0.1$ s. Recalculating we find that

$$e^{\left(\frac{1.001-1}{0.1}1\right)} = e^{0.01} \approx 1.01. \tag{2.24}$$

A power increase of only 1% would occur with the delayed neutron addition. During normal reactor operations the reactor is kept slightly sub-critical on prompt neutrons

to allow the delayed neutrons to sustain the reaction.

2.2.5 Point Kinetics Equations

The following are the point kinetic equations:

$$\frac{dn}{dt} = \left[\frac{\rho(t) - \beta}{\Lambda}\right] n(t) + \sum_{i=1}^{6} \lambda_i C_i, \tag{2.25}$$

and

$$\frac{dC_i}{dt} = \frac{\beta_i}{\Lambda} n(t) - \lambda_i C_i, \tag{2.26}$$

where: $n(t)$ is the neutron density,
i is the index of the group that contains precursors with similar decay times.
C_i is the concentration of ith precursor group,
$\rho(t)$ is the reactivity,
λ_i is the ith precursor group decay constant,
β is the total delayed neutron fraction,
β_i is the ith group delayed neutron fraction,
Λ is the mean generation time.

This set of seven differential equations describe both the time-dependent neutron population and precursor concentrations. At first glance it may seem that the equations can be solved in a straightforward manner however that is not the case as the reactivity term $\rho(t)$ is a function of time itself and is directly influenced by the neutron density. Therefore the solutions to these equations are non-linear in nature [18].

2.3 Heat Transfer In the Reactor

Heat transfer is the transmission of thermal energy due to spatial differences in temperature. Whenever a temperature gradient is formed in a medium or between media heat transfer will occur. There are three different forms of heat transfer that can occur.

Conduction is the transfer of energy from more energetic particles to less energetic particles due to the interactions they have between them. If a metal rod was at a higher temperature at one end, via conduction the heat would dissipate through the rod over time until the a uniform temperature is reached. The rate at which heat

conduction occurs is described by *Fourier's Law*

$$\vec{q} = -k\nabla T. \tag{2.27}$$

Here $\vec{q}$ is the heat flux, k is the thermal conductivity of the medium and T is the temperature.

Convection is the transfer of heat due to the diffusion of thermal energy between particles and the energy that is transferred by the macroscopic motions of the medium, which is typically a fluid. The fluid motion means that at any moment a large number of particles are travelling in collective groups. In the presence of a temperature gradient this macroscopic movement contributes to the overall heat transfer. Since the individual particles in the collective groups still have random motion the total heat transfer is a superposition of the bulk fluid motion and random motion of the particles. It should be noted that when referring to the heat transfer solely due to the bulk motion of fluid the term *advection* is used. The rate equation that is used when describing convection is called *Newton's Law of Cooling*:

$$q'' = h\left(T_s - T_\infty\right), \tag{2.28}$$

where q'' is the convective heat flux, h is the heat transfer coefficient, T_s is the surface temperature, and T_∞ is the bulk fluid temperature.

Unlike the previous modes of heat transfer *thermal radiation* does not require a medium, in fact it is most efficient in a vacuum. Thermal radiation is the emission of energy from matter. This emission is due to changes in electron configuration in the molecules or atoms that make up the material. The energy is transferred via electromagnetic waves and can be described using the *Stefan-Boltzmann Law*.

$$E = \epsilon\sigma_s T^4, \tag{2.29}$$

Where E is the surface emissive power, ϵ is the emissivity, σ_s is the Stefan-Boltzmann constant and T_s is the absolute surface temperature [19, p. 10-22].

Variations in the thermal-hydraulic properties of the reactor materials can have a significant effect on the neutron flux. Changes in density, volume and temperature affect the neutron cross sections and these properties are particularly important in the fuel, coolant and moderator. A wide variety of computer codes have been developed to

predict the thermal-hydraulic properties under steady-state and transient conditions. Thus it is up to the user to determine which code is best suited for their objective as the models and methods vary [20].

2.4 The Monte Carlo Method

Monte Carlo is a class of mathematical methods applying computational power to obtain numerical results via repeated random sampling. In general any problem having a probabilistic interpretation may be solved using these methods. In physics-related problems Monte Carlo methods are used to simulate systems with many coupled degrees of freedom such as fluids, coupled solids and interacting particle systems. Applications in mathematics, business, space-exploration and risk analysis can also be found using Monte Carlo methods.

The statistical nature of Monte Carlo means its uncertainties are statistical. In principle Monte Carlo uses the law of large numbers to approximate an integral that describes the expectation value of a random variable. This is achieved by taking the empirical mean of independent samples of the variable. The statistical error generated from this method is related to the variance of the measurements [1, p. 295-297].

2.4.1 A simple example: π approximation

To illustrate the method of Monte Carlo a simple algorithm for approximating the value of π is used.

To generate the approximation we look at the unit circle on the Cartesean plane and focus on only the upper right positive quadrant. We restrict our range to [0,1] on both axis. π can be interpreted as the ratio between the area of the quarter circle and the total area:

$$\frac{\text{Quarter Circle}}{\text{Total Area}} = \frac{\frac{1}{4}\pi 1^2}{1^2} = \frac{\pi}{4}$$

To use this relation we generate random points on our graph and define the ratio between the number of points landing inside the circle and total number of points as $z/4$. We interpret the approximation of π as z. Figure 2.2 shows the value of z for varying number of points.

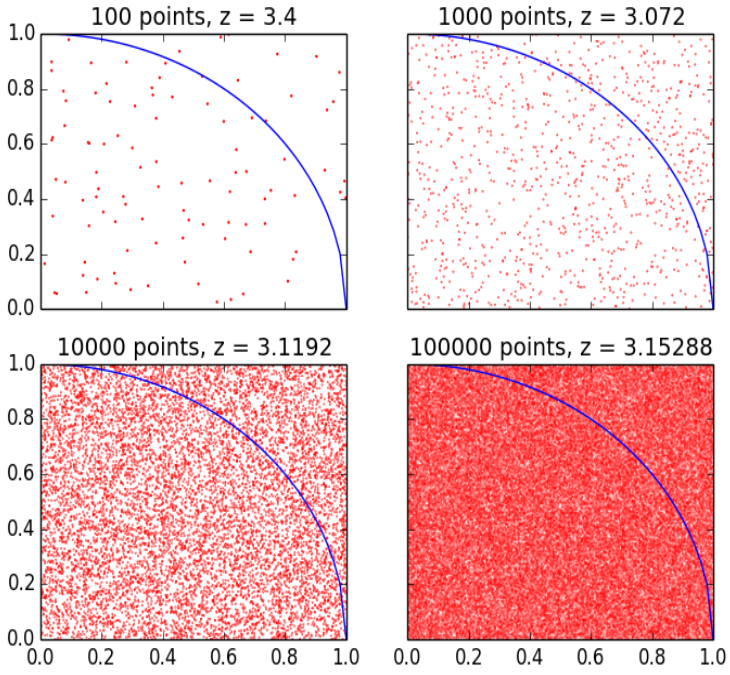

Figure 2.2: Approximation of π.

This example outlines some important aspects of the Monte Carlo method. The method relies on randomly sampled numbers, and the accuracy of the result increases with more samples. We can see that with 100000 points our approximation was good to only one decimal place. Since Monte Carlo relies on the generation of large sets of random numbers it is more computational expensive than deterministic methods.

Chapter 3

Methodology

3.1 Simulation Flow

In G4-STORK a simulation is first broken down into a series of N_r runs, each run covers a set time duration T. Between two runs tallies are scored, calculations are made and the geometry is adjusted if desired. Each run comprises of N_e events which must be simulated for the run to end. A flow chart of the simulation process can be seen in Figure 3.1.

Similar to other Monte Carlo codes, G4-STORK tracks particles as they move in a series of steps with instantaneous interactions occurring at the end of each step. Processes are handled very generally as it enables the user to define their own. A process must do the following: first declare whether it is applicable given the current particle status and material. If so, it then must propose a valid step length ([0,10^{302}] mm for a neutron). If selected to be modelled it must model its interaction by proposing changes to the particles attributes (state, momentum, etc.). If secondary particles are to be modelled then the process must also define their starting attributes [21].

To allow the simulations to be stopped at a set time a time step limiter process was created. The limiter works by calculating a max step length $d_{\max}$ the particle can take.

$$d_{\max} = v_p \times (t_{run} - t_p), \tag{3.1}$$

v_p is the current velocity of the particle, t_{run} is the time of the end of the run, and t_p is the current time of the particle. As the simulation continues the difference between

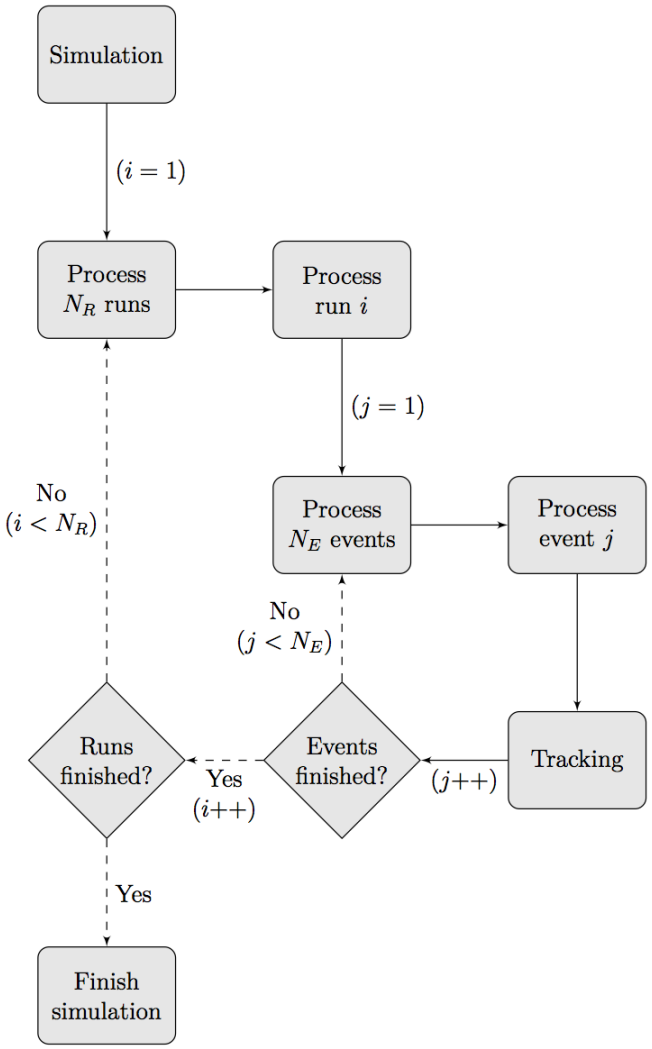

Figure 3.1: Simulation flow in G4-STORK [4].

the times will eventually reach zero, effectively stopping the simulation of that track [22].

3.1.1 Neutron Population Stabilization

Neutron populations in a reactor are subject to exponentially grow or shrink in a short period of time. In our simulation the neutron population $N(t)$ will change as the following function of time:

$$N(t) = N_0 \exp\left(\frac{k_{\text{eff}} - 1}{l} t\right), \tag{3.2}$$

where N_0 is our starting population and l is the average neutron lifetime. For a neutron population with a k_{eff} not sufficiently close to 1 the population will grow or shrink by a factor of e (≈ 2.7) for every l time that passes. Neutron lifetimes can be on the order of nano to milliseconds which means that in our simulations the populations can grow large enough to take up all computer memory or can shrink to statistically insignificant numbers. To remedy this a renormalization scheme was developed in order to keep the population stable [4].

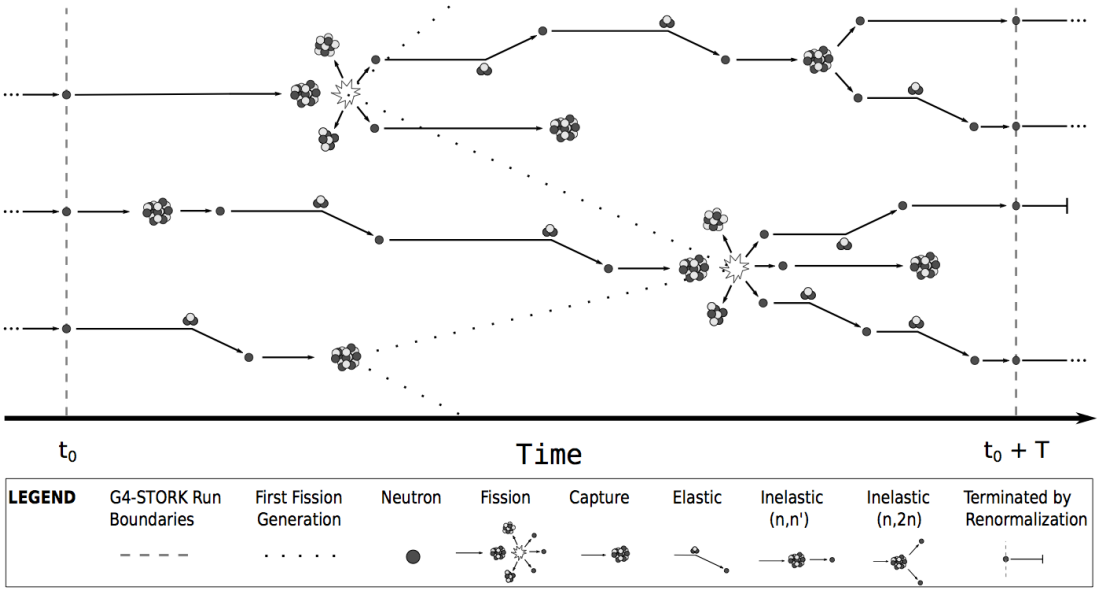

Figure 3.2: The simulation and renormalization of primaries.

At the beginning of the simulation a defined number of neutrons N is taken

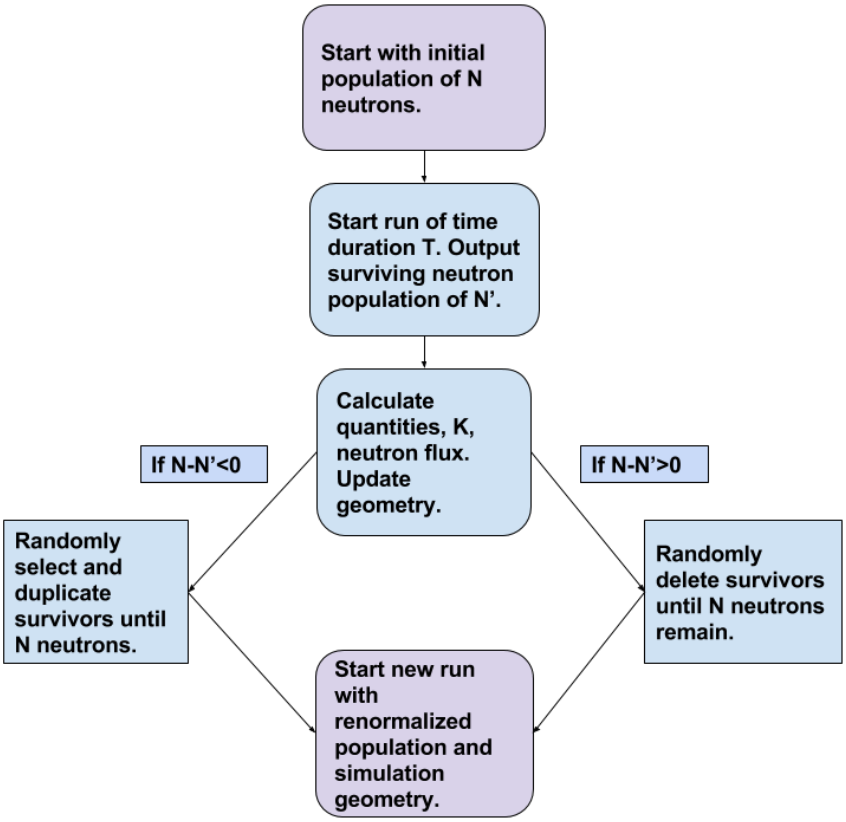

Figure 3.3: Simulation flow in G4-STORK.

to be the renormalizing population number, this is the number of primaries in our simulation. Between runs the surviving neutrons, deemed survivors are subjected to renormalization. If the number of survivors is greater than the number of primaries we randomly delete survivors until they are equal. Conversely if the number of survivors is less than the number of primaries we randomly select and duplicate members of our surviving population to match the primaries. This process can be seen in Figure 3.3.

3.1.2 Error Calculation

The statistical error reported from simulation codes is usually the easiest to use. As the larger the sample size or number of neutrons the lower the error becomes:

$$\delta\hat{x} = \frac{\sigma}{\sqrt{N}} \tag{3.3}$$

where $\bar{x}$ is an average value and $\delta\bar{x}$ its error, σ is the standard deviation and N is the sample size used for the average.

It is in our best interest to increase the number of neutrons we use in our simulation to decrease the statistical error. There are however errors that arise from the discrepancies in the nuclear data, differences in geometry/materials as well as the methodologies used to simulate the nuclear processes. These errors which are known as code or data bias account for some of the differences between G4-STORK and other simulation codes as well as real experiment. Each code is subject to their own bias as they may used different data libraries with varying degrees of accuracy and various methods.

3.1.3 The K-Multiplication Factor

The dynamic K-effective multiplication factor is calculated using the current production and loss rates at the end of each run.

$$k_{\text{eff}} = \frac{P(t)}{L(t)}. \tag{3.4}$$

$P(t)$ and $L(t)$ are the production and loss rates. We can transform this calculation by assuming that these rates are constant in a run of time length T.

$$k_{\text{eff}} = \frac{P(t) \times T}{L(t) \times T}. \tag{3.5}$$

This calculation is provided at the end of each run and relies on the tallying of the production and loss of neutrons. This method holds true if the geometry and materials of the reactor remain static during the T time period in which the tallies are scored.

3.1.3.1 Neutron Flux

In the calculation of neutron flux in G4-STORK the positions and momenta of the surviving neutrons contribute equally. The user is required to input the volume of interest and the energy range of interest.

$$\phi = \frac{\sum_i \nu_i}{V}. \tag{3.6}$$

ν_i are the velocities of the neutrons falling within the energy range and V is the volume of interest.

A correction factor must be calculated in order to properly scale the flux relative to the operating power of the reactor. This scaling factor is given as follows:

$$\text{Scaling Factor} = \frac{\text{Operating Power} \times T}{\#\text{ of fissions} \times \text{Energy per fission}}, \tag{3.7}$$

where T is the time duration of the run and the energy per fission is taken as 198 MeV.

The flux may be calculated between runs for a general volume a interest to view the flux as a function of time. The volume of interest V can also be split into smaller volumes in order create a spatial plot of the flux.

Algorithm 1 outlines the flux calculation from a survivor neutron population. E_i, r_i and ν_i are the energy, position and velocities of the ith entry of the n_s length survivor list. The algorithm first checks that the survivor is in the integration region and specified energy range. Then calculates the simulated flux, the scaling factor S_f is then used to find the actual flux.

Algorithm 1: Calculation of neutron flux.

Data: n_s,r_i,E_i,V,ν_i, S_f
Result: ϕ

```
for i → n_s do
    if IsInRegion(r_i) and IsInEnergyRange(E_i) then
        φ_sim += ν_i;
    end
end
φ = S_f × φ_sim/V;
return;
```

3.1.4 Shannon Entropy

To quantitatively measure the convergence of our simulations the *Shannon entropy* is calculated between runs in G4-STORK. Shannon entropy is the expected value or average amount of information we receive from a message or packet [23]. The entropy of the fission sites and survivors are measured to determine if convergence is reached. If the entropy between runs has converged than it is concluded that the source has also converged.

To calculate the entropy (H) we discretize our simulation world into N_b identical boxes and then apply the following equation:

$$H = -\sum_{j=1}^{N_b} P_j \log_2 P_j, \tag{3.8}$$

where P_j is the probability of a site being in that box j. We define this probability as:

$$P_j = \frac{n_s^j}{N_s}, \tag{3.9}$$

where n_s^j is the number of sites in box j and N_s is the total number of sites.

If a box contains no sites then it does not contribute to the overall calculation, similarly if all sites are contained in one box the overall entropy would be zero. If the sites were uniformly distributed among all the boxes then a maximal value of the entropy would be reached.

$$H_{max} = -\log_2 \left(\frac{1}{N_b}\right). \tag{3.10}$$

3.1.5 On-the-fly Doppler Broadening

G4-STORK uses its own engine to broaden the cross-sections during simulations. To find the broadened cross-section an iterative process is used. First the speed of the nuclei is determined by sampling from a Gaussian distribution centered around its most probable speed. This most probable speed comes from the Maxwell-Boltzmann distribution and is a function of the nuclei's mass and temperature. Using the difference between the nuclei's sampled speed and the lab frame particle speed we find a new rest frame energy which is used as input for a cross-section. This process is done N times, with each cross-section contributing to an overall averaged cross-section $\bar{\sigma}$. The algorithm finishes once the convergence limit is reached.

Since N is taken as the maximum between 10 and $T/60$, with increasing temperatures this method becomes increasing inefficient as N becomes larger, thus a data library that is near the temperatures of interest is preferable. Algorithm 2 outlines the implementation of this method.

Algorithm 2: On-the-fly-Doppler Broadening.

Data: $\vec{v}$,δ, T
N = max(10, T /60);
while $\bar{\sigma} > (1 \pm \delta)\sigma_{sum}/count$ **do**
  $\bar{\sigma} = \sigma_{sum}$/count;
  for $i \to N$ **do**
    r = RandGaussian($\mu = 1$);
    $\vec{V} = r(kT/M)^{1/2}$;
    $\vec{v'} = \vec{v} - \vec{V}$;
    $\sigma_i = \sigma(\frac{1}{2}mv'^2)$;
    σ_{sum} += σ_i;
  end
  count += N;
end
return $\bar{\sigma}$;

3.2 Time-dependent Simulation World

To link the simulated reactor with the real reactor, inputs from the SLOWPOKE were needed. In G4-STORK we must link two different states of the reactor. We start at one of these states and use the physics models to allow the reactor and its neutrons to evolve in time into the next reactor state. In terms of the SLOWPOKE-2 we have the rod-in and rod-out reactor states. At rod out we assume that the reactor is in a steady state mode of operation, this initializes the properties of the transient simulation. It allows us to establish a steady state simulated neutron flux, precursor population, fuel temperature and density, and other thermal-hydraulic properties in the reactor.

3.2.1 Neutron Power

At any given time the neutron power $P_n(t)$ will be directly linked with the current fission rate $R_f(t)$ in the reactor.

$$P_n(t) \propto R_f(t). \tag{3.11}$$

To equate these two we create a proportionality constant.

$$P_n(t) = C_n \times R_f(t). \tag{3.12}$$

The coefficient C_n converts the fission rate into power that is related to the actual power of the reactor. This number essentially determines how much energy each simulated fission deposits. This is to compensate for the lack of neutrons in our simulation compared to the real reactor. Hence each simulated fission is a representation of many real life fissions. Ultimately this number must be user set to initialize a connection between fissions and reactor power.

To calculate this coefficient we take $t = 0$ to be a full power steady state operation:

$$P_n(t = 0) = P_{n,0}, \tag{3.13}$$

$$R_f(t = 0) = R_{f,0}, \tag{3.14}$$

allowing us to solve for the conversion coefficient,

$$C_n = \frac{P_{n,0}}{R_{f,0}}. \tag{3.15}$$

3.2.2 Lumped Capacitance Method for Fuel Pin Modelling

To model the transient heat conduction of the fuel pins the method of Lumped Capacitance was employed [24]. The Lumped Capacitance method is an approximate solution to the transient conduction of heat between a solid and its surroundings. It is assumed that the temperature of the solid is spatially uniform at any point in time during the transient. According to Fourier's law this would imply that the thermal conductivity in the solid is effectively infinite, this is obviously incorrect but can be used as an approximation if the conductive resistance within the solid is small relative to the resistance between the heat transfer of the solid and its surroundings [19, p. 256-267]. To make this assumption one must calculate the Biot number and verify it is within the correct range of values, this verification is done in Appendix B.

To derive the change of temperature equations for the fuel pins we begin by using the energy balance equation.

$$\frac{dE_{in}}{dt} - \frac{dE_{out}}{dt} = \frac{dE_{st}}{dt} \tag{3.16}$$

The neutron fission power is the driving term for the energy generated within the pin. To simplify the problem we lump together the fuel assembly and air gap to act

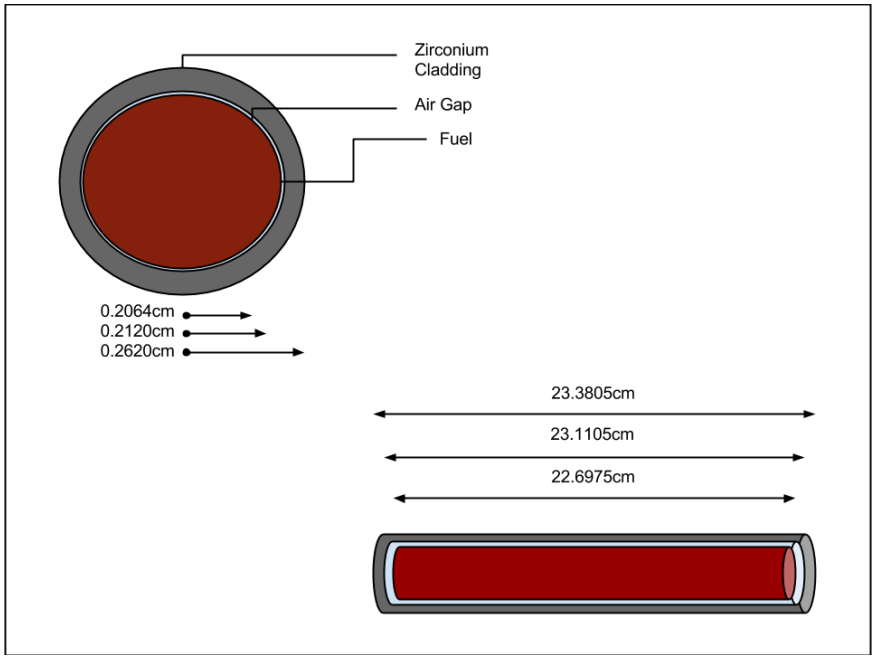

Figure 3.4: Cross-sectional view of SLOWPOKE fuel pellet.

together as a heat transfer medium. Thus the heat lost in the fuel is via convection into the surrounding coolant. Our equation is now:

$$\dot{Q} - hA(T - T_\infty) = \rho c V \frac{dT}{dt}, \tag{3.17}$$

where $\dot{Q}$ is the fission power,
h is the heat transfer coefficient,
A is the contact area,
T_∞ is the coolant temperature,
ρ is the fuel density, c is the heat capacity and
V is the fuel volume.

This first-order differential equation provides us with the framework to model the change in fuel temperature with input from the fission rate of the reactor. This link is a necessary step in simulating the transient temperature feedback mechanism seen in the SLOWPOKE.

3.2.3 Transient Solution to Differential Equations

The differential equations that govern the behaviour of the main variables can be expressed as follows:

$$\dot{x} = a(x) \cdot x + b(x) \cdot u(x,t), \tag{3.18}$$

where we consider $u(x,t)$ to be the driving term.

To obtain a closed form solution, a , b, and u must be constant in the time interval $t + \Delta t$. Taking the Laplace transform yields the following.

$$x(s) = \frac{x_0}{s-a} + \frac{bu}{s(s-a)}, \tag{3.19}$$

and

$$x_0 = x(t_0). \tag{3.20}$$

The time dependent solution is obtained from taking the inverse Laplace transform of the previous equation,

$$x(t) = x_0 e^{at} + bu \int_{t_0}^{t} e^{-at} dt. \tag{3.21}$$

Using our previous assumption with our integral over the interval $t + \Delta t$ we find the following difference equation,

$$x_{k+1} = x_k \cdot e^{a_k \Delta t} + \left(e^{a_k \Delta t} - 1\right) \cdot (bu/a)_k. \tag{3.22}$$

3.2.4 Updating Fuel Properties

To account for changes in the geometry and materials as a result of transient operations in the reactor we model the density and volume response of the fuel. To update the density we rely on empirical relations which describe the density as a function of temperature [25].

The formula is described below:

$$\rho_{UO_2}(T) = \frac{\rho_{UO_2}(T = 273K)}{(9.99672 \times 10^{-1} + 1.179 \times 10^{-5} T - 2.429 \times 10^{-9} T^2 + 1.219 \times 10^{-12} T^3)^3},$$

where $\rho_{UO_2}(T = 273K) = 10970$ [kg/m^3].

When updating the volume of the fuel we must ensure that the overall mass of the fuel is kept constant. Thus as the density evolves with time as a function of

temperature we can ensure a constant mass by taking the product of the densities and volumes. At any given time t we can express the fuel mass M.

$$M_{UO_2}(t) = \rho_{UO_2}(t)V_{UO_2}(t). \tag{3.23}$$

We link the density and volume changes between timesteps by:

$$\rho_{UO_2}(t_1)V_{UO_2}(t_1) = \rho_{UO_2}(t_2)V_{UO_2}(t_2). \tag{3.24}$$

Another simplification to make in the code is to assume that the rod only expands radially. This places a restriction on how much expansion the fuel can have as eventually it will touch the zirconium sheath. Referring to Figure 3.4 the fuel pellet can only expand as far as 0.212 cm, the inner radius of the zirconium sheathe.

An effect in the fuel properties that we have not accounted for is the isotopic composition that changes as the the neutrons interact with the U-235 and U-238 and form fission byproducts. Due to the much shorter timescale of the SLOWPOKE's transients compared to the much longer timescale that fuel burn-up occurs on we can safely neglect modelling this effect [18].

3.3 Delayed Neutrons

The production of delayed neutrons is important for reactor control. They allow reactors to be controlled with mechanical means as outlined in Section 2.2.4. G4-STORK provides the user three options for simulating delayed neutrons: a natural creation method, instantaneous creation and precursor decay method.

3.3.1 Natural Creation Method

The GEANT4 toolkit provides the final state mechanism for generating prompt and delayed neutrons as a resultant of fission reactions. Based on the material cross-section and incoming neutron energy a delayed neutron has the probability to be created. The delayed neutrons produced via this method will have a birth time that may be on the order of 10^{-2} to 10^{1} seconds in the future.

In G4-STORK the delayed neutrons are set aside and saved, they are loaded as survivors and set to appear in the simulation according to their birth time. Due

to the relatively long birth times compared to simulation time a problem arises in which the simulation lacks a properly converged delayed neutron source. In theory to generate such a source we would require a simulation that replicates the start-up of the real reactor. The amount of computational time needed for this is unrealistic. Therefore this method may not be properly utilized until G4-STORK's speed improves dramatically.

Precursor Group	Decay Constant (1/s)	Fission Yield (10^{-3}/fission)
1	0.013336	0.6346
2	0.032379	3.557
3	0.12078	3.14
4	0.30278	6.797
5	0.84949	2.138
6	2.8530	0.4342

Table 3.1: Delayed Neutron data for U-235 [26].

3.3.2 Instantaneous Creation Method

The instantaneous creation method is a solution to the problem of natural creation method described above. If a delayed neutron is set to be created this method will instead force it to appear instantly along with the prompt neutrons. Assuming that the reactor is in a critical or near critical state this method takes advantage of the fact that the delayed neutrons should be appearing at a constant rate proportional to the current fission rate. However due to this fact this method is unsuitable to simulate transient behaviour. Once a reactor undergoes a transient it will over- or underestimate the delayed neutron source. Thus another method is needed if a transient simulation is to be performed.

3.3.3 Precursor Decay Method

G4-STORK follows time-dependent populations of precursors. The governing equation is given as:

$$\frac{dC_i}{dt} = -\lambda_i C_i + \gamma_i R(x,t), \tag{3.25}$$

where C_i is our precursor population of group i, λ_i the decay constants, γ_i fission yield and $R(x,t)$ is the fission rate of U-235 in units of fissions/s/cm^3. Table 3.1 shows

these values for U-235. To generate an initial population we assume a steady-state condition. G4-STORK generates an output file of all fission locations and incident energies. To use this we define our fission rate as:

$$R(x,t) = \sum_{i}^{n_f} \delta(x - x_i)\delta(t - t_i), \tag{3.26}$$

where n_f is the number of fissions. Integrating over time and space:

$$C_i = \alpha_i n_f, \tag{3.27}$$

where α_i is the effective precursor yield:

$$\alpha_i = \frac{\gamma_i}{\lambda_i T}, \tag{3.28}$$

where T is the time duration of the input simulation. The total precursor yield is then defined:

$$\alpha_{tot} = \sum_{i} \alpha_i. \tag{3.29}$$

Algorithm 3 is used to create the starting distribution from a previous simulation output.

A similar procedure is used to add precursors between runs, at run j the number of precursors added:

$$C_i \mathrel{+}= \gamma_i n_j. \tag{3.30}$$

Algorithm 4 is used to add precursors. In both algorithms a fission event has the chance to generate a precursor from any of the 6 groups. Thus the dice are rolled first to determine if a precursor is created using the total fission yield. Then if true, dice are rolled again to determine which group the precursor belongs to using the ratios of group fission yield to total fission yield.

Once we have an established population of precursors Russian roulette is used to remove them and generate delayed neutrons. To determine whether a precursor decays we roulette to find the time of decay:

$$T_{decay} = \frac{-\ln(r)}{\lambda}, \tag{3.31}$$

where r is a random number in the interval (0,1]. The roulette is run for every precursor

Algorithm 3: Creation of initial precursor population.

Data: α_i, α_{tot}, n_f
Result: C_i

```
for k=0 → n_f do
    r = random ∈ (0,1);
    if r < α_tot then
        r = random ∈ (0,1);
        for i=1 → 6 do
            α_ratio += α_i / α_tot;
            if r < α_ratio then
                C_i ++;
                break;
            end
        end
    end
end
return;
```

Algorithm 4: Creation of precursors between runs.

Data: γ_i, γ_{tot}, $n_{f,run}$
Result: C_i

```
for k=1 → n_f,run do
    r = random ∈ (0,1);
    if r < γ_tot then
        r = random ∈ (0,1);
        for i=1 → 6 do
            γ_ratio += γ_i / γ_tot;
            if r < γ_ratio then
                C_i ++;
                break;
            end
        end
    end
end
return;
```

and if the decay time (T_{decay}) is less than the upcoming run duration a delayed neutron is added to the survivor list and a precursor is removed. To generate a delayed neutron the fission site list that is held in G4-STORK is randomly sampled and used to provide a starting position. The birth time is taken as the sum of the decay time and the start of the upcoming run. Finally the momentum is sampled from data taken from literature.

Algorithm 5: Precursor decay

```
Data: C_i,λ_i,T_run
Result: DNList
for i=1 → 6 do
    for j=1 → C_i do
        r = random ∈ (0,1);
        T_decay = −ln(r)/λ_i;
        if T_decay < T_run then
            dNeutron = CreateDelayedNeutron();
            DNList.append(dNeutron);
            C_i − −;
        end
    end
end
return;
```

It is important to note that when used in conjunction with the population renormalization scheme the precursors must be combed in the same manner. Thus after run j the combed precursors are given by equation 3.32.

$$C_i^{j+1} = C_i^j \times \frac{1}{k_{\text{run}}^j}. \tag{3.32}$$

Due to the isotope composition of the fuel in the SLOWPOKE-2 the primary fissioning isotope is U-235 from thermal neutrons. Thus a simplification was made in using only six precursor groups based on U-235.

Chapter 4

Results

4.1 Control Rod Worth

To calculate the reactivity worth of the central control rod of the SLOWPOKe two simulations were run with the same input but different control rod placements. An initial point source of 20,000 neutrons was used, placed in the center of the reactor core using a Gaussian distribution of energies centred around 2.0 MeV emulating the neutrons produced by prompt fission. The simulations each underwent a single run of 2 ms allowing sufficient time for the distribution to converge. The control rod was moved out by 20.66 cm and the effective multiplication factor was used to calculate reactivity.

$$\text{Control Rod Worth} = (4.9 \pm 2.0)\ \text{mk}$$

The control rod worth was experimentally measured to be 5.45 mk [11].

The error stated was generated from assuming that the production and loss processes during the simulation followed a Poisson distribution, that is both the production and loss rates are constant and production and loss events are independent. This assumption is reasonable given the simulation time of 2 ms.

4.2 Neutron Flux

The neutron fluxes are plotted in Figures 4.1 and 4.2 at 20 kW of power and for the thermal neutron spectrum (0.00 - 0.04 eV) as well as the cadmium absorbing spectrum (0.00 - 0.40 eV). It should be noted that the actual flux for the rod-in case is zero since the reactor is not at power but for comparison reasons the rod-in flux was scaled in the same manner as the rod-out case.

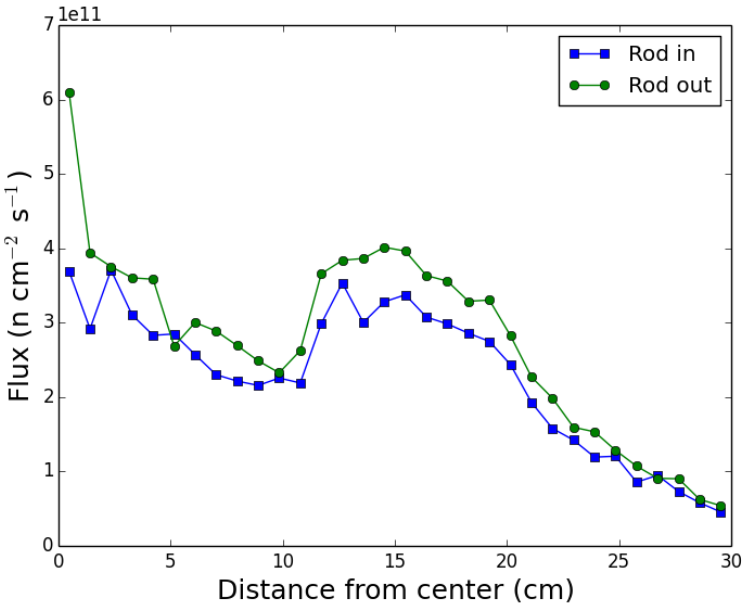

Figure 4.1: Neutron flux vs distance in the thermal energy region.

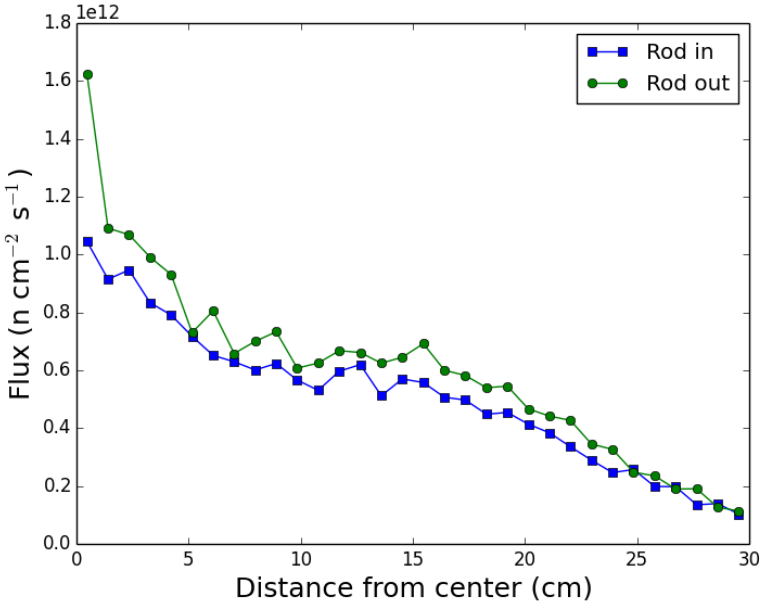

Figure 4.2: Neutron flux vs distance in the cadmium absorbing region.

4.3 Transient Simulation of Rod Removal

Figure 4.3 shows the flux response from a rod pluck-out of the reactor. A 20 ms time average was used to smooth out the original plots (Figures 13 and 17). The pluck-out begins at approximately time $t = 200$ ms and immediately we see an increase in flux before it begins to level off. We can see that with the rod left in the reactor it is still at power but only with the decaying precursors keeping the flux alive. As the precursor source is turned off the flux quickly dies out.

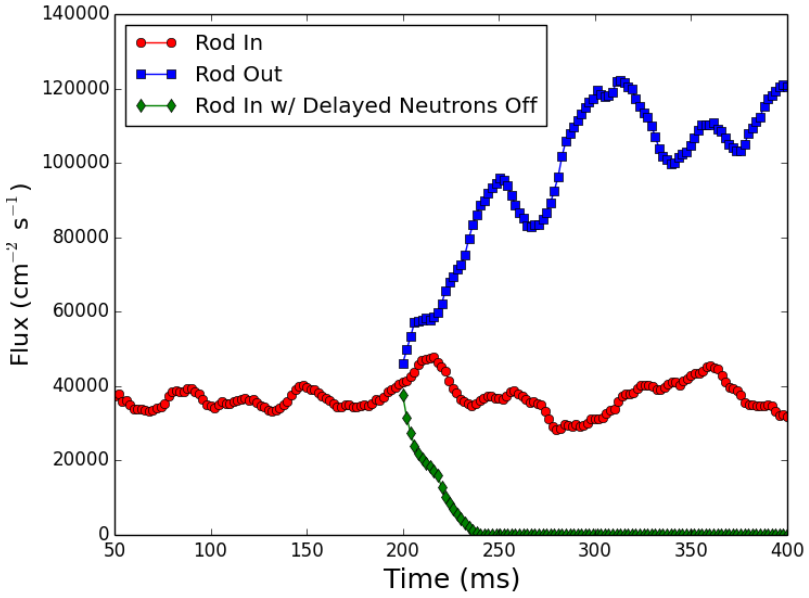

Figure 4.3: Neutron flux during rod movement and delayed neutron shutoff.

From Figure 4.4 the rod left in we see that the precursor population is decreasing indicating that after a sufficient amount of time the reactor will shut off. When pulling the rod out the population increases due to an increased fission rate.

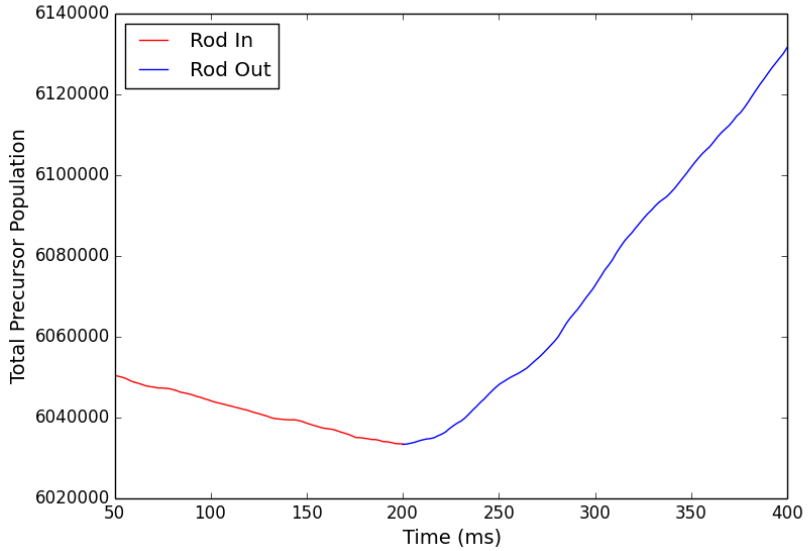

Figure 4.4: Total precursor population during rod movement.

The multiplication factor change from rod movement is shown in Figure 4.5.

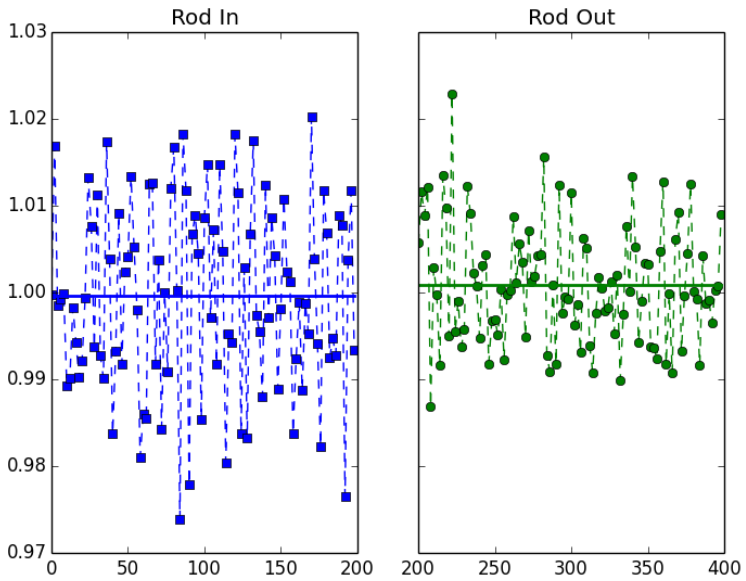

Figure 4.5: K-effective during rod pluck out. The averaged multiplication factors are 0.9980 and 1.0008 for rod-in and rod-out.

The averaged k-effective values are to illustrate the increase in reactivity in the system, however they may not represent an accurate measurement. The standard deviations are quite high with $\sigma_{in} = 0.011$ and $\sigma_{out} = 0.006$. This may be due to the lack of neutrons in the system which ranges from roughly 6000-30000 and that the center control rod may not be sufficiently sampled.

4.4 Temperature Feedback

The SLOWPOKE has a well-studied safety feature in that the excess reactivity of the reactor as a function of average core temperature has a maximum. This feature is due to the negative temperature feedback of the reactor. As the fuel temperature increases the cross-sections of the uranium in the fuel begin to broaden. In particular the absorption cross-section of U-238. An initial test with static cases was performed. The temperature of the fuel was increased and the reactivity was measured. At each temperature a set of 100 runs with 3000 neutrons per run was used. As a preliminary test no delayed neutrons were used. The results can be seen in Figure 4.6.

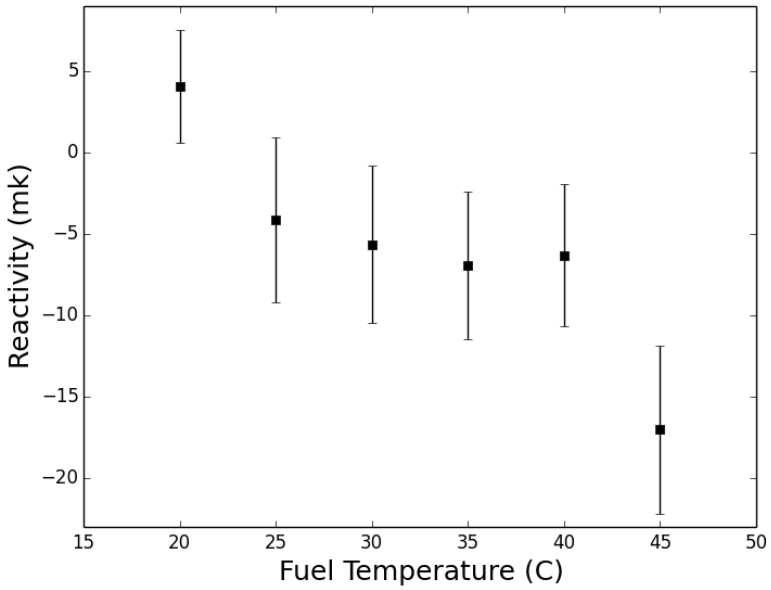

Figure 4.6: Reactivity as a function of fuel temperature.

Additionally a transient simulation was run with the rod plucked out from a rod in source. An initial precursor distribution was used to simulate the delayed neutrons. Thus the simulation is not of a start up but of a de-powered reactor being re-powered. A fission-to-power coefficient (see Equation 3.12) of 30J/fission was used in this test. This was necessary for two reasons: the first being that our simulation cannot reach near the number of neutrons that a real SLOWPOKE would have, secondly there is a time constraint in that our simulations may only reach several fractions of a second and this may not be enough to allow sufficient changes to the fuel geometry. In accordance with Equation 6, the heat transfer coefficient was set to 2100 $\mathrm{W/m^2 \cdot K}$. It should be addressed that our result should be taken as a basis to justify the ability of G4-STORK to model feedback as a direct consequence of time-dependent geometry evolution.

Figures 4.7 and 4.8 show the reactivity as the temperature increases and the neutron population response as a result.

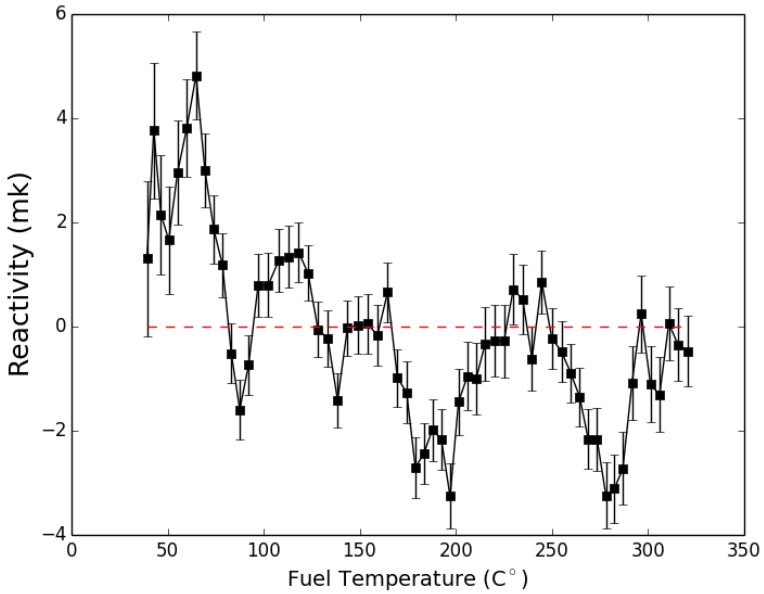

Figure 4.7: The reactivity response to increasing temperature.

The neutron population shows a decreasing trend after the initial rise in population Figure 4.8. This implies that a negative temperature feedback is affecting the neutrons.

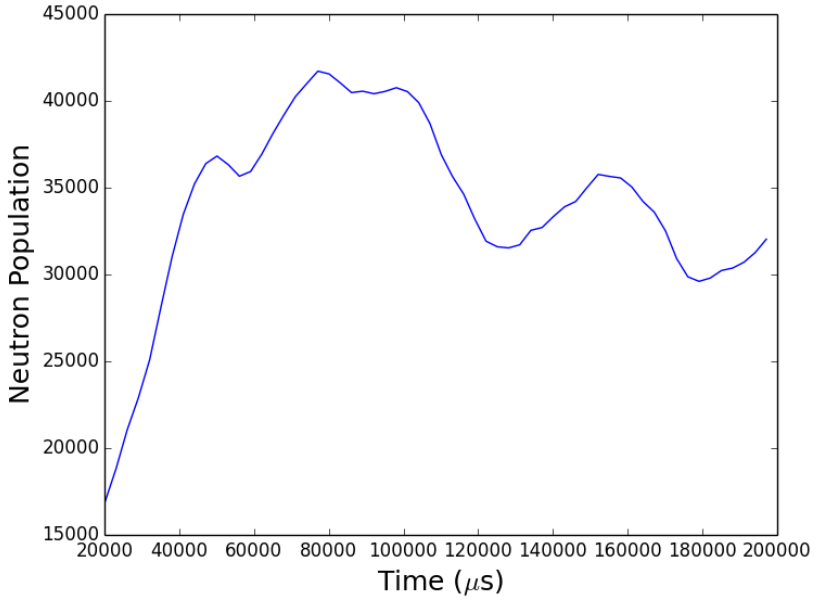

Figure 4.8: Neutron population during transient pluck out with temperature feedback.

Chapter 5

Discussion

5.1 Static Calculations with G4-STORK

The rod worth of the SLOWPOKE-2 at RMC was experimentally measured to be 5.45 mk. The same reactor geometry was modelled in MCNP-4A and it was found that the control rod was worth 7.85 mk from rod in to rod out [11]. Replicating this same reactor G4-STORK found a control rod worth of (4.9 ± 2.0) mk. The differences between the results of G4-STORK and MCNP may be attributed to the methods used to calculate the reactivity. G4-STORK uses the time-dependent dynamic k-effective method whereas MCNP uses a time-independent method which relies on the changes from one generation to the next. This result shows that the methods employed in G4-STORK can model reactivity changes accurately.

The discrepancy between the experiment and simulated worth can be attributed to several factors. The data from open literature used to model the SLOWPOKE had to make several small approximations based on the available schematics and drawings. Therefore there are differences between the simulated reactor geometry and the real reactor geometry, an example being the estimated cadmium sheath lining in the control rod which may be ±1 cm from the simulated location. Another example are the exact compositions of the material and the impurities that are present in the reactor. Finally there is of course the statistical uncertainty in that the control rod region may not be sufficiently sampled however this should also be evident in Figures 4.1 and 4.2 thus it may be ruled out.

The SLOWPOKE reactor has been measured to provide a relatively high neutron

flux on the order of 10^{12} cm^{-2} s^{-1}. From Figures 4.1 and 4.2, the simulated fluxes are on the same order of magnitude. The amplitude of the flux is reduced when the control rod is fully inserted with the flux shape remaining relatively intact. The densities of neutron and fission sites in Appendix A also imply a conserved shape with control rod movement. This means that our perturbation mainly affects the overall reactivity of the system and supports the use of the SLOWPOKE reactor as a favourable reactor for simulation. Additionally the fuel rods of the SLOWPOKE extend out to a radius of roughly 10 cm and this is demonstrated in the flux of the thermal energy region as it increases sharply after 10 cm before dropping off in the outer regions of the reactor.

Figure 1.4 shows the experimentally measured excess reactivity as a function of the core temperature and we can compare this to the result in Figure 4.6. Our result implies a negative temperature feedback of the reactor similar to the experiment. However, there are some discrepancies in that the reactivities measured in G4-STORK cover a larger range than in experiment and that the shape is not depicted. This may be due to the fact that since we are only modelling the effect of the fuel in relation to the feedback we are ignoring other effects such as the temperature and density changes in the coolant, reflector and other materials in the reactor. Since the SLOWPOKE is already under-moderated decreasing density of the coolant/moderator running through the core would result in a negative reactivity effect, but in the temperature ranges of 15-50 °C the density of water does not change within 1%. Assuming that the neutrons in the core of the reactor were in thermal equilibrium, their energies would shift by 10%. The positive excess reactivity seen in the experiment may be due to the shift in thermalization of the neutrons. Once a critical temperature has been reached the negative reactivities of the decreased density and fuel temperatures would take over.

5.2 Transient simulations in G4-STORK

The precursor-delayed neutron method developed for our simulations provides a most-accurate and robust way of producing delayed neutrons in the reactor. Figures 4.4 and 16 both illustrate the time-dependency of the precursors as they are directly affected by the current reactor state. One caveat to using this method however is the starting distribution of the groups as they are a representation of the past. In G4-STORK we achieve an initial distribution by integrating over a previous simulation. Using a previous simulation it is possible to solve for a distribution (Equation 3.27).

The rod pluck-out experiment in Figure 1.3 can be compared to our simulated experiment in Figures 4.8 and 14. With the reliable assumption that the neutron population is directly proportional to the reactor power we can see that the shape of our simulated power resembles that of the experiment. The "bumps" or "waves" seen in the simulation are due to statistical fluctuations in the population. Since we are many orders of magnitude less than realistic populations the random behaviour in the neutrons are more evident.

The reactivity trend in the experiment can also be seen in the simulation however due to large statistical fluctuations this is not as clear.

5.3 Assumptions and Simplifications

The new heat transfer and delayed neutron methods have both shown promise in their capabilities to model a time-dependent reactor. There are however some simplifications and assumptions in their implementation.

The following list summarizes the simplifications and assumptions used in the heat transfer model:

- only the fuel is updated in temperature, density and volume,
- heat transfer occurs between only the fuel and the coolant,
- the heat transfer coefficient is constant,
- the fuel has a uniform spatial temperature,
- heat generation is uniform in the fuel.

In terms of importance, modelling the changes in the properties of the coolant is priority. Due to the undermoderation of the SLOWPOKE-2 any changes to the density of the coolant will have a significant effect. During the heating up of the coolant/moderator natural convective flows occur the reactor. Modelling the fluid dynamics of the coolant would also be important since the density of the coolant flowing through the core will be affected by this as well as the heat transfer coefficient of the fuel. Given that the Biot number is sufficiently small (Appendix B) the Lumped Capacitance model should be a sufficient approximation and should not contribute a significant error in accuracy.

The precursor-delayed method also makes use of some simplifications used below:

- only six precursor groups from U-235 in the thermal energy range were used,
- the input simulation to generate the initial precursor distribution is in steady state.

Since most of the fissions occurring in the SLOWPOKE-2 are from U-235 it was an acceptable assumption to ignore groups from U-238 similarly since most of the fissions occurring are due to thermal neutrons other energies were ignored. The coding of the model was done so that additional groups can be implemented quite easily. When generating a steady state precursor distribution we must assume that the input simulation is in precursor balance itself. Given that the input simulation has a $k_{eff} \approx 1$ this approximation should hold.

Chapter 6

Conclusions

By using our GEANT4 derived code to model the SLOWPOKE-2 reactor we aim to prove the feasibility of using GEANT4 as a platform for the modelling of the next generation designs and reactor accident/transient scenarios. The SLOWPOKE-2 was chosen as a starting point due to its small compact form, single centred control rod and behaviour that is tightly coupled to its neutronics.

Static simulations were performed that matched the SLOWPOKE's control rod worth of 5.45 mk compared to the simulated value of (4.9 ± 2.0) mk. The simulated neutron flux also matched the experimentally measured flux of 10^{12} cm^{-2} s^{-1}. The temperature feedback was also simulated using static cases at increasing temperatures between 15-45 °C. The simulations showed a negative temperature feedback which agrees with the reactor's true behaviour.

To replicate the transient behaviour of the SLOWPOKE a precursor-delayed method was developed to simulate the time-dependent decay of precursors into delayed neutrons. A lumped-parameter model was also incorporated in order to provide time-dependent fuel temperatures that used direct input from the current fission rate. Simulating a rod pluck out with these models we found that the initially positive reactivity drops and becomes negative as the fuel temperature increases. Responding to this the neutron population (reactor power) reaches a maximum and shows the self-regulating ability of the SLOWPOKE.

Several simplifications were made in the heat transfer of the SLOWPOKE-2. Namely the temperature and density of the coolant was not modelled. This is likely to significantly contribute to the discrepencies seen when comparing the static feedback

cases of G4-STORK and experiment.

There are some future developments in the code that can greatly improve its ability. Currently G4-STORK runs at only a fraction of the speed of other Monte Carlo codes thus a speed-up is needed. With this speed-up a longer duration of time can be simulated as well as a larger number of neutrons can be used.

Additional models such as a dedicated coupling to a full-core thermal-hydraulics code can provide a more accurate simulation. It is recommended that once this and the code speed-up are achieved the SLOWPOKE model be revisited.

The addition of photons in the reactor can also be achieved with the code as the physics models are already included in the GEANT4 toolkit. This is an added level of accuracy that is not seen in other codes.

Overall the simulations of the SLOWPOKE show that the GEANT4 derived G4-STORK code can be a valuable tool for transient simulations. With further development time and additional models we can expect G4-STORK to achieve a high-level of accuracy and capability for various applications in reactor modelling.

Bibliography

[1] Jos Thijssen, *Computational Physics, 2nd edition*, New York: Cambridge University Press, (2007). Print.

[2] L. Russell, 2012. *Simulation of Time-Dependent Neutron Populations for Reactor Physics Applications using the Geant4 Monte Carlo Toolkit.* (MS Thesis). [Hamilton, ON]: McMaster University.

[3] "TOP-C (Task Oriented Parallel C/C++)". Boston, MA. Cooperman, Gene, 2014. Web. 04 Nov 2015.

[4] L. Russell, A. Buijs, and G. Jonkman, April 2013, *G4-STORK: A Monte Carlo Reactor Kinetics Simulation Code*, Nuclear Science and Engineering 176 (2014), 370–375.

[5] S. Mahzooni, 2015, *Simulation of Subcritical Experiments in ZED-2 using G4-STORK.* (MS Thesis), [Hamilton, ON]: McMaster University.

[6] A. Otto and A.Buijs, 2015, *Simulation of the McMaster Nuclear Reactor Using the G4STORK Code.* (Report), [Hamilton, ON]: McMaster University.

[7] "About GEANT4", *GEANT4 Consortium.* 2006. Web. 05 Nov 2015.

[8] "Research Reactors", *Canadian Nuclear Association.* Web. 07 Nov 2015.

[9] D. Razon, S. Kaveh, August 1997, *SLOWKIN: A Simplified Model for the Simulation of Transients in the SLOWPOKE-2 Reactor.* (Report), IGE-219 (rev.l), Montreal, QC.

[10] R.E. Kay, J.W. Hilborn, N.B. Poulsen, January 1976, *The Self-Limiting Power Excursion Behaviour of the SLOWPOKE Reactor, Results of Experiments and Qualitative Explanation.* (Report), AECL-4770.

[11] J.R.M. Pierre, 1996, *Low Enrichment Uranium (LEU)- Fuelled SLOWPOKE-2 Nuclear Reactor Simulation with the Monte Carlo based MCNP-4A.* (MS Thesis). [Kingston, ON]: Royal Military College.

[12] Bart Sjenitzer, 2013, *The Dynamic Monte Carlo for Transient Analysis of Nuclear Reactors.* Diss. [Delft, Netherlands]: Delft University.

[13] M. Mahjoub and J. Koclas, October 18-21, 2015, *OpenMC - TD, A New Module for Monte Carlo Time Dependent Simulations Used to Simulate a CANDU6 Cell LOCA Accident*, 7th International Conference on Modelling and Simulation in Nuclear Science and Engineering (7ICMSNSE), Ottawa ON, Canada.

[14] "MCNP6 Users Manual v1.0", *Los Alamos National Laboratory*, (2013).

[15] J. Duderstadt, L. Hamilton, 1976, *Nuclear Reactor Physics*, USA: Wiley. (1976). Print.

[16] R.W. Bauer, J.D. Anderson, S.M. Grimes, and V.A. Madsen. 1997, *Application of simple Ramsauer model to neutron total cross sections.* (Report), [Livermore, CA]: Lawrence Livermore National Lab.

[17] W.M. Stacey, *Nuclear Reactor Physics, 2nd edition.* Weinhelm, Germany: Wiley-vch. (2007). Print.

[18] "Point Reactor Kinetics", M. Ragheb, (2014).

[19] F.P. Incropera, D.P. DeWitt, T.L. Bergman, A.S. Lavine, *Fundamentals of Mass and Heat Transfer, 6th edition.* NY: Wiley. (2006). Print.

[20] M. Kazimi and M. Massoud, 1980, *A Condensed Review of Nuclear Reactor Thermal-Hydraulic Computer Codes for Two-Phase Flow Analysis* (Report), Energy Laboratory Report No. MIT-EL 79-018.

[21] "GEANT4 Users Manual - Chapter 5", *Geant4 Collaboration.* 2015. Web 07 Nov 2015.

[22] L. Russell, A. Buijs, and G. Jonkman, May 2014, *G4-STORK: A Geant4-based Monte Carlo reactor kinetics simulation code*, Annals of Nuclear Energy 71 (2014), 451–461.

[23] "Entropy", *Encyclopedia of Mathematics*. R.L. Dobrushin V.V. Prelov; 2011. Web 01 Nov 2015.

[24] "Fuelpin Engineer's Manual", F.D. Rance, (1982).

[25] S.G. Popov, J.J. Carbajo, V.K. Ivanov, G.L. Yoder, 2000, *Thermal Physical Properties of MOX and UO_2 Fuels Including the Effects of Radiation*. (Report) [Oak Ridge, TN]: Oak Ridge National Laboratory.

[26] M. M. Bretscher, 1997 *Perturbation-Independent Methods for Calculating Research Reactor Kinetic Parameters*. (Report), [Lemont, IL]: Argonne National Laboratory.

[27] G. Gauthier and A. Buijs, 2014, *Final Report G4-STORK Heat Transfer/SLOWPOKE*. (Report), [Hamilton, ON]: McMaster University.

[28] "Convective Heat Transfer", *Engineer's Toolbox* . Web 12 Nov 2015.

[29] "Physics Reference Manual, version 4.9.4", *GEANT4 Consortium*, 2010. Web. 09 Nov 2015.

[30] J.R. Lamarsh and A.J. Baratta, *An Introduction to Nuclear Engineering, 3rd Edition*. Prentice-Hall, (2001). Print.

[31] A. Tan and A. Buijs, 2015, *Monte Carlo Simulations of the SLOWPOKE-2 Reactor*. 39th Annual CNS/CNA Student Conference, St. John, NB, Canada.

[32] "ENDF/B-VII.1.", *Brookhaven National Laboratory*, 2015. Web, 04 Sept 2015.

[33] D. E. Cullen, C. J. Clouse, R. Procassini, and R. C. Little, 2003, *Static and Dynamic Criticality: Are They Different?*. (Report), [Livermore, CA]: Lawrence Livermore National Lab.

[34] B. Sjenitzer, and J.E. Hoogenboom, May 8-12, 2011, *Implementation of the Dynamic Monte Carlo Method for Transient Analysis in the General Purpose Code Tripoli*, International Conference on Mathematics and Computational Methods Applied to Nuclear Science and Engineering (M&C 2011) Rio de Janeiro, RJ, Brazil.

[35] A. Buijs, W. Ford and A. Tan, October 18-21, 2015 *Static Benchmarks for G4-STORK, a Time-Evolution Monte Carlo Code for the Simulation of Multiplying Media*, 7th International Conference on Modelling and Simulation in Nuclear Science and Engineering (7ICMSNSE), Ottawa, ON, Canada.

Appendix A

G4-STORK Static Analysis Plots

The following plots illustrate the SLOWPOKE's response from a inserted rod to a retracted rod. The retracted rod state represents 20 ms afterwards. It is useful to note that when plucked out the survivor densities maintain shape and only increase in intensity.

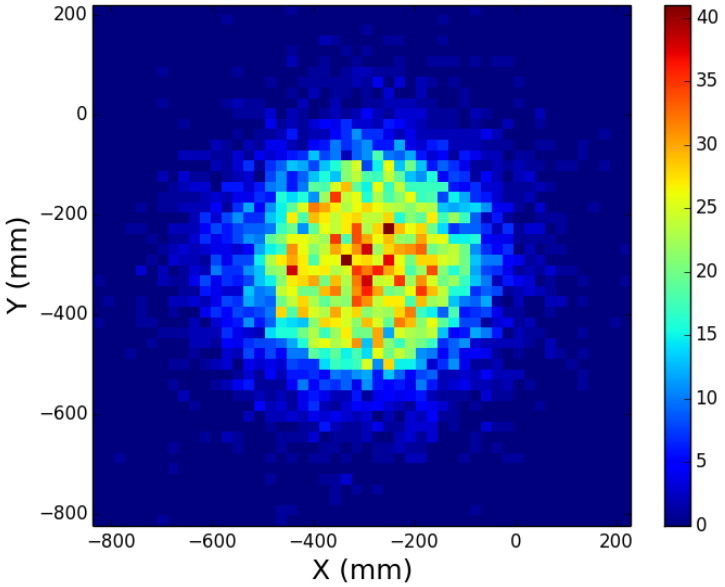

Figure 1: 2D Histogram (x-y plane) of the neutron survivors (Rod inserted).

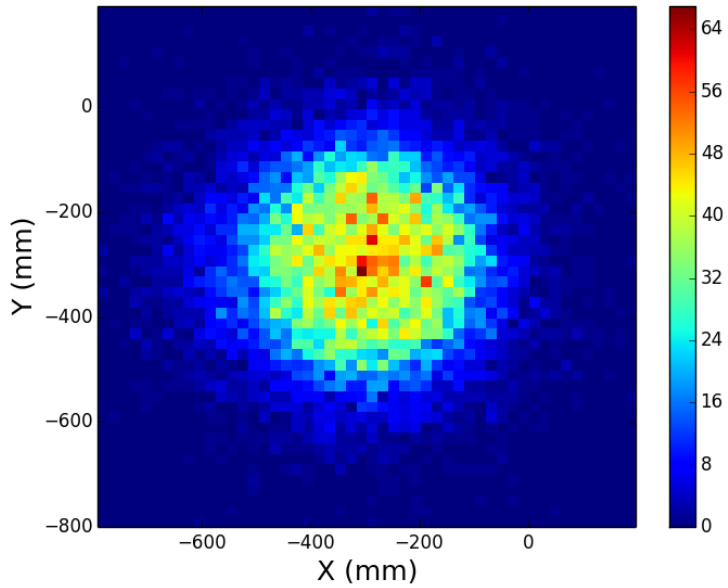

Figure 2: 2D Histogram (x-y plane) of the neutron survivors (Rod retracted).

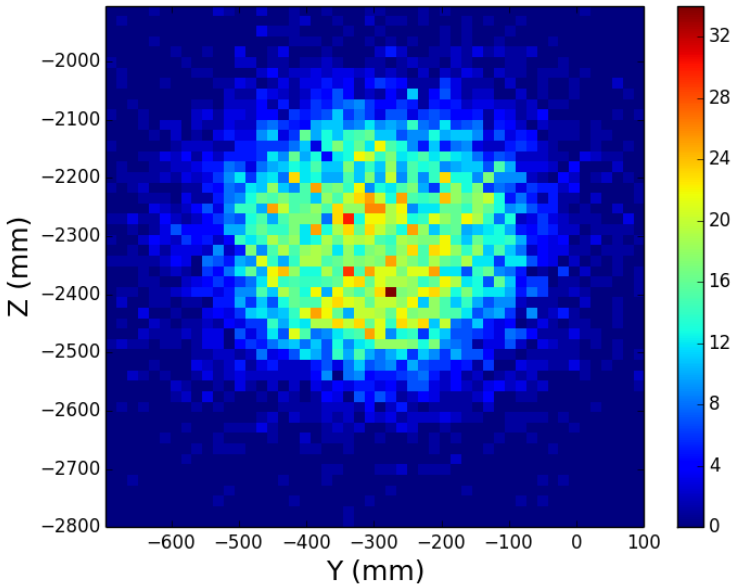

Figure 3: 2D Histogram (y-z plane) of the neutron survivors (Rod inserted).

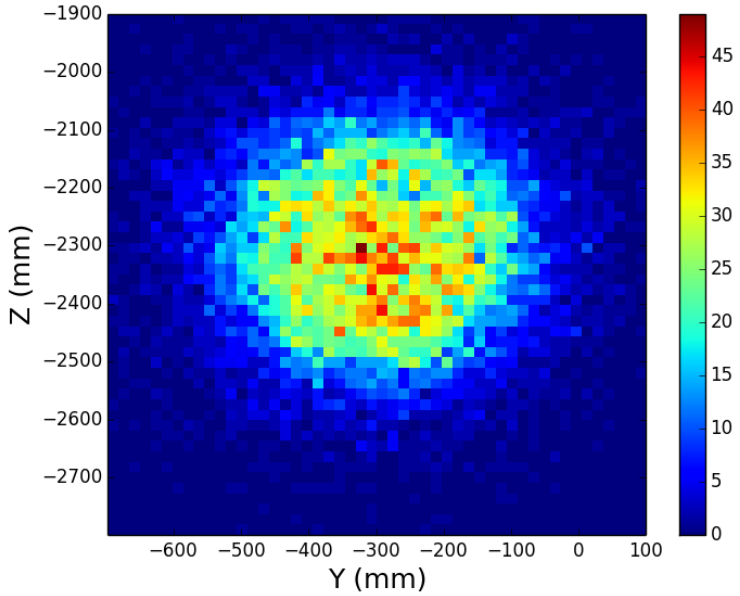

Figure 4: 2D Histogram (y-z plane) of the neutron survivors (Rod retracted).

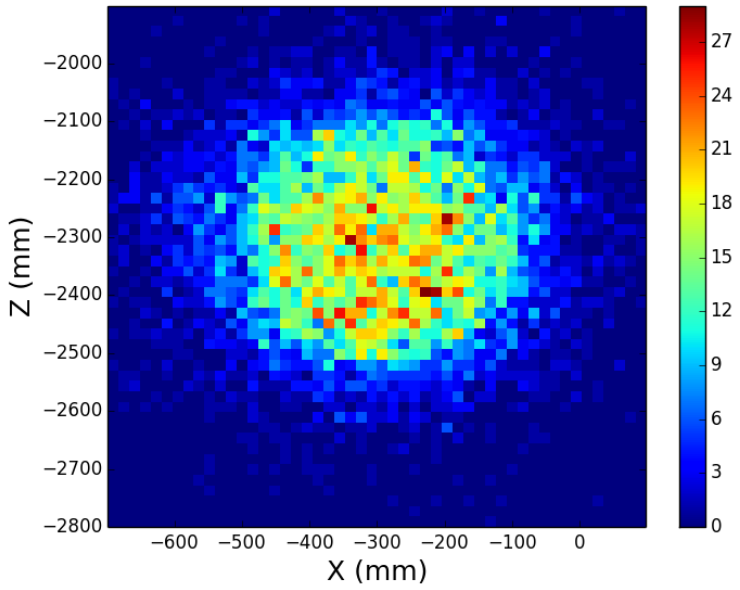

Figure 5: 2D Histogram (x-z plane) of the neutron survivors (Rod inserted).

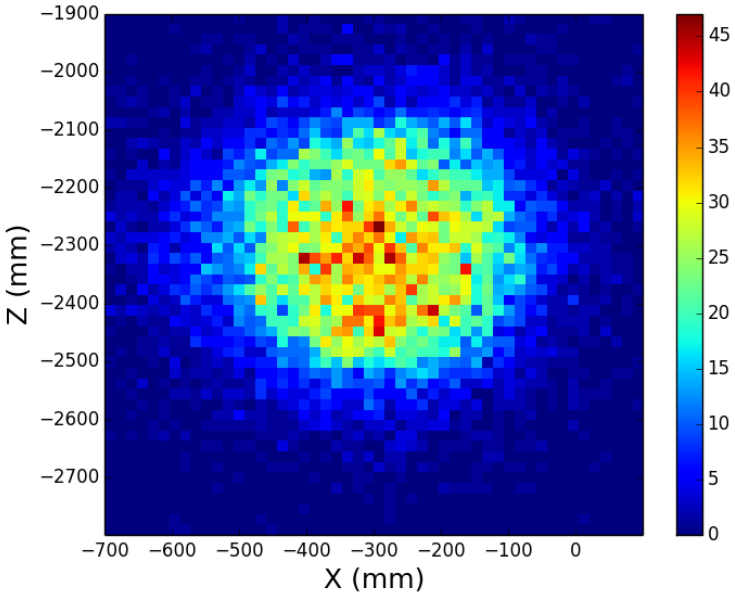

Figure 6: 2D Histogram (x-z plane) of the neutron survivors (Rod retracted).

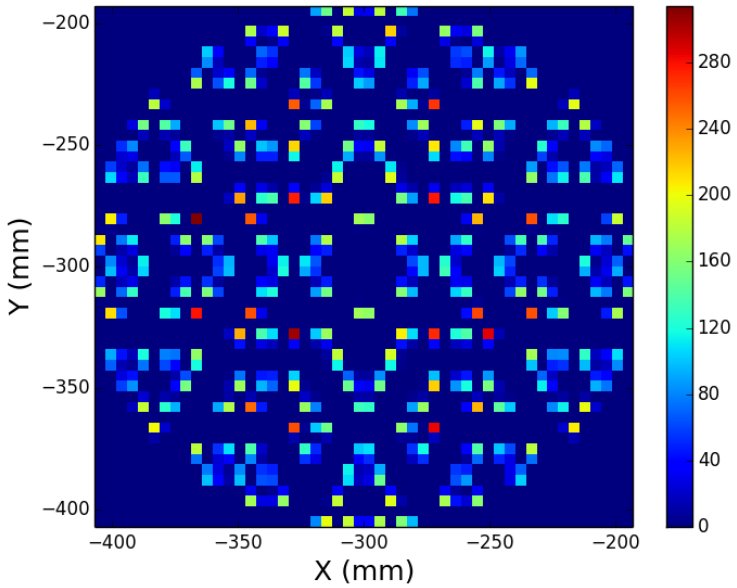

Figure 7: 2D Histogram (x-y plane) of the fission sites (Rod inserted).

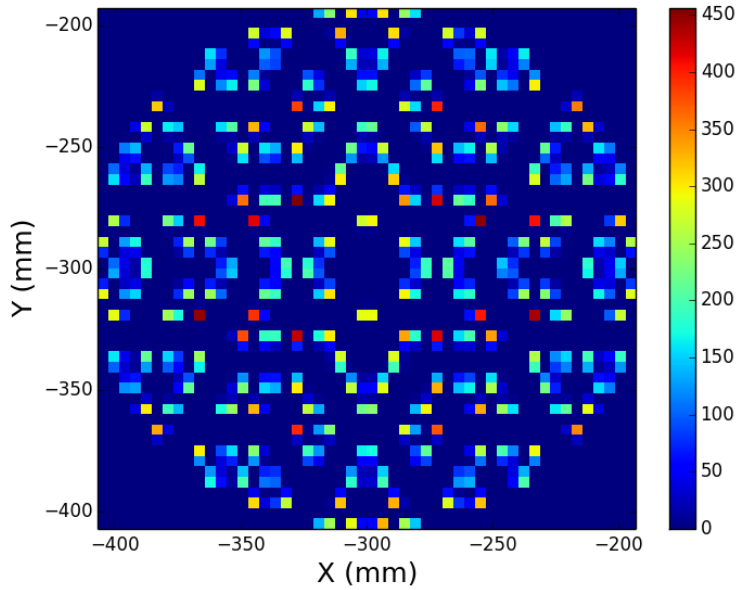

Figure 8: 2D Histogram (x-y plane) of the fission sites (Rod retracted).

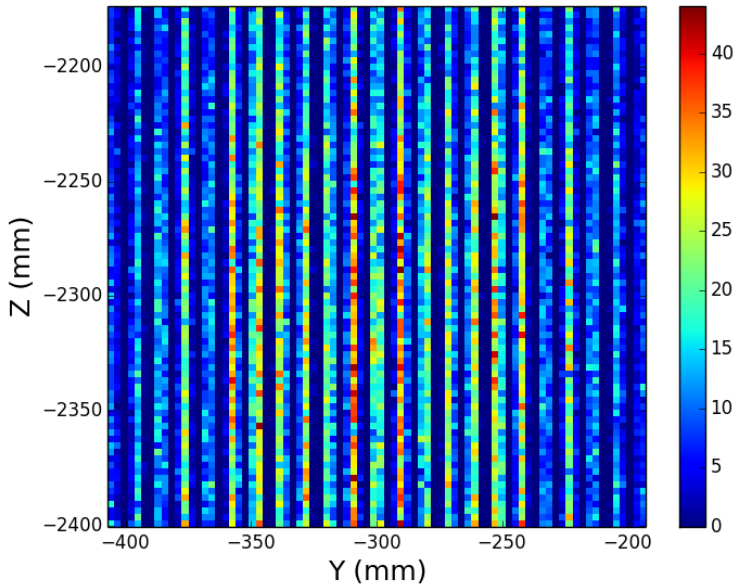

Figure 9: 2D Histogram (y-z plane) of the fission sites (Rod inserted).

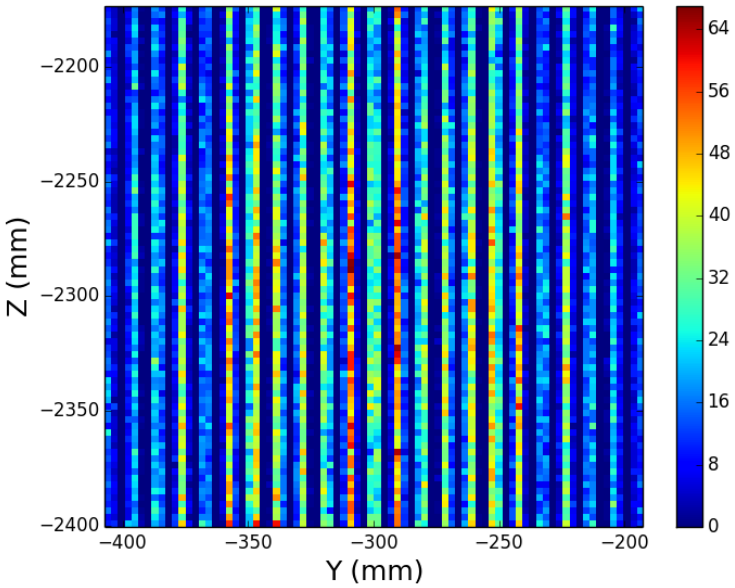

Figure 10: 2D Histogram (y-z plane) of the fission sites (Rod retracted).

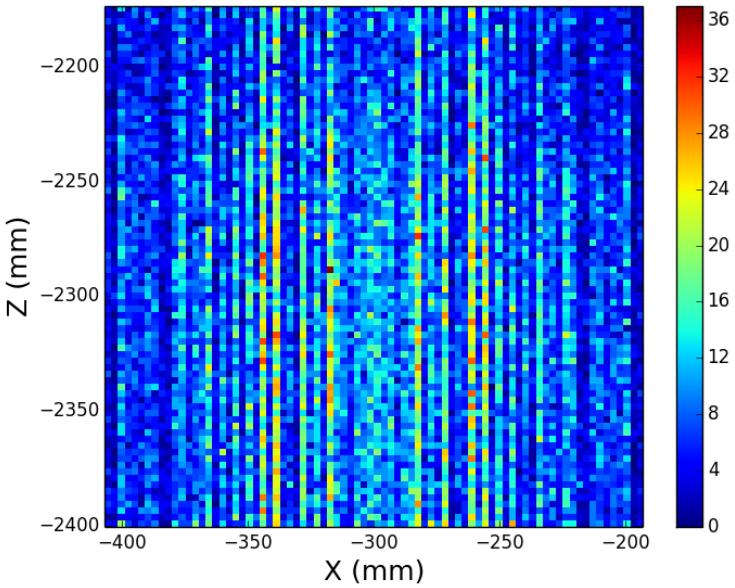

Figure 11: 2D Histogram (x-z plane) of the fission sites (Rod inserted).

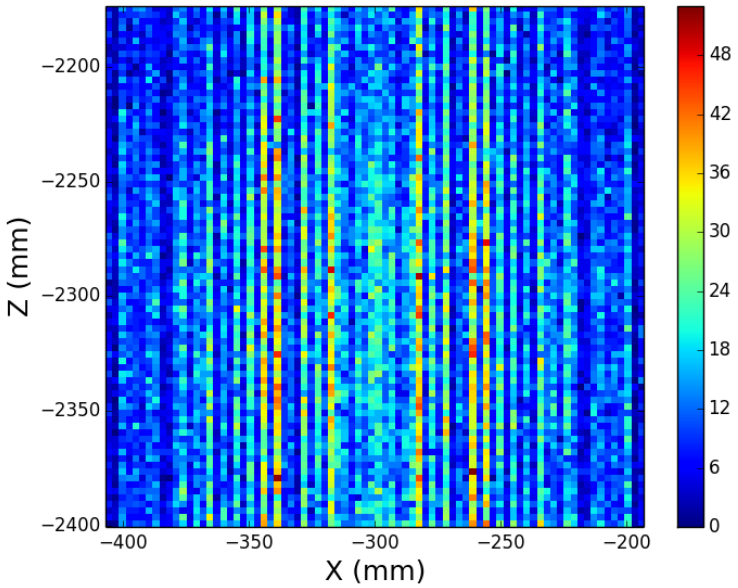

Figure 12: 2D Histogram (x-z plane) of the fission sites (Rod retracted).

Transient Plots

These plots are time-dependent plots initiated from a rod pluck with and without temperature feedback effects.

Temperature Feedback

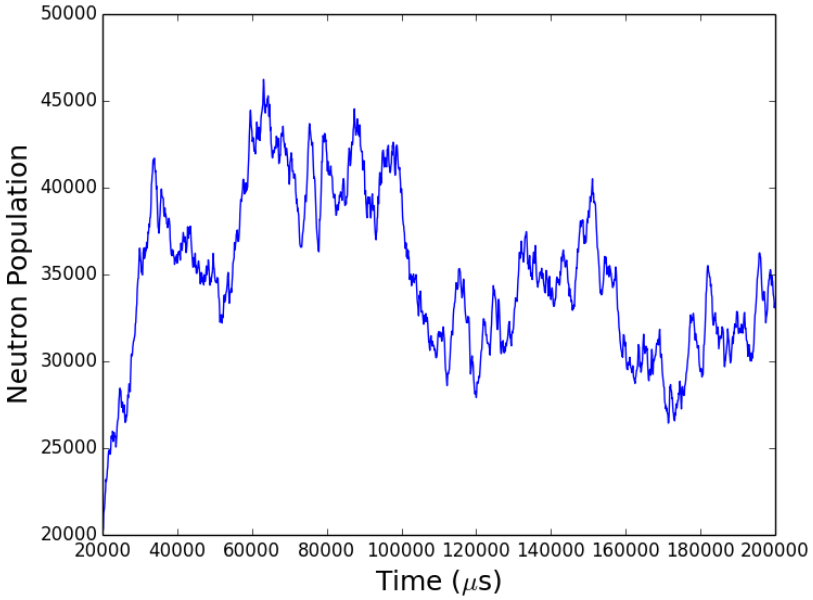

Figure 13: Neutron population during transient pluck out with temperature feedback (no time average).

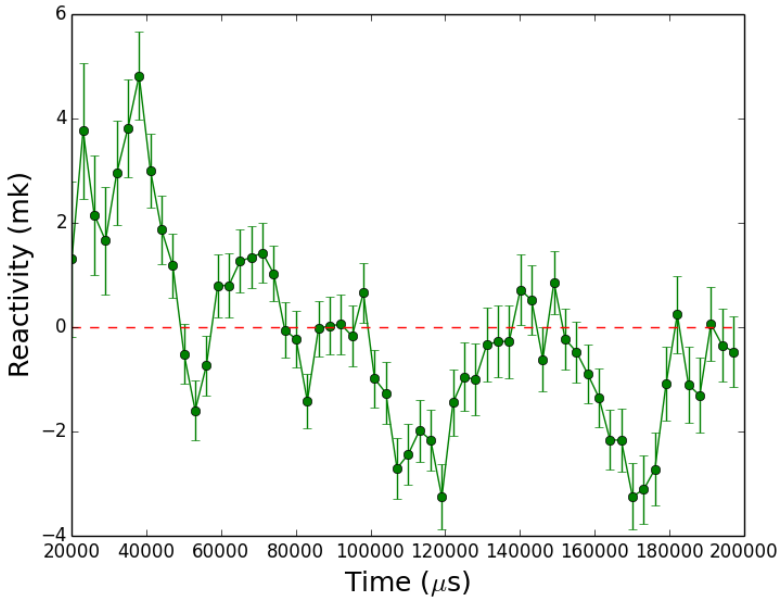

Figure 14: Reactivity vs time from rod pluck out.

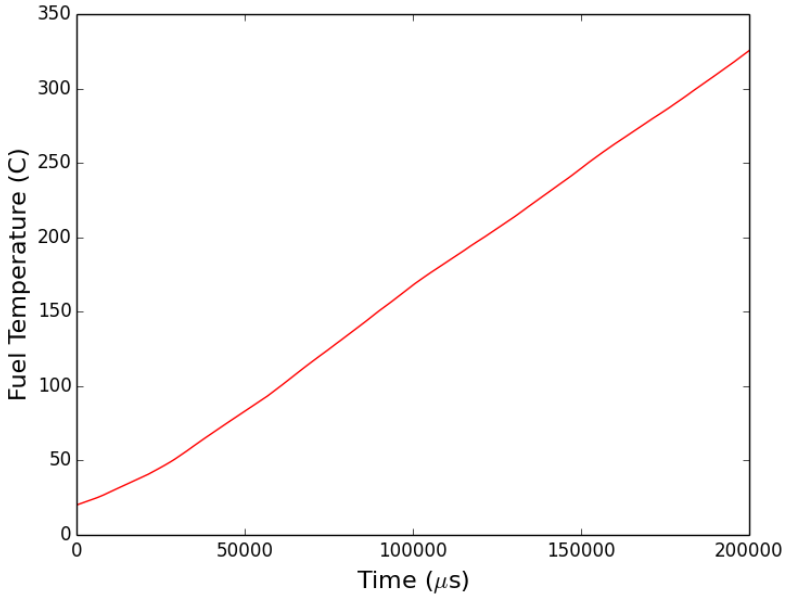

Figure 15: Temperature vs time from rod pluck out.

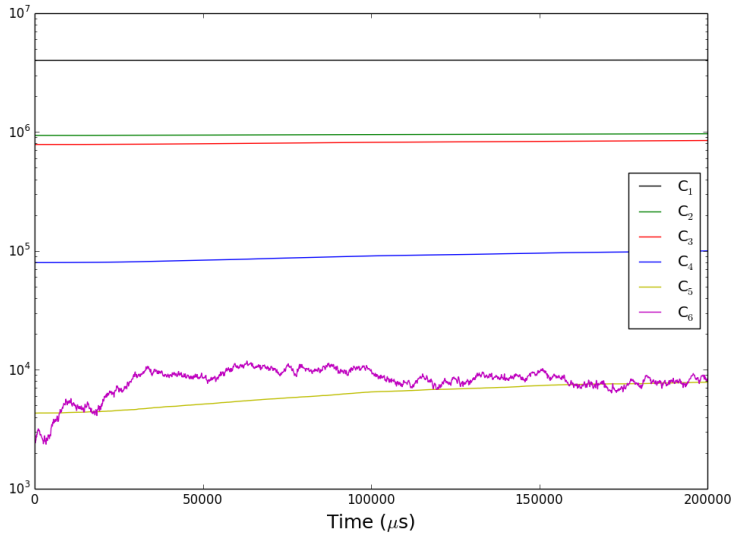

Figure 16: Populations of the six precursor groups during rod pluck out.

No Temperature Feedback

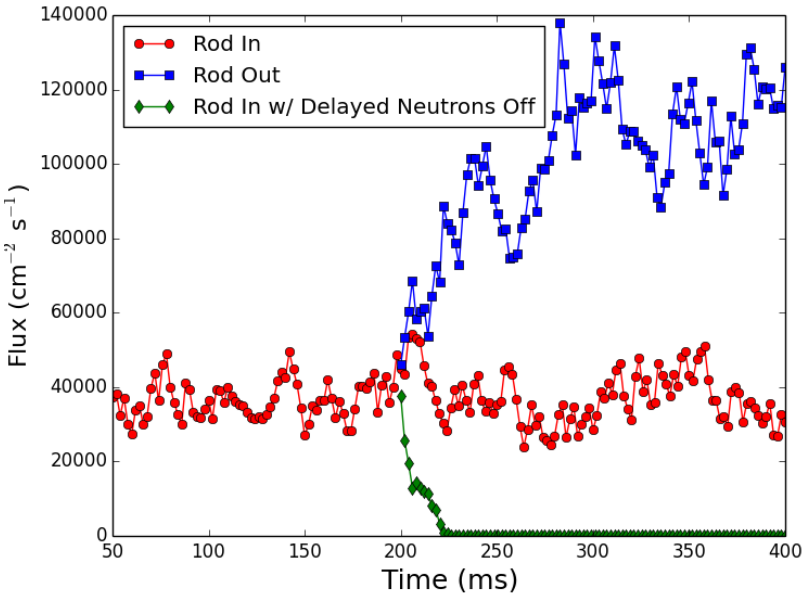

Figure 17: Transient of rod pluck out (no time average).

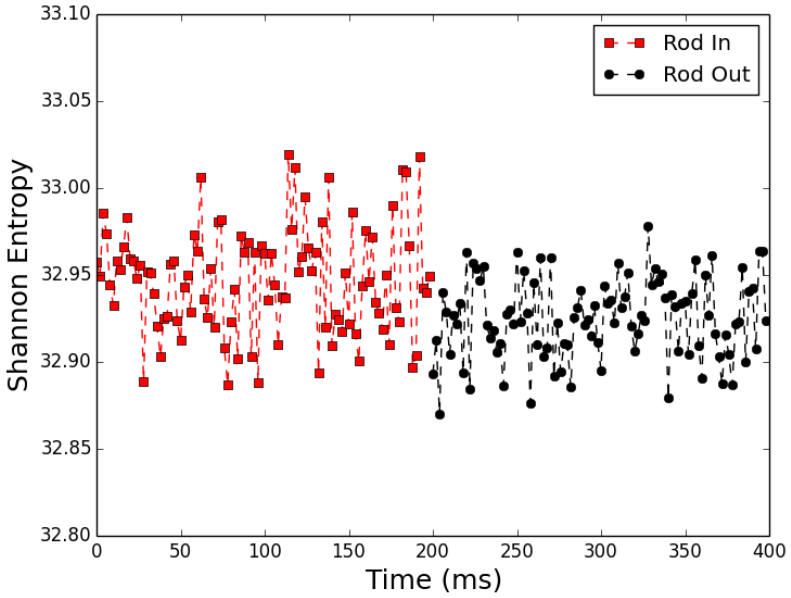

Figure 18: Fission site entropy.

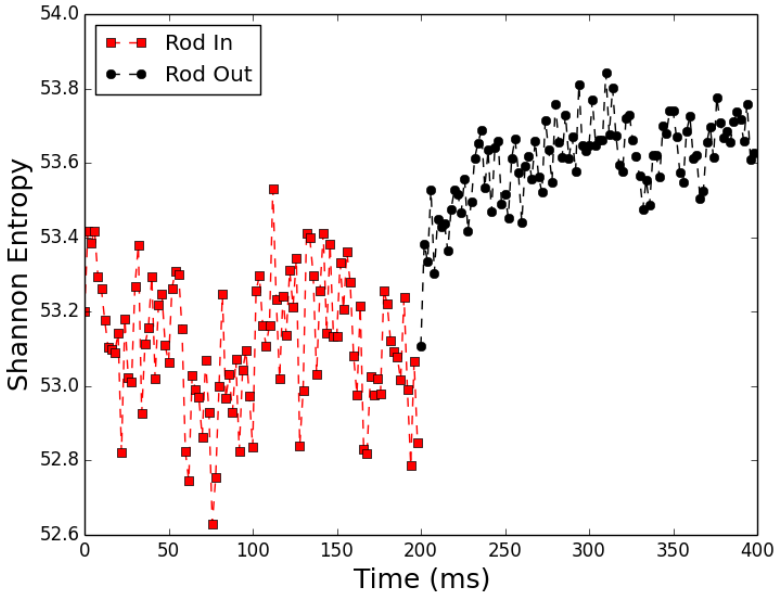

Figure 19: Survivor site entropy.

Appendix B

Lumped Capacitance of the SLOWPOKE-2 Fuel Pin Assembly

The method of Lumped Capacitance was used in modelling the heat transfer between the fuel and its surroundings. When dealing with transient heat transfer it is suggested that the Biot number is first calculated in order to characterize the problem. The Biot number is a dimensionless parameter giving the ratio between the heat transfer resistances inside and at the surface of a body. It is defined as:

$$Bi = \frac{hL_c}{k} \tag{1}$$

where h is the heat transfer coefficient,
L_c is the characteristic length,
and k is the thermal conductivity.
In order the use the Lumped Capacitance model we have the condition that the Biot number must be sufficiently small [19, p. 256-261]:

$$Bi < 0.1 \tag{2}$$

Thus for our transient conduction we must satisfy this condition by placing limitations of the heat transfer coefficient between our fuel and its surrounding coolant.
The characteristic length is commonly chosen as:

$$L_c = \frac{V_{body}}{A_{surface}} \tag{3}$$

with this and the thermal conductivity in place we can vary the Biot number, and thus the heat transfer coefficient to satisfy the sufficiently small criteria.
Using the dimensions:

$$L_c = 0.131cm \tag{4}$$

thus our restriction for the heat transfer coefficient:

$$h < 0.1 \times 27.6 \text{ W/m·K} \times \frac{1}{0.131 \times 10^{-2} \text{ m}} \tag{5}$$

$$h < 2107 \text{ W/m}^2\text{· K} \tag{6}$$

Justification of our usage of this model is also supported by literature. Heat transfer coefficients for convective heat transfer in water are typically between 50 - 3000 W/m·K [28], which our restricted range is valid. Also the time constant for the heat transfer from the fuel to the outside surface of the sheathe is estimated to be 10 ms [10]. Therefore there is very little variation in the radial temperature profile of the fuel assembly.

Appendix C

Contributed Code in G4-STORK

The following codes are the StorkHeatTransfer and StorkDelayedNeutron classes used in G4-STORK.

StorkHeatTransfer Class

```
//StorkHeatTransfer.cc - Class designed to update the fuel properties based on
//fission rate.

#include "StorkHeatTransfer.hh"


// Constructor
StorkHeatTransfer::StorkHeatTransfer(const StorkParseInput* fIn)
{

    worldname = fIn->GetWorld();

    if(worldname == "SLOWPOKE")
        InitializeSlowpokeModel(fIn);

}
// Destructor
StorkHeatTransfer::~StorkHeatTransfer(void)
{
}

//Initialize SLOWPOKE model.
void StorkHeatTransfer::InitializeSlowpokeModel(const StorkParseInput* input)
{
    dt = input->GetRunDuration();
    heatTransferCoeff = input->GetHeatTransferCoefficient();
    temp_infty = input->GetAmbientTemperature();
    saveMaterialData = input->SaveTemperature();
    saveMaterialfilename = input->GetTemperatureDataFile();
    fissionToEnergyCoeff = input->GetFissionToEnergyCoefficient();
    baselineFissionRate = input->GetBaselineFissionRate();
    createHeader = true;
    return;
}

void StorkHeatTransfer::RunThermalCalculation(MSHSiteVector fissionSites)
{
    fSites = &(fissionSites);

    if(worldname == "SLOWPOKE"){
        RunSlowpokeModel();
    }

    return;
}

void StorkHeatTransfer::RunSlowpokeModel()
{

    //Set fuel dimensions
    SetFuelDimensions(theWorld->GetFuelDimensions());

    //Calculate the heat distributions
    CalcHeatDistribution();

    //Update the fuel properties.
    UpdateFuelProperties(theWorld->GetFuelTemperatures(),theWorld->GetFuelDensities(),theWorld->GetFuelRadii();

    if(saveMaterialData)
        SaveSLOWPOKEFuelProperties(saveMaterialfilename);
```

```
    theWorld->SetPhysChanged(true);
    theWorld->SetMatChanged(true);

    return;
}
//Update the fuel properties
void StorkHeatTransfer::UpdateFuelProperties(G4double* FuelTemperatures,G4double* FuelDensities,
G4double* FuelRadii)
{

    //Initialize variables.
    StorkMaterialHT *material;
    G4double newTemperature;
    G4double newDensity;
    G4double newRadius;
    G4double oldDensity;
    G4String fuelNum;
    const char * num;
    G4double heatCapacity;
    G4double mass;
    G4double surfaceArea;
    G4double HeatInMaterial;
    G4int size = fnDistribution.size();
    fuelTempAvg = fuelDensityAvg = fuelRadiusAvg = 0.0;

    G4double TotalRunFissions = fSites->size();

    G4double heatGenerated = CalcEffectiveHeatGenerated(TotalRunFissions);

    //Get material map.
    StorkMaterialMap *matMap = theWorld->GetMaterialMap();

    //Iterate through fission material list.
    std::map< G4String, G4double >::iterator it;
    for(it = fnDistribution.begin(); it != fnDistribution.end(); it++){

        //Get the fuel name and number ID;
        fuelNum = (it->first);

        num = (fuelNum.erase(0,4)).c_str();

        G4int i = std::atoi(num);

        //Calculate the heat generated in each material.
        HeatInMaterial = heatGenerated*(it->second);

        //Get material.
        material = static_cast<StorkMaterialHT*>((*matMap)[it->first]);

        //Fuel dimensions in mm convert to cm.
        G4double radius = FuelRadii[i]*pow(10,-1);
        G4double length = fuelDimensions[2]*pow(10,-1);

        //Get material and geometric properties.
        heatCapacity = material->GetSpecificHeatCapacity()*pow(10,9);
        oldDensity = material->GetDensity();
        mass = oldDensity*(cm3/g)*CLHEP::pi*length*radius*radius;
        surfaceArea = 2*CLHEP::pi*length*radius;

        //Calculate new material properties.
        newTemperature = CalcFuelTemperature(HeatInMaterial, mass, surfaceArea, FuelTemperatures[i],
        heatCapacity);
        newDensity = CalcFuelDensity(newTemperature);
        newRadius = CalcFuelRadius(FuelRadii[i], oldDensity, newDensity);
```

```
        //Set the new fuel properties.
        FuelTemperatures[i] = newTemperature;
        FuelDensities[i] = newDensity;
        FuelRadii[i] = newRadius;
        //Calculate averages.
        if(saveMaterialData){
            fuelTempAvg += G4double(newTemperature/size);
            fuelDensityAvg += G4double(newDensity/size)*cm3/g;
            fuelRadiusAvg += G4double(newRadius/size);
        }

    }


    return;
}

void StorkHeatTransfer::CalcHeatDistribution(){


    //Get the total number of fissions.
    G4int numberOfFissions = fSites->size();

    //Percentage per fission
    G4double fPercent = (1./numberOfFissions);

    //Create material list flag.
    //G4bool createList = false;

    //Clear the distribution map.
    fnDistribution.clear();

    //Get pointer to the Navigator
    G4Navigator* theNavigator = G4TransportationManager::GetTransportationManager()->GetNavigatorForTracking();
    theNavigator->ResetStackAndState();

    std::vector<StorkTripleFloat>::iterator itr = fSites->begin();

    //Iterate through all fission sites of the run.
    for(; itr!=fSites->end(); itr++){

        G4ThreeVector site = itr->GetData();

        //Get the material.
        G4String currentMaterialName = theNavigator->LocateGlobalPointAndSetup(site)->GetLogicalVolume()->
        GetMaterial()->GetName();

        //Check to see if it is already in the material list, else create it.
        if(fnDistribution[currentMaterialName]){
            fnDistribution[currentMaterialName] += fPercent;
        }
        else
            fnDistribution[currentMaterialName] = fPercent;
    }
    return;

}
```

```
//Calculates new temperature based on heat generation. Uses FUELPIN solver method.
G4double StorkHeatTransfer::CalcFuelTemperature(G4double heatGeneration, G4double mass,
G4double surfaceArea, G4double oldTemp, G4double heatcapacity){

    //Initialize with proper units.
    G4double newTemperature;
    //htc in J/cm2*K
    G4double htc = heatTransferCoeff*pow(10,-4);
    //t left in ns
    G4double t = dt*pow(10,-9);

    //Calculate coefficients for new temperature.
    G4double a = -htc*surfaceArea/(mass*heatcapacity);
    G4double bu = (heatGeneration + htc*surfaceArea*temp_infty)/(mass*heatcapacity);

    //Calculate new temperature.
    newTemperature = oldTemp*exp(-a*t) + (1 - exp(-a*t))*(bu/a);

    return newTemperature*kelvin;
}

// Calculates new density (U2O) based on temperature.
G4double StorkHeatTransfer::CalcFuelDensity(G4double temperature){

    //Initialize variables.
    G4double density, a, b, c, d ,e, t;
    t = temperature;
    a = 6.6160*pow(10,19);
    b = 9.99672*pow(10,-1);
    c = 1.179*pow(10,-5);
    d = -2.429*pow(10,-9);
    e = 1.219*pow(10,-12);

    //Calculate using empirical formula.
    density = a*pow( (b + c*t + d*pow(t,2) + e*pow(t,3)) ,-3);

    return density;
}

//Calculate the fuel radius based on the density and conserving mass.
G4double StorkHeatTransfer::CalcFuelRadius(G4double oldRadius, G4double oldDensity, G4double newDensity)
{
    G4double newRadius = oldRadius*pow( oldDensity/newDensity, 0.5);
    if(newRadius>0.212*cm){
        G4cerr << "WARNING: Fuel is expanding beyond limits!" << G4endl;
        return oldRadius;
    }

    return newRadius;

}

void StorkHeatTransfer::SetWorld(StorkWorld* world)
{
    theWorld = world;
}
```

```
void StorkHeatTransfer::SaveSLOWPOKEFuelProperties(G4String filename)
{
    if(createHeader){
        // Declare and open file stream
        std::ofstream outFile(filename.c_str(),std::ofstream::app);

        // Check that stream is ready for use
        if(!outFile.good())
        {
            G4cerr << G4endl << "ERROR:  Could not write material temperatures to file. " << G4endl
            << "Improper file name: " << filename << G4endl
            << "Continuing program without material temperature data output" << G4endl;

            return;
        }
        outFile.fill(' ');
        outFile << "    *** SLOWPOKE REACTOR - LUMPED PARAMETER THERMAL MODEL ***   " << G4endl;
        outFile << "            VERSION - 1.0 - DATE: 07/14/2015        " << G4endl;

        outFile << std::setw(18) << "  Avg. Fuel Temperature "
                << std::setw(18) << "  Avg. Fuel Density  "
                << std::setw(18) << "  Avg. Fuel Volume  "
                << G4endl
                << "-----------------------------------------------------------------"
                << G4endl;

        outFile.close();
        createHeader = false;
    }
    std::ofstream outFile(filename.c_str(),std::ofstream::app);

    if(!outFile.good()){
        G4cerr << G4endl << "ERROR:  Could not write material temperatures to file. " << G4endl
        << "Improper file name: " << filename << G4endl
        << "Continuing simulation without material temperature data output" << G4endl;
    }
    else{

        outFile << std::setw(18) << fuelTempAvg
                << std::setw(18) << fuelDensityAvg
                << std::setw(18) << fuelRadiusAvg
                << G4endl;

        outFile.close();
    }
    return;
}

G4double StorkHeatTransfer::CalcEffectiveHeatGenerated(G4double currentFissions)
{
    //Get the corresponding baseline fissions.
    G4double baselineFissions = baselineFissionRate*dt;

    //If the current rate is less than the baseline than return no heat generated.
    if(currentFissions<baselineFissions)
        return 0.;

    //Otherwise return the heat; difference in number of fissions times the effective energy/fission
    //and rescale it.
    return fissionToEnergyCoeff*(currentFissions - baselineFissions);
}
```

```
//StorkHeatTransfer.hh


#ifndef STORKHEATTRANSFER_H
#define STORKHEATTRANSFER_H

#include "StorkWorld.hh"
#include "StorkContainers.hh"
#include "StorkMaterialHT.hh"
#include "G4TransportationManager.hh"
#include <math.h>
#include <string.h>
#include "G4ParticleDefinition.hh"
#include "G4ParticleTable.hh"

class StorkHeatTransfer
{
    public:
        // Default constructor
        StorkHeatTransfer(const StorkParseInput* fIn);
        ~StorkHeatTransfer(void);

        void InitializeSlowpokeModel(const StorkParseInput* fIn);
        void RunThermalCalculation(MSHSiteVector fissionSites);
        void RunSlowpokeModel();
        void SetWorld(StorkWorld* world);
        void SetFuelDimensions(G4ThreeVector fDim) {fuelDimensions = fDim;}


    protected:
        void CalcHeatDistribution();
        void UpdateFuelProperties(G4double* FuelTemperatures,G4double* FuelDensities,G4double* FuelRadii);
        G4double CalcFuelTemperature(G4double heatGeneration, G4double mass, G4double surfaceArea,
                                                                G4double oldTemp, G4double heatcapacity);
        G4double CalcFuelDensity(G4double temperature);
        G4double CalcFuelRadius(G4double oldRadius, G4double oldDensity, G4double newDensity);
        void SaveSLOWPOKEFuelProperties(G4String filename);
        G4double CalcEffectiveHeatGenerated(G4double currentFissions);


    protected:
        StorkWorld* theWorld;
        MSHSiteVector* fSites;
        G4double reactorPower;
        G4String worldname;
        G4double dt;
        G4double heatTransferCoeff;
        G4double temp_infty;
        std::map< G4String, G4double > fnDistribution;
        std::vector<G4String> fnMaterialList;
        G4double effectiveFissionEnergy;
        G4ThreeVector fuelDimensions;
        G4bool createHeader;
        G4bool saveMaterialData;
        G4String saveMaterialfilename;
        G4double fuelTempAvg;
        G4double fuelDensityAvg;
        G4double fuelRadiusAvg;
        G4double fissionToEnergyCoeff;
        G4double baselineFissionRate;

};

#endif // STORKHEATTRANSFER_H
```

StorkDelayedNeutron Class

```
/*
StorkDelayedNeutron Class
*/

#include "StorkDelayedNeutron.hh"

StorkDelayedNeutron::StorkDelayedNeutron(G4String dnFilename,G4double runduration, G4int numPrimaries)
{
    //Set the fission file name.
    delayedSourceFile = dnFilename;

    //Get the run Duration
    runDuration = runduration;

    //Get neutron particle object
    G4ParticleTable* pTable = G4ParticleTable::GetParticleTable();
    neutron = pTable->FindParticle("neutron");

    //Setup initial precursor population
    GetInitialPrecursors(numPrimaries);

}

StorkDelayedNeutron::~StorkDelayedNeutron()
{

}

G4bool StorkDelayedNeutron::GetInitialPrecursors(G4int numPrimaries)
{
        // Local variables
        char line[256];
        G4int numRuns;
    G4int numEntries;
        G4double primariesPerRun;
    G4double runTime;
    G4double adjustmentFactor;
    G4bool createPrecursors = true;
    Precursors.resize(6);


        // Load data from file
        std::ifstream dnfile(delayedSourceFile);

        // Check if file opened properly
        if(!dnfile.good())
        {
                G4cerr << "*** WARNING:  Unable to open initial delayed neutron file:"
        << delayedSourceFile << G4endl;
                return false;
        }

        // Skip header lines
        while(dnfile.peek() == '#')
                dnfile.getline(line,256);

    //Read in precursor data if nonzero.
    for(G4int i = 0; i<6; i++){
        dnfile >> Precursors[i];
        if(Precursors[i]!=0) createPrecursors = false;
    }
```

```
    // Resize the fission vectors.
    fSites.resize(numEntries);
    fEnergy.resize(numEntries);


    // Read in delayed neutron distribution parameters
    // Number of entries, runtime, number of runs collected over, primaries per run.
    dnfile >> numEntries >> runTime >> numRuns >> primariesPerRun ;

    // Read in fission data.
    for(G4int i=0; i<numEntries && dnfile.good(); i++)
    {
            dnfile >> fSites[i][0] >> fSites[i][1] >> fSites[i][2] >> fEnergy[i];
    }

    //Create the precursors if needed.
    if(createPrecursors){
        //Total time of previous simulation.
        G4double totalTime = pow(10,-9)*numRuns*runTime;
        //Iterate through fission sites and energies to create precursors.
        // Will need to change this method. Precursor populations should be directly calculated.

        G4int totalPrecursors = G4int(numEntries*TotPrecursorConstants/totalTime);

        for(G4int i=0; i<totalPrecursors; i++){

            // For now assume that U235 is the only fuel, uncomment or edit
            //  if you want to factor in more types of fuel.
            //Can also edit the StorkDelayedData.hh to change delay constants and fission yields.

            //index = fissionIndex(fEnergy[i],fSites[i]);

            G4double r = G4UniformRand()*TotPrecursorConstants;

            G4double temp = 0.0;

            for(G4int j=0;j<6;j++){

                temp += PrecursorConstants[j];

                if( r < temp){
                    Precursors[j]++;
                    break;
                }
            }
        }

    }



    //Adjustment factor for differing number of primaries.
    adjustmentFactor = numPrimaries/G4double(primariesPerRun);
    //Rescale the groups.
    for(G4int i=0; i<6;i++){
        Precursors[i] = G4int(adjustmentFactor*Precursors[i]);
    }

    return true;
}
```

```
void StorkDelayedNeutron::SetFissionSource(MSHSiteVector fissionSites, DblVector fissionEnergies)
{

    fSites.clear();
    fEnergy.clear();

    fSites.insert(fSites.end(),fissionSites.begin(),fissionSites.end());
    fEnergy.insert(fEnergy.end(),fissionEnergies.begin(),fissionEnergies.end());

    return;
}


void StorkDelayedNeutron::AddPrecursors()
{
    G4int entries = fSites.size();
    G4int index;
    G4double totalYield;

    if(false){
        G4double runTime = G4double(runDuration*pow(10,-9));
        for(G4int i = 0; i<6;i++){
            Precursors[i] = G4int(entries*PrecursorConstants[i]/runTime);
        }
        return;
    }

    for(G4int i = 0; i < entries; i++){

        // For now assume that U235 is the only fuel, uncomment or edit
        // if you want to factor in more types of fissionable isotopes.
        //Can also edit the StorkDelayedData.hh to change delay constants and fission yields.

        //index = fissionIndex(fEnergy[i],fSites[i]);

        index = 0;


        //Get the total yield.
        totalYield = TotalYields[index];

        //Roll the dice to determine if a precursor is created.
        if(G4UniformRand() < totalYield){

            //Initialize temporary variable (cumulative yield) to determine which precursor.
            G4double temp = 0.0;
            //Random number for precursor group.
            G4double rand = G4UniformRand()*totalYield;

            //Iterate through the precursor yields.
            for(G4int j = 0; j<6; j++){
                temp = temp + FissionYields[index][j];
                if(rand < temp){
                    Precursors[j]++;
                    break;
                }

            }

        }

    }

    return;
}
```

```
NeutronSources StorkDelayedNeutron::GetDelayedNeutrons(G4double runEnd)
{
    //Initialize variables.

    G4double mass = neutron->GetPDGMass();
    G4ThreeVector theMomDir;
    G4double nMom;
    NeutronSources dNeutrons;
    StorkNeutronData theDelayed;

    for(G4int i=0; i<6; i++){

        for(G4int j = 0; j< Precursors[i] ; j++){

            //Roulette to find the time of decay.
            G4double r = G4UniformRand();
            G4double TimeOfDecay = (-log(r)/DecayConstants[0][i])*pow(10,9);
            //G4cout <<TimeOfDecay << G4endl;

            //Check if within upcoming run if so decay a precursor and create a delayed neutron.
            if (TimeOfDecay<runDuration){


                //Get a random indice for a fission site.
                G4int R_ind = (G4int) std::floor(G4UniformRand()*fSites.size() + 0.5);
                G4ThreeVector site(fSites[R_ind].data);

                //Remove a precursor.
                Precursors[i]--;

                //Gaussian sample for a momentum.
                nMom = G4RandGauss::shoot(sqrt(2*mass*EnergyYields[0][i]),0.05);

                // Create the incident neutron
                theMomDir.setRThetaPhi(nMom, G4UniformRand()*CLHEP::pi,
                                       G4UniformRand()*2.0*CLHEP::pi);
                //Get global time of decay.
                G4double GlobalDecayTime = (runEnd-runDuration)+TimeOfDecay;

                //Roulette for lifetime.
                G4double lifetime =(-log(G4UniformRand())/DecayConstants[0][i])*pow(10,9);

                // Set delayed neutron data.
                theDelayed = StorkNeutronData(GlobalDecayTime ,lifetime ,site,theMomDir);

                // Add to delayed neutron list
                dNeutrons.push_back(theDelayed);
            }

        }
    }
    //Return the list of created delayed neutrons.
    return dNeutrons;
}
```

```
void StorkDelayedNeutron::RenormalizePrecursors(G4double krun){


    for(G4int i = 0; i<6; i++){
        Precursors[i]/=krun;
    }
    return;
}



//Finds the isotope involved in the fission process.
G4int StorkDelayedNeutron::fissionIndex(G4double fEnergy, G4ThreeVector fSite)
{
    //Get the isotope by re-simulating the interacting of the material with the neutron.
    G4String iso = theFissionProcess->GetFissionNucleus(fEnergy,fSite).GetIsotope()->GetName();
    G4int index;

    //Select the correct data (using the index).
    if(iso == "U235"){
        if(fEnergy<0.0000005) index = 0;
        else index = 3;

    }
    else if(iso == "U238"){
        index = 6;

    }
    else if(iso == "U233"){
        if(fEnergy<0.0000005) index = 2;
        else index = 5;

    }
    else if(iso == "Pu239"){
        if(fEnergy<0.0000005) index = 1;
        else index = 4;
    }
    else if(iso == "Pu240"){
        index = 7;
    }
    else if(iso == "Th232"){
        index = 8;
    }
    else {
        index = 0;
    }
    return index;
}
```

```
#ifndef STORKDELAYEDNEUTRON_H
#define STORKDELAYEDNEUTRON_H

#include "StorkContainers.hh"
#include "StorkDelayedNeutronData.hh"
#include "G4ParticleDefinition.hh"
#include "G4ParticleTable.hh"

class StorkDelayedNeutron
{
    //Public methods
    public:
    //Constructor
    StorkDelayedNeutron(G4String dnFilename,G4double runDuration,G4int numPrimaries);

    //Destructor
    virtual ~StorkDelayedNeutron();

    //Method to add precursors between runs.
    void AddPrecursors();

    //Renormalize precursors
    void RenormalizePrecursors(G4double krun);

    //Get and set functions
    void SetPrecursors(std::vector<G4int> p) {Precursors = p;}
    std::vector<G4int> GetPrecursors() {return Precursors;}

    //Generate delayed neutrons through roulette of precursors.
    NeutronSources GetDelayedNeutrons(G4double runEnd);

    //Set the fission sites and energies.
    void SetFissionSource(MSHSiteVector fissionSites, DblVector fissionEnergies);

    //Private methods
    private:

    // Produce initial precursor groups
    G4bool GetInitialPrecursors(G4int numPrimaries);




    //Samples an isotope using incident fission energy and site, returns an index.
    G4int fissionIndex(G4double fEnergy, G4ThreeVector fSite);


    //Private variables
    private:
    //Initial fission file name.
    G4String delayedSourceFile;

    //Fission source data.
    MSHSiteVector fSites;
    DblVector fEnergy;
    std::vector<G4int> Precursors;

    G4ParticleDefinition *neutron;

    //Run duration
    G4double runDuration;

};

#endif
```