A Predictive Non-Intrusive Reduced Order Model of a High-Temperature Gas-Cooled Reactor using Operator Inference

TU Delft

T.K. Scheepmaker

A Predictive Non-Intrusive Reduced Order Model of a High-Temperature Gas-cooled Reactor using Operator Inference

Thomas Scheepmaker, 4666410

A Master thesis submitted to the TU Delft as partial fulfillment for the degree of:
Master of Applied Physics

TU Delft
To be defended publicly:
14 June 2024

Supervisor:
Dr. Ir. Danny Lathouwers

Daily supervisor:
Marc van den Berg

Thesis committee members:
Dr. Ir. Danny Lathouwers
Dr. Artur Palha da Silva Clérigo
Dr. Marlies Goorden

Abstract

In this thesis, a predictive non-intrusive reduced order model of a high-temperature gas-cooled reactor has been created. We first set out to create a representative multi-physics model coupling the neutronics and thermodynamics in the HTGR. To this end we first made a representative model of neutronics and thermodynamics separately in the HTGR. These representative models were used to train and test our reduced order models (ROMs). It has been found that the representative model converges spatially and temporally to the expected values of analytical and pseudo analytical solutions for the neutronics, the temperature, and the coupled model. However, the convergence in the spatial domain was found to be slower than the mathematical expectation from the implemented finite volume method, converging at a rate of $\Delta x^{1.5}$, instead of Δx^2. The temporal convergence rate was Δt as is expected from the implemented first order backward differentiation formula. In literature review, we have found that Operator Inference provides a way to construct ROMs that are both predictive and non-intrusive. We used Operator Inference to create three ROMs. A ROM of the temperature model, a ROM of the neutronics model, and lastly a ROM of the coupled model. The temperature ROM created with 12 modi and 1500 snapshots was found to be accurate to the representative model for any sort of power input, achieving a root mean square error below 10^{-5} compared to the representative model, while predicting 3000 seconds longer than the training at $\Delta t = 1$ s. for 1000 random power inputs not included in training. The neutronics ROM created with 7 modi and 3960 snapshots retrieved accurate results, with a root mean square error around 10^{-4} compared to the representative model if we used a homogeneous temperature in the reactor. If we defined more temperature zones in the reactor we used a different interpolation technique. For this we found the results to be of lesser quality with an average root mean square error closer to 10^{-2} for a temperature profile far from a steady state temperature profile, while the results were of greater quality with an average root mean square error of 10^{-5} for temperature profiles close to a steady state temperature profile. Finally, we developed two methods of creating a reduced order model of the coupled model. First, we created a sequential model, where the outputs of the individual temperature ROM and neutronics ROM fed into each other. Second, we created a merged model, in which both the temperature and neutronics were determined by our reduced order model in one single model. A loss of forced cooling was simulated by changing the heat transfer coefficient at the coolant channel boundary. We trained the ROMs for 50 seconds with $\Delta t = 10^{-3}$ s, with 5 different heat transfer coefficient equidistant between 0 and 0.03 $\mathrm{Wcm^{-2}K^{-1}}$. It has been found that the merged model was robust and provided accurate results, it could predict to 200 seconds estimating for a heat transfer coefficient of 0.00375 $\mathrm{Wcm^{-2}K^{-1}}$. with an average root mean square error in time for both the neutron flux profile and the temperature profile of less than 10^{-2}. The sequential model was unstable, with root mean square errors orders of magnitude higher than the merged model.

Acknowledgements

First and foremost, I want to express my sincere and deep gratitude to my supervisors, Marc van den Berg and Danny Lathouwers. By listening to my problems week in and week out, each time patiently explaining, theorizing and thinking with me, I was able to develop myself and complete this thesis to the utmost of my abilities. Furthermore, the exhaustive and concrete feedback provided by Marc at every stage of the thesis greatly enhanced and polished the result. Second, I want to thank Sanne for always being there for me in my times of need. She may not have always known, but she was always a great aid by my side through many hugs. Next, I want to thank the brokkoli broeders, without whom I could not have completed these long years studying applied physics. Working together and relaxing together has always given me a reason to work and enjoy my time not working. I would specifically like to thank Brent and Julian for our weekly Monday morning study sessions, during which I could always vent the problems I faced at that moment, and together with whom I could always find solutions to those problems. JC Compact also deserves my appreciation, though I was not always there for them, they were always there for me, giving me much-needed distraction. Physics is not my only passion; making music is an important part of my life. In Krashna, I can always fulfill this passion, and in Krashna I find a place to breathe and relax. This year, in which stress sometimes reached a boiling point, I could always come back down to earth and enjoy making amazing music together with amazing people. To all the people of Krashna that make that experience possible, thank you. No acknowledgement is complete without thanking your parents. It is they who stand at the root of what has been achieved here, and it is they whom I will always thank for everything.

Thomas Scheepmaker
Delft, June 2024

Contents

Abstract i

Acknowledgements ii

1 Introduction 1

2 Theory and Design 4
2.1 Nuclear fission . . . 4
2.2 Neutronics . . . 5
2.2.1 Two-Group Diffusion . . . 6
2.2.2 Delayed neutrons . . . 6
2.2.3 Multiplication factor . . . 7
2.3 Thermodynamics . . . 7
2.3.1 Reactor Power . . . 7
2.4 Coupling . . . 8
2.5 High-Temperature Gas-Cooled Reactor design . . . 8

3 Representative High-Fidelity Model 10
3.1 Discretization . . . 10
3.1.1 Root Mean Square error . . . 11
3.1.2 Spatial: Finite Volume Method . . . 11
3.1.3 Temporal: Backward Differentiation Formula . . . 14
3.2 Numerical Implementation . . . 15
3.2.1 Temperature . . . 15
3.2.2 Neutronics . . . 17
3.2.3 Coupled . . . 19

4 Reduced Order Model 23
4.1 Reduced Order Modelling . . . 23
4.2 Proper Orthogonal Decomposition . . . 24
4.3 Operator Inference . . . 26
4.4 Reduced Order Model implementation . . . 27
4.4.1 Preprocessing . . . 27
4.4.2 Parameters in OpInf . . . 27
4.4.3 Implementing OpInf for temperature . . . 30
4.4.4 Implementing OpInf for neutronics . . . 31
4.4.5 ROM of coupled temperature and neutronics code . . . 31

5 Results 34
5.1 Validation of the representative high-fidelity model . . . 34
5.1.1 Representative temperature model . . . 34
5.1.2 Representative neutronics model . . . 37
5.1.3 Representative coupled model . . . 40
5.2 Verification of ROM_T and ROM_N . . . 43
5.2.1 ROM_T . . . 43

5.2.2 ROM_T . . . 47
5.3 Coupled ROM of temperature and neutronics . . . 54
5.3.1 Preliminary test . . . 55
5.3.2 Training analysis . . . 57
5.3.3 Comparative case . . . 59

6 Discussion **63**
6.1 Representative High-Fidelity Model . . . 63
6.2 Reduced order model accuracy . . . 64

7 Conclusions **67**
7.1 Recommendations . . . 68

Bibliography **69**

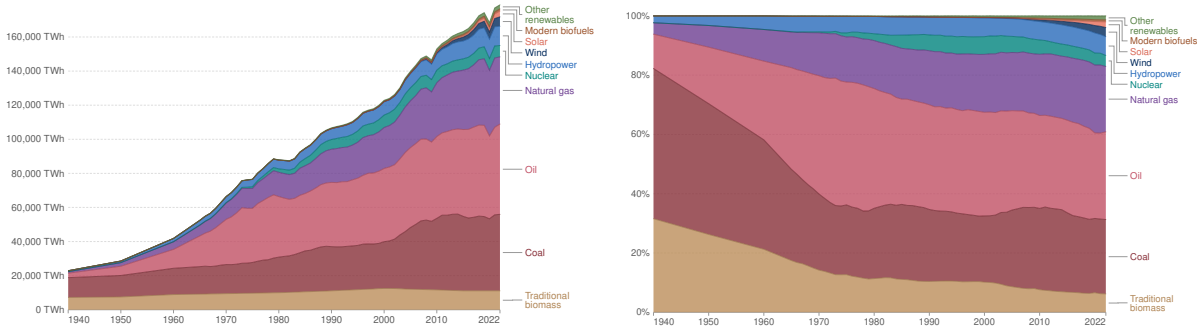
A Appendix **72**
A.1 Full derivation of the finite volume method . . . 72
A.1.1 Functional within domain . . . 72
A.1.2 Functional at left side of domain with Neumann boundary condition . . . 72
A.1.3 Functional at right side of domain with mixed boundary condition . . . 73
A.2 Parameters . . . 73
A.2.1 Temperature . . . 73
A.2.2 Neutronics . . . 73
A.2.3 Delayed . . . 74
A.3 1 group infinite homogeneous reactor temporal convergence . . . 75
A.4 Spatial and temporal convergence of the thermal flux of the representative model . . . 77
A.5 ROM_T test cases . . . 78
A.6 Average RMS error of the ROM_T using 3 fuel blocks . . . 79
A.7 Profiles of RMS errors in temperature parameter space for different modi and snapshots used in ROM_N . . . 80
A.8 Training analysis of mROM for training set with 3 training runs . . . 81
A.9 Training analysis of mROM for training set with 5 training runs . . . 82

Chapter 1

Introduction

Energy defines human progress. The question of what sources of energy are at a civilization's disposal is a determining factor of how advanced a civilization is [29]. Where first muscle power was our source of energy, we developed carbon based combustion. By burning coal, oil and gas we harnessed the power to move cities. However, since the 1970s it has been known that excessive burning of these carbon based energy sources will lead to irreversible climate change [15]. At the moment, we are trying to reduce our carbon emissions to avoid a global catastrophe. In 2015, 196 countries signed the Paris agreement, which aims to limit the maximum heating of the planet to 1.5 degrees Celsius [9]. In order to achieve this goal, we need to move to net zero emission energy sources.

As it stands today, only 18.2% of global energy requirements are achieved with renewable and nuclear energy [37], meanwhile global consumption has grown and will continue to grow. The efforts thus far are not enough. We can not rely completely on renewable energy, due to two drawbacks. First of all, the power generation of renewable sources is not constant, which is a problem for heavy industry [41]. Moreover, some sectors are very difficult to electrify, for example shipping. Shipping accounts for 3 % of carbon emissions [11], yet less than 1 % is electrified [10]. The difficulty in electrification comes from long distance voyages which would require recharging multiple times while en route [24].



(a) Global energy consumption by source, from 1940 to 2022. *(b) Energy source as percentage of the total.*

Figure 1.1: Graphs showing global energy consumption by source, and showing the source of energy as a percentage of the total, from [37].

To fill this gap and achieve the Paris goals, we look towards nuclear energy in general and Small Modular Reactors (SMR) in particular. The small size of SMRs means that the upfront investment costs are low compared to large nuclear power plants. Furthermore their modularity makes them competitive throughout their lifetime compared to these large plants [17]. This makes SMRs an interesting energy source for industry in both developed and developing countries. Their small size also allows for their usage as a power source in e.g. container ships, which are otherwise difficult to electrify.

In this work, we study the micro High-Temperature Gas-cooled Reactors (micro-HTGR). These reactors are inherently safe, in part due to their low power density and large heat capacity. Because of this, they form a very interesting decarbonized power source for the shipping industry [2].

In order to ascertain the safety of a reactor, such as the micro-HTGR, we need to model the reactor for all operational conditions in the lifetime of the reactor, in other words all transients caused by incidents. These models are computed using high-fidelity codes, such as the TU Delft in-house PHANTOM-S_n code, which provides valuable insight into the neutron populations inside a reactor. In a reactor the temperature and neutronics are inherently linked: temperature rises if there is more fission, and if the temperature rises the fission probability decreases. In computational models we couple the neutron population to the temperature in a reactor. The in-house calculations, however, are expensive to run, taking days at a time to calculate one event [14]. This makes them impractical if we want to research numerous transients or perform an uncertainty quantification.

We want to create a simulator of a nuclear reactor. In a simulator we want to model and simulate all possible operating transients, such a simulator should thus calculate these transients faster than the real-time events. Having a simulator of a nuclear reactor allows for quick iterations in the design of nuclear reactors. Due to the necessity that calculations should be faster than the real-time events, this simulator cannot use high-fidelity codes.

To make the nuclear reactor simulator, we want to create a Reduced Order Model (ROM) of the reactor. A ROM is a mathematical model used to approach a full dynamical system. A ROM is trained on dynamics of a system, and filters the most important of these dynamics, creating a space from these most important dynamics. By only keeping these most important dynamics, we reduce the orders of the system. You can think of a ROM as a Taylor expansion, with a Taylor expansion one can describe a complex function through a number of simpler polynomials. The first polynomials in the Taylor expansion are most important in approximating the function. Thus by only using these first few polynomials we can compress, or reduce the order of, the function [8].

For such a ROM to be successfully implemented in simulating a micro-HTGR, we want to train it with the in-house codes. Because these legacy codes are difficult to manipulate, we want the ROM to be non-intrusive, we want to reduce any model without a priori knowledge on that model. Furthermore we want our ROM to be predictive. In a micro-HTGR simulator we need to determine neutronics and temperature for an uncountable amount of transients, we could train a ROM for every possible transient, but it is less complicated if our ROM can predict transients of a coupled neutronics and thermodynamic code without being specifically trained for all transients. This leads us to pose the following research question:

Can we create a Reduced Order Model of a coupled neutronics and temperature micro-HTGR reactor code which is both non-intrusive and predictive?

To answer this question we created a number of sub questions:
Is there an existing non-intrusive and predictive ROM method?
Can we create a non-intrusive and predictive ROM of a temperature model?
Can we create a non-intrusive and predictive ROM of a neutronics model?
How can we combine a coupled temperature and neutronics code and create a ROM of that coupled code?

The stated goal of this thesis is thus to determine the possibility of creating the aforementioned ROM and investigating the best approach to create such a ROM. To provide this proof of principle, we created a representative model with which we can make the ROM. This representative model was constructed to quickly train and test ROMs with various transients.

To answer these questions and to fulfill our goal the thesis is set-up in the following way: In chapter 2 we shall discuss the necessary physical theory and the design of an HTGR. In chapter 3 we introduce the representative high-fidelity model, the model used and created in this thesis. In chapter 4 the a discussion of different methods to achieve a reduced order model of the representative model shall be put forth and the final methodology chosen will be explained. In chapter 5 we provide the results of the validation and verification of the representative and the reduced model. These results shall than be discussed in chapter 6. From these results and their discussions conclusions will be drawn in chapter 7. In chapter 8 some recommendations for further investigations shall be presented. After chapter 8 a bibliography is given, after which appendices are included.

This thesis is presented as the conclusion of the master end project, performed at the RPNM research group of the RID, under the supervision of Dr. Ir. Danny Lathouwers and Marc van den Berg, as partial fulfillment of the degree of master of applied physics at the TU Delft.

Chapter 2

Theory and Design

This chapter gives a brief overview of the physical theory needed to understand the neutronics and thermodynamics used in this thesis. Furthermore, it presents the basic design of the micro-HTGR on which this research is based.

2.1 Nuclear fission

In a nuclear reactor we use the energy from nuclear fission to generate electricity. Traditional nuclear reactors use Uranium 235 as fuel. This fuel is bombarded in the reactor core with neutrons. The neutrons incident on the fuel have a probability to be absorbed by the fuel, the individual U-235 atoms, after which they create Uranium 236. This atom then splits apart creating fission product, neutrons (on average 2.4 per reaction), and releasing a tremendous amount of energy. This process is shown in Figure 2.1.

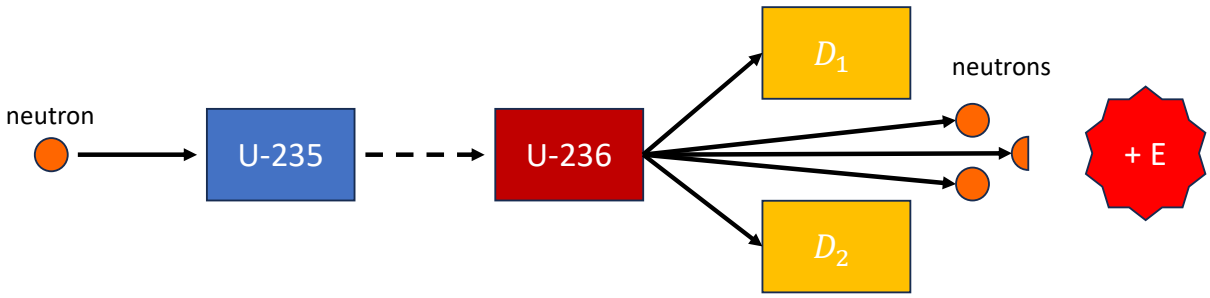


Figure 2.1: A schematic of the principle of fission. A neutron (circle on the left) is incident on the fuel (Uranium-235). This neutron is absorbed, after which the fuel becomes Uranium-236 and undergoes fission. After fission, two daughters are split off the original fuel atom (D_1 and D_2) together with on average 2.4 neutrons and energy.

In a traditional reactor this energy is used to heat water, from which we create steam, which is used to drive a turbine, generating electricity. In the HTGR the coolant is not water but helium, helium is heated by the fission energy, after which the heat of helium is exchanged to water which in turn drives a turbine.

Important in this process is the neutron population. Neutrons are needed in order to fission. However, with every fission event more neutrons are released than are needed for the fission event. We want the amount of neutrons to remain the same in the reactor, a so-called steady-state. We want this, because if the neutron population declines we are dealing with a sub-critical reactor, which will shut down, and if the neutron population increases we are dealing with a super-critical reactor, which in time will explode.

It is the task of the nuclear engineer to create a reactor in which the neutron population remains stable, a so called steady-state, this is done by designing the reactor in such a way that some, not too many and not too few, neutrons from the fission event can no longer induce another fission event. If the reactor deviates from this steady-state the reactor enters a transient. In the design of a reactor we introduce safety systems both passive (like negative temperature feedback) and active (like control rods) to make the reactor return to a steady-state.

2.2 Neutronics

In neutronics we look at the neutron population in time, specifically the neutron fluxes in time. The neutron scalar flux is a quantity describing neutron density in space with an associated speed, $\phi(x,t) = vN(x,t)\left[\text{cm}^{-2}\text{s}^{-1}\right]$, with $N(x,t)$ the neutron population density and v the neutron velocity. A full description of the change in fluxes is determined through the neutron transport equation. The transport equation is only solvable for simple cases. Therefore a simplified description, both mathematically and in its numerical implementation, is used: the neutron diffusion equation. This simplification holds true if the neutron source is isotropic, the medium is weakly absorbing, and the flux is not rapidly changing [20]. This is true in our reactor, apart from the boundaries, where the fluxes do change rapidly. The deviation of fluxes at the boundary is however relatively small, and given the stated goal of the thesis, we are allowed to make this simplification. Therefore the neutron diffusion equation will be used throughout this thesis. The neutron diffusion equation [20],

$$\frac{1}{v}\frac{\partial\phi(\mathbf{r},t)}{\partial t} - \nabla\cdot D\nabla\phi(\mathbf{r},t) + \Sigma_a\phi(\mathbf{r},t) = \nu\Sigma_f\phi(\mathbf{r},t), \tag{2.1}$$

gives us the the change of neutron fluxes in time and place. Here v is the velocity of the neutrons $[\text{cms}^{-1}]$, $\phi(\mathbf{r},t)$ the scalar neutron flux $\left[\text{cm}^{-2}\text{s}^{-1}\right]$, the amount of neutrons going through a volume at position $\mathbf{r}$ per second at time t, ∇ the spatial derivative, D the Diffusion coefficient $[\text{cm}^2]$, Σ_a the absorption cross-section, ν the amount of neutrons released per fission event, and Σ_f the fission cross-section. Cross-sections $\Sigma_i\,[\text{cm}^{-2}]$ give the probability that a neutron will undergo a certain type of reaction with an atom. In this equation we see that neutrons are only added through the fission term ($\nu\Sigma_f$), after which they diffuse away, or are absorbed.
We briefly discuss the diffusion coefficient D, this parameter is defined as,

$$D = \frac{1}{3\left(\Sigma_T - \bar{\mu}_0\Sigma_s\right)},$$

where Σ_T is the total cross-section, the sum of the absorption and scattering cross-sections. $\bar{\mu}_0$ is the average cosine of the scattering angle after a scattering event. This number is determined as $\bar{\mu}_0 = \frac{2}{3A}$ with A the mass number of the target nucleus [20] which is small for any nucleus larger than lithium, giving an approximation of the diffusion coefficient as,

$$D = \frac{1}{3\left(\Sigma_a + \Sigma_s\right)}.$$

In reactor physics we use homogenized cross-sections. This means that, even though the core contains a variety of different materials, we consider the reactor core itself as one single material.

In this thesis we will work with a representative model that determines neutronics in a 1 dimensional reactor, reducing equation 2.1 to,

$$\frac{1}{v}\frac{\partial\phi(x,t)}{\partial t} - \frac{\partial^2}{\partial x^2}D\phi(x,t) + \Sigma_a\phi(x,t) = \nu\Sigma_f\phi(x,t). \tag{2.2}$$

As neutrons move they have kinetic energy. The cross-sections of the neutron-matter interactions change depending on the kinetic energy of the neutrons [20]. Neutrons created from fission typically have a high energy. These so-called fast neutrons have a relatively low probability to fission with the uranium in the core. In order to increase the probability of fission, the neutrons need to have a lower energy. These low energy neutrons are called thermal neutrons. The lowering of energy is done through scattering, neutrons scatter consecutively against elements, such as carbon atoms, to slow down (moderate) to a lower energy level. In reality energy is continuous, and are classified in numerous energy groups, in this thesis we are working with a simplified model, only taking two of these energy groups into account, the fast and thermal group.

2.2.1 Two-Group Diffusion

Thermal neutrons are created by the down-scattering of fast neutrons. To account for this we add the thermal group to our diffusion equation 2.2 and include the scattering between the groups. In the resulting equation,

$$\begin{aligned} \textit{Fast}: \quad & \frac{1}{v_1}\frac{\partial \phi_1(x,t)}{\partial t} - D_1\frac{d^2\phi_1(x,t)}{dx^2} + \Sigma_{a_1}\phi_1(x,t) + \Sigma_{s_{12}}\phi_1(x,t) && = \nu_1\Sigma_{f_1}\phi_1(x,t) + \nu_2\Sigma_{f_2}\phi_2(x,t) \\ \textit{Thermal}: \quad & \frac{1}{v_2}\frac{\partial \phi_2(x,t)}{\partial t} - D_2\frac{d^2\phi_2(x,t)}{dx^2} + \Sigma_{a_2}\phi_2(x,t) && = \Sigma_{s_{12}}\phi_1(x,t), \end{aligned} \tag{2.3}$$

we see that the term $\Sigma_{s_{12}}$ is a loss term for the fast group accounting for the down-scattering of fast neutrons to thermal neutrons thereby constituting the source term for the thermal neutrons. Furthermore the thermal neutrons that cause fission are now a source in the fast group. Equation 2.3 almost provides a full-description of the neutronics in a reactor. In reality a nuclear reactor is not critical by prompt neutrons (those neutrons created from fission) alone. This would result in an extremely volatile steady-state, where a reactivity insertion of 0.1 would result in a reactor period of 0.1 s, which means that the power after 1 second is equal to 1×10^4 times the original power [38]. Instead, criticality is reached with delayed neutrons.

2.2.2 Delayed neutrons

Delayed neutrons arise from the fission products, mentioned in Section 2.1. After the uranium 236 splits apart a number of fission products are created. These fission products have decay chains of their own. In this decay process some daughters of the fission products, the so-called delayed neutron precursors, emit neutrons in their decay. These neutrons are delayed neutrons and are of paramount importance in the stability of a nuclear reactor. As having delayed neutrons greatly increases the reactor period, a reactivity insertion of 0.1 would now result in a reactor period of 70 s [38], which means that after 1 second the power in the reactor would be 1.01 times the original power, considerably lower than without delayed neutrons. Equation 2.4,

$$\begin{aligned} & \frac{1}{v_1}\frac{\partial \phi_1(x,t)}{\partial t} - D_1\frac{d^2\phi_1(x,t)}{dx^2} + \Sigma_{a_1}\phi_1(x,t) + \Sigma_{s_{12}}\phi_1(x,t) && = (1-\beta)\left[\nu_1\Sigma_{f_1}\phi_1(x,t) + \nu_2\Sigma_{f_2}\phi_2(x,t)\right] + \lambda C(x,t) \\ & \frac{1}{v_2}\frac{\partial \phi_2(x,t)}{\partial t} - D_2\frac{d^2\phi_2(x,t)}{dx^2} + \Sigma_{a_2}\phi_2(x,t) && = \Sigma_{s_{12}}\phi_1(x,t) \\ & \frac{dC(x,t)}{dt} && = \beta[\nu_1\Sigma_{f_1}\phi_1(x,t) + \nu_2\Sigma_{f_2}\phi_2(x,t)] - \lambda C(x,t), \end{aligned} \tag{2.4}$$

adds the precursor concentration to equation 2.3, as well as the delayed neutrons coming from decay of the precursors in $\lambda C(x,t)$, with λ the decay constant $[\mathrm{s}^{-1}]$ and C the concentration of the delayed neutron precursors. Furthermore β is the delayed neutron fraction, the fraction of neutrons coming from precursor decay. Note that this expression has 1 precursor group. In reality, there are numerous precursor groups, typically homogenized into 6 precursor groups. In this thesis, we chose to work with 1 group. This choice was made so we could verify the representative models with the

analytical reactor kinetics, provided in [20]. From the addition of the delayed neutrons a prompt jump can occur in the reactor. A prompt jump occurs if a transient is induced in the reactor. From [20] we see that power in time is at first characterised by prompt neutrons. This power is on a short time frame and can be quite large, due to the short lifetime of prompt, and the large amount of prompt neutrons in a reactor. As the lifetime of delayed neutrons is much larger, their effect on the power only comes at a later time frame, once we have reached this time frame the change in power is smaller compared to the prompt jump.

2.2.3 Multiplication factor

A reactor is in steady-state when its multiplication factor (k_{eff}) is 1. The multiplication factor describes the ratio of the neutron population in one generation to the neutron population in the preceding generation. Mathematically $k_{eff} = \frac{P(t)}{L(t)}$, the amount of neutrons produced in time $P(t)$ divided by the amount of neutrons lost in time $L(t)$. Furthermore, we define the neutron lifetime as $l_{eff} = \frac{N(t)}{L(t)}$, the average time until a neutron is lost in the reactor, defined as the total neutron population $N(t)$ at time t, divided by the neutron loss $L(t)$. With these ideas, we can define the change in neutron population in time as,

$$\frac{dN}{dt} = \frac{k_{eff} - 1}{l_{eff}} N(t), \tag{2.5}$$

where we see that if $k_{eff} = 1$ the neutron population does not change in time, achieving a steady-state.

2.3 Thermodynamics

This work couples neutronics with thermodynamics. We do not use the full thermodynamical theory, instead we use a specific simplification, the heat equation [13]. The heat equation is an expression that describes the diffusion of heat [12]. This means that we can only use this expression on solid materials, the full consideration of fluids in thermo-hydraulics is beyond the scope of this research. For the fluids within the reactor we assume a constant temperature. The one dimensional heat equation,

$$\rho c_p \frac{\partial}{\partial t} T(x,t) = \lambda \frac{\partial^2}{\partial x^2} T(x,t) + Q(x,t), \tag{2.6}$$

solves for temperature T [K] at time t and at position x. In this equation ρ is the density of the material [gcm^{-3}], c_p is the specific heat of the material [Jg^{-1}K^{-1}], λ is the heat transfer coefficient Wcm^{-1}K^{-1}, and $Q(x,t)$ is the heat source [W] at location x and time t. With this expression we can solve for the temperature in any solid.

2.3.1 Reactor Power

Reactor power provides the link temperature has with neutronics. A fission event releases about 200 MeV of energy [20]. To describe the power production $P(x,t)$ within a reactor we define,

$$P(x,t) = w_1 \Sigma_{f_1} \phi_1(x,t) + w_2 \Sigma_{f_2} \phi_2(x,t). \tag{2.7}$$

Power is dependent on the neutron fluxes multiplied by the fission cross-section and the energy release per fission event, w_1, w_2 [J] for respectively the fast and thermal fluxes. This power becomes the heat source in our heat equation 2.6, with which we can describe the coupling between the temperature and the neutronics.

2.4 Coupling

Inside a core neutronics and temperature are inherently linked in two ways. As mentioned in Section 2.3.1, the energy released from fission events is the direct heat source for the temperature calculation. Apart from temperature being linked to neutronics, neutronics is also influenced by temperature. Specifically cross-sections are influenced by temperature,

$$\begin{aligned}
\frac{1}{v_1}\frac{\partial \boldsymbol{\phi_1}(t)}{\partial t} - D_1(\mathbf{T})\frac{\partial^2 \boldsymbol{\phi_1}(t)}{\partial x^2} + \Sigma_{a_1}(\mathbf{T})\boldsymbol{\phi_1}(t) + \Sigma_{s_{12}}(\mathbf{T})\boldsymbol{\phi_1}(t) &= (1-\beta)[\nu_1\Sigma_{f_1}(\mathbf{T})\boldsymbol{\phi_1}(t) + \nu_2\Sigma_{f_2}(\mathbf{T})\boldsymbol{\phi_2}(t)] + \lambda \boldsymbol{C} \\
\frac{1}{v_2}\frac{\partial \boldsymbol{\phi_2}(t)}{\partial t} - D_2(\mathbf{T})\frac{\partial^2 \boldsymbol{\phi_2}(t)}{\partial x^2} + \Sigma_{a_2}(\mathbf{T})\boldsymbol{\phi_2}(t) &= \Sigma_{s_{12}}(\mathbf{T})\boldsymbol{\phi_1}(t) \\
\frac{d\boldsymbol{C}(t)}{dt} &= -\lambda \boldsymbol{C} + \beta[\nu_1\Sigma_{f_1}(\mathbf{T})\boldsymbol{\phi_1}(t) + \nu_2\Sigma_{f_2}(\mathbf{T})\boldsymbol{\phi_2}(t)].
\end{aligned} \tag{2.8}$$

In neutronics, we assume that the atoms on which neutrons are incident are stationary. In reality this is not the case, atoms move, moreso if their temperature increases. An increase in fuel temperature will lead to Doppler-broadening of resonance absorption [20], in other words more neutrons will be absorbed. In our computations, we use `SERPENT` a Monte Carlo code [30] to determine cross-sections at two temperatures: 294K and 2500K. When we want to determine cross-sections at any other temperature between these two, we use linear interpolation. The result of this coupling is that if the fluxes increase, the power in the core increases, resulting in increased temperature. This increased temperature in turn increases the absorption cross-section of the fuel, which leads to lower neutron fluxes, which lowers power. This creates a negative feedback which helps in creating passive stability in a reactor.

In our representative model we want to capture the working of the HTGR, including this negative feedback. Therefore we couple our temperature and neutronics model by setting the power from fluxes as input in the temperature equation and altering the cross-sections as a result of temperature change.

2.5 High-Temperature Gas-Cooled Reactor design

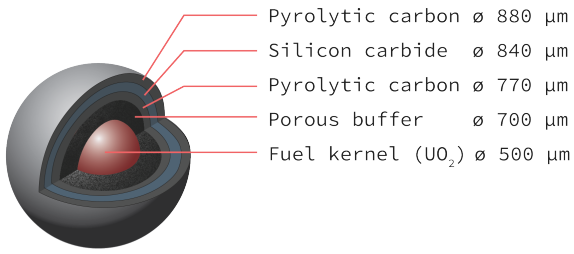
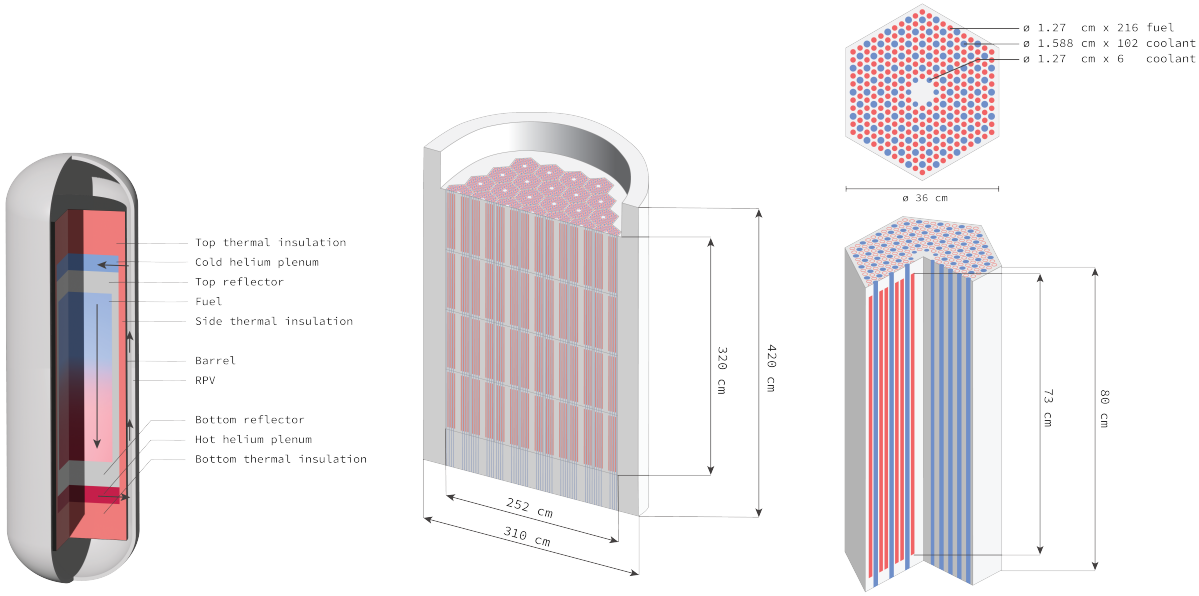


Figure 2.2: Schematic of a TRISO particle. The constituting elements and their diameter as measured from the centre are shown. From [32].

This research focuses on the micro-HTGR. This micro-HTGR, a specific SMR, is build as a prismatic design similar to the U-Battery [32]. The U-Battery [2] [19] was a micro-HTGR which would have been able to fulfill a role in power supply in remote or mobile applications. Though the U-Battery itself is discontinued [7], the idea of using such an SMR persists. In part due to their economic competitiveness with large plants, but also the inherent safe design of these micro-HTGRs. They operate at a low power density, have good thermal conductivity and use TRi-structural ISOtropic (TRISO) particles as fuel. A TRISO particle is a way to contain uranium fuel in a reactor fuel assembly [33], shown in Figure 2.2.

The micro-HTGR modelled here consists of hexagonal fuel assemblies which contain graphite moderator, helium coolant channels and fuel channels containing TRISO particles. Figure 2.3 shows the schematics of the micro-HTGR in question: an impression of the full reactor vessel, the reactor core with fuel assemblies, and a single fuel assembly out of the reactor core. Most of the fuel channels are made up of 20% enriched uranium TRISO particles, 20 fuel channels on the outside consist of thorium TRISO particles to give the reactor further stability. The helium coolant gas is let in at a temperature of 250°C, and has an outlet temperature of around 750°C. This reactor can produce between 10 and 20 MW of thermal energy at normal operations.



(a) Visualisation of the full core. Arrows indicate helium flow.

(b) schematic of the core with fuel assemblies with given sizes.

(c) Schematic of a single fuel assembly, including a 2D slice at an arbitrary z.

Figure 2.3: The schematics of the micro-HTGR discussed in this thesis. The fuel consists of TRISO particles, mostly of 20% enriched uranium, but with 20 outer fuel channels consisting of thorium. The coolant consists of helium gas between 250 and 750°C. The moderator in this reactor is graphite. Figures from [32].

Within this thesis we focus on the single fuel assembly of the HTGR. One fuel assembly holds the three distinct materials in the reactor, fuel, coolant, and graphite. Buy only taking one fuel assembly we capture all physical functioning of the system, while maintaining computational and numerical simplicity.

Chapter 3

Representative High-Fidelity Model

Determination of neutronics and temperature in a new reactor design is done in so-called high-fidelity codes. The TU Delft has two of these high-fidelity codes in-house, PHANTOM-S_n (based on the neutron transport equations) for neutronics and OPERA for temperature in a reactor. These high-fidelity codes form the Full Order Model (FOM) of a nuclear reactor.

Instead of working directly with these in-house high-fidelity models, a representative high-fidelity one-dimensional model, acting as a surrogate FOM, was created in Python. Creating such a representative model provides a number of benefits. First of all, by creating this representative model, the underlying physics of a ROM are more apparent to the researcher, allowing straightforward analysis. Secondly, the Python model is more user-friendly. The computational language of Python is more widely known and understood than the in-house FORTRAN codes. Adding to this Python packages are available for easy visual representation of models, data, and computation. Finally, most importantly, this one-dimensional representative model is orders of magnitude faster than the FOM of a reactor core. Where a timestep in an in-house code takes about 2 hours, a timestep in a representative model takes a fraction of a second [39]. This allows for quick iterations through different types and methods of ROMs, making it easier to investigate the effects of the different parameters in constructing the ROM.

In this chapter the spatial and temporal discretization through respectively the Finite Volume Method (FVM) and the Backward Differentiation Formula (BDF) will be explained. After which the specific implementation of these schemes in temperature and neutronics code will be presented. At the end the way in which we couple the temperature and the neutronics will be shown.

3.1 Discretization

The representative model constructed in this research is based on the diffusion equations of temperature and neutronics as presented in chapter 2. These equations solve a one-dimensional representation of a reactor through Partial Differential Equations (PDE). We use FVM and BDF to discretize in space and time.

To go through the discretization methods, we shall use the thermal PDE, from equation 2.6,

$$\rho c_p \frac{\partial}{\partial t} T(x,t) = \lambda \frac{\partial^2}{\partial x^2} T(x,t) + Q(x,t). \tag{3.1}$$

3.1.1 Root Mean Square error

When using FVM the numerical Root Mean Square (RMS) error decreases quadratically with the grid size Δx, when using BDF the numerical RMS error decreases linearly with the time-step Δt. We define the RMS error as,

$$RMS_{error} = \sqrt{\frac{1}{N}\frac{\Sigma\left(y-\hat{y}\right)^2}{\Sigma\hat{y}^2}}. \tag{3.2}$$

Where N is the amount of control volumes, y is the analytical or high-fidelity value, and $\hat{y}$ is the estimated value, in this thesis either the representative model or a ROM. We integrate over the entire domain, where in the discretization this integral becomes a sum over all control volumes.

3.1.2 Spatial: Finite Volume Method

In space we discretize using the FVM. Whose defined RMS error helps us to see whether our numerical implementation is correct. FVM was chosen due to the relative ease in its computational implementation.

In space we define our domain as $D \in [0, L]$. For the spatial discretization we will look at the steady-state solution of equation 3.1, in other words: $\frac{\partial}{\partial t}T(x,t) = 0$, giving us the time independent version of equation 3.1 as,

$$-\lambda\frac{\partial^2}{\partial x^2}T(x,t) = Q(x,t). \tag{3.3}$$

This thermal PDE is subject to boundary conditions. We are here concerned with two specific natural boundary conditions: homogeneos Neumann and mixed [28],

$$\begin{aligned} Neumann: \left.\frac{\partial}{\partial x}T(x,t)\right|_{x=B_1} &= 0, \\ Mixed: \left.-\lambda\frac{\partial}{\partial x}T(x,t)\right|_{x=B_2} &= \alpha\left(T(x,t)-T_{ref}\right). \end{aligned} \tag{3.4}$$

Where $B_{1,2}$ are the boundary points to be defined later, α is the heat transfer coefficient at the boundary [$\mathrm{Wcm^{-2}K^{-1}}$]. T_{ref} [K] is a coolant temperature outside the boundary.

We make use of a homogeneous Neumann boundary when a system has symmetry. When we deal with diffusion in a symmetric system, the spatial derivative at the point of symmetry is 0, giving the homogeneous Neumann boundary. The mixed boundary is used in cases where there is diffusion across a boundary.
We split the domain D into N_x equidistant subintervals e_k. It should be noted that in principle an equidistant spatial grid is not strictly necessary. However for computational simplicity (especially when we couple neutronics and temperature) it has been decided to work with an equidistant grid.

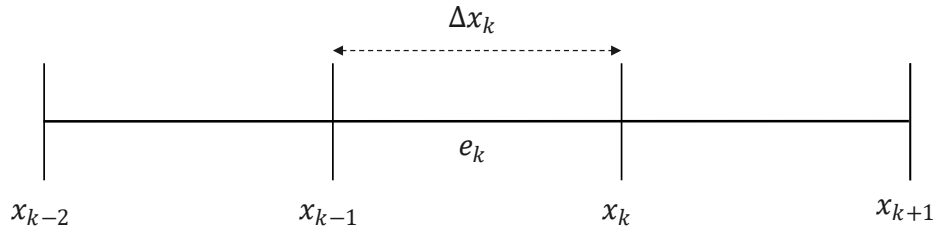


Figure 3.1: Illustration of the discretized grid used in the representative model. Every subinterval e_k is of length Δx, and is bordered by points x_{k-1} and x_k. [28]

In Figure 3.1 a section of the discretized grid is shown. The subintervals into which the domain is split are denoted as e_k, with $k = 1, 2, ..., N$. Each subintervals is enclosed by two points: x_{k-1} and x_k, with the subintervals having length $\Delta x_k = x_k - x_{k-1}$.

In the implementation of FVM we denote three types of control volumes: an internal control volume, a control volume subject to a homogeneous Neumann boundary, and a control volume subject to a mixed boundary.

FVM within the domain

To find the temperature at x_k we need to discretize equation 3.1 at that point. The first step is shown in Figure 3.2, we create a control volume with the point of interest x_k in the centre. This control volume is bordered by two new points, halfway in between the next points of interest: $x_{k-\frac{1}{2}}$, between x_k and x_{k-1}, and $x_{k+\frac{1}{2}}$, between x_k and x_{k+1}.

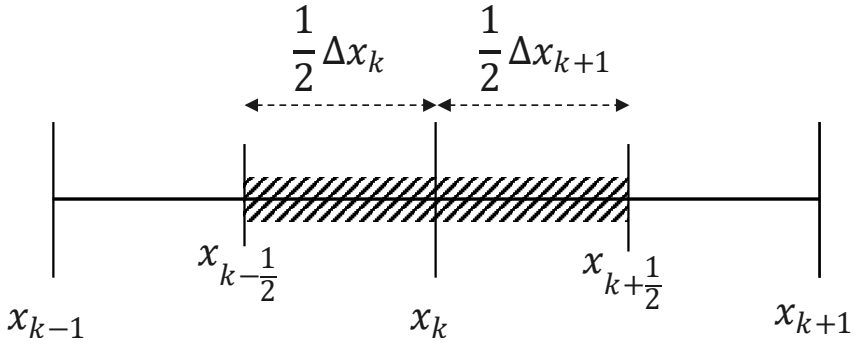


Figure 3.2: Illustration of the subdomain defined within the domain. The subdomain has borders at $x_{k-\frac{1}{2}}$ and $x_{k+\frac{1}{2}}$, within which the point x_k resides. This subdomain has length $\frac{1}{2}\Delta x_k + \frac{1}{2}\Delta x_{k+1}$ which in our implementation is equal to Δx_k

By integrating equation 3.3 over the control volume $D_1 \in [x_{k-\frac{1}{2}}, x_{k+\frac{1}{2}}]$ (the dashed area in Figure 3.2) and taking central divided differences to work away the left over derivative, we are able to obtain an expression of equation 3.3 at point x_k. The complete derivation is included in appendix A.1.1. The resulting expression for temperature (from [28]) at point x_k, T_k, is

$$\frac{-\lambda_{k-1/2}}{\Delta x_k} T_{k-1} + \left(\frac{\lambda_{k-1/2}}{\Delta x_k} + \frac{\lambda_{k+1/2}}{\Delta x_{k+1}} \right) T_k - \frac{\lambda_{k+1/2}}{\Delta x_{k+1}} T_{k+1} = \frac{1}{2}(\Delta x_k + \Delta x_{k+1}) Q_k + E(\Delta x). \tag{3.5}$$

In this equation, λ_i is the thermal conductivity at point i. When this point is in between actual points, we determine the harmonic mean between those points: $\lambda_{k-\frac{1}{2}} = \frac{2\lambda_k \lambda_{k-1}}{\lambda_k + \lambda_{k-1}}$. Δx is the size of the control volume, given in Figure 3.2 as $\frac{1}{2}\Delta x_k + \frac{1}{2}\Delta x_{k+1}$. Q_k is the heat inserted at point k, T_i is the temperature at point i, and $E(\Delta x)$ is the truncation error of this numerical method compared to the physical system, or analytical solution, which is a quadratic function of Δx (Δx^2) in an RMS error sense.

Neumann boundary

We take from equations 3.4 the homogeneous Neumann boundary at the left side of our domain, at $x = 0$. The boundary condition at this point thus reads $\frac{\partial}{\partial x}T(x,t)\Big|_{x=0} = 0$. The left side of Figure 3.3 shows the graphical representation of this boundary control volume, $D_2 \in [0, x_{\frac{1}{2}}]$. To evaluate equation 3.3 at this point we again integrate over the subdomain, now D_2, arriving at equation 3.6,

$$\frac{\lambda_{1/2}}{\Delta x_1}T_0 - \frac{\lambda_{1/2}}{\Delta x_1}T_1 = \frac{1}{2}\Delta x_1 Q_0 + E(\Delta x). \tag{3.6}$$

The full derivation (from [28]) is included in appendix A.1.2. The entries of equation 3.6 have the same meaning as equation 3.5.

Mixed boundary

We take the mixed boundary at the right side of the domain, at $x = L$. The natural boundary condition becomes: $-\lambda\frac{\partial}{\partial x}T(x,t)\Big|_{x=L} = \alpha\left(T(x,t) + T_{ref}\right)$. This boundary is shown in Figure 3.3. To evaluate the PDE at this boundary, we take the integral over the new subdomain $D_3 \in [x_{N-\frac{1}{2}}, x_N]$ giving equation 3.7 [28],

$$-\frac{\lambda_{N-1/2}}{\Delta x_N}T_{N-1} + \left(\frac{\lambda_{N-1/2}}{\Delta x_N} + \alpha\right)T_N = \frac{1}{2}\Delta x_N Q_N + \alpha T_{ref} + E(\Delta x), \tag{3.7}$$

whose full derivation is included in appendix A.1.3. The entries are the same as equation 3.5, now with T_{ref}, the reference temperature which, in our case, is the temperature of our coolant channel.

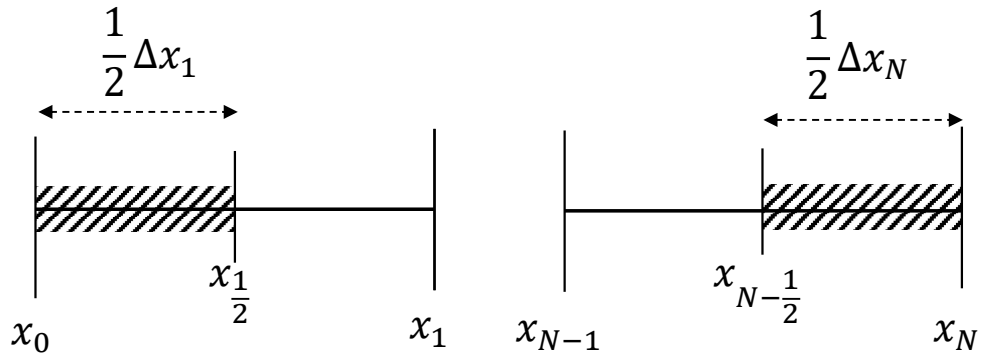


Figure 3.3: Schematics of the left boundary in the FVM implementation on the left and the right boundary in our FVM implementation on the right. For the left boundary, the point of interest, x_0 is bound only on one side by $x_{\frac{1}{2}}$, our subdomain has length of $\frac{1}{2}\Delta x_1$. For the right boundary, the point of interest, x_N is bound only on one side by $x_{N-\frac{1}{2}}$, our subdomain now has length of $\frac{1}{2}\Delta x_N$.

3.1.3 Temporal: Backward Differentiation Formula

In time we work in the domain $T \in [0, t_{final}]$. This domain is discretized in N_t parts, each lasting Δt long. It should be noted that the time steps do not necessarily need to be identical. In this work they have been chosen to be identical for computational simplicity.
We take the full temperature PDE, equation 3.1. We discretize the time derivative using BDF using one time step.

We first discretize the time derivative

$$\rho c_p \frac{\partial}{\partial t} T(x,t) \approx \rho c_p \frac{T(x,t+\Delta t) - T(x,t)}{\Delta t} = \rho c_p \frac{T(x)^{(n+1)} - T(x)^{(n)}}{\Delta t}.$$

In BDF, this temporal derivative is equal to the functional at the next time step, so we formulate equation 3.1, stepping forward in time,

$$\rho c_p \frac{T(x)^{(n+1)} - T(x)^{(n)}}{\Delta t} = \lambda \frac{\partial^2}{\partial x^2} T(x)^{(n+1)} + Q(x)^{(n+1)} + E(\Delta t). \tag{3.8}$$

In expression 3.1.3 $Q(x)^{(n+1)}$ denotes the power input at the next time step. If this power input is not yet known we use as an approximation that $Q(x)^{(n+1)} = Q(x)^{(n)}$. The error term $E(\Delta t)$ is a temporal truncation error linear in Δt.

3.2 Numerical Implementation

In the following section the specific numerical implementation for temperature and neutronics within our reactor is described.

3.2.1 Temperature

For both neutronics and temperature we simplify the HTGR from [18] to a 1 dimensional reactor. We do this taking a z coordinate roughly in the middle, and using the radius as our 1 dimensional coordinate, in this thesis we call this coordinate x. This is done for computational simplicity and speed. We model the fuel assembly by defining a fuel block, which repeats in the core. This is shown in Figure 3.4, a close-up of Figure 2.3c.

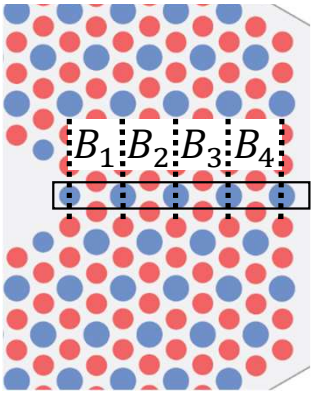


Figure 3.4: A close-up of a 2D slice of a fuel assembly of an HTGR. We take a cross-section within this slice (delimited with a full black line). In this cross-section we identify a repeating geometry, which we call fuel blocks (whose borders within the cross-section are shown with dotted lines), in the section chosen we find 4 fuel blocks, denoted as B_1, B_2, B_3 and B_4. Each fuel block consists of a fuel channel, graphite channels, and coolant channels.

A fuel block consists of three parts, a fuel channel in the centre surrounded by moderator graphite channels, which themselves are enclosed by coolant channels. This configuration is shown in Figure 3.5.

In our numerical implementation we do not actually determine the temperature within the coolant channels. The coolant channel contains a helium gas, which requires thermo-hydraulics to accurately determine the temperature. These considerations are beyond the scope of this research. Therefore we assume a constant temperature in the coolant channels, where every coolant channel may have a distinct temperature. This means that within a fuel block such as Figure 3.5 we only determine temperature within an area, called the cell (L_{Cell}) and not in the entire fuel block (L_{Block}).

In mathematical terms this means that the temperature domain is $D_T \in [-L_{Cell}/2, L_{Cell}/2]$ with $L_{Cell} = 2 \cdot r_f + 4 \cdot r_g$. At both the left and the right boundary (respectively $x = -L_{Cell}/2$ and $x = L_{Cell}/2$), we use a mixed boundary with the temperature of the coolant channel being the reference temperature. Heat is only added through fission in the fuel channel, and dissipated only at the coolant channels.

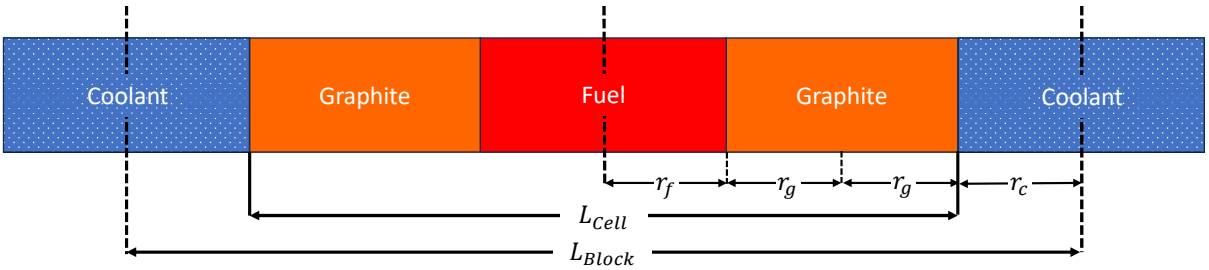


Figure 3.5: Different channels within a fuel vlock. The temperature mesh is referred to as the fuel block. In the middle of the block is the fuel channel, this fuel is $2 \cdot r_f$ wide, the fuel is enclosed on each side by a graphite moderator, $2 \cdot r_g$ wide, at the edges the coolant channels reside, both $2 \cdot r_c$ wide. Within this mesh, we define L_{Block} as the entire length of one single fuel block and L_{Cell} as the length of the cell of which we can determine the temperature numerically.

In Figure 3.5 the graphical interpretation of the temperature mesh is shown. The mesh which we use is made up of N_x control volumes of size Δx, which come from the discretization of D_T.

In our numerical implementation we solve the heat diffusion equation,

$$\rho c_p \frac{\partial}{\partial t} T(x,t) = \lambda \frac{\partial^2}{\partial x^2} T(x,t) + Q(x,t),$$

in two ways. First spatially and then temporal.
In space, we can solve the entire temperature PDE using the Python library of SciPy, specifically the `scipy.sparse.linalg.spsolve` library [40]. This library uses iterative solving methods to solve sparse matrices, and is able to solve problems of the form of $\underline{\mathbf{A}}\mathbf{x} = \underline{\mathbf{B}}$, for sparse matrix $\underline{\mathbf{A}}$, vector or matrix $\underline{\mathbf{B}}$, and unknown vector $\mathbf{x}$. In order to use this library, we formulate the PDE as,

$$\underline{\mathbf{A}}\mathbf{T}(t) = \mathbf{Q}(t). \tag{3.9}$$

Here $\mathbf{T}$ is the temperature vector with length N_x. $\underline{\mathbf{A}}$ is the diffusion matrix,

$$\underline{\mathbf{A}} = \frac{-1}{\Delta x^2} \begin{bmatrix} \lambda_{\frac{1}{2}} & -\lambda_{\frac{1}{2}} & 0 & \cdots & 0 \\ -\lambda_{\frac{1}{2}} & \lambda_{\frac{1}{2}} + \lambda_{1+\frac{1}{2}} & -\lambda_{1+\frac{1}{2}} & 0 & \\ 0 & -\lambda_{1+\frac{1}{2}} & \lambda_{1+\frac{1}{2}} + \lambda_{2+\frac{1}{2}} & -\lambda_{2+\frac{1}{2}} & \vdots \\ \vdots & & & & \\ & \cdots & & -\lambda_{N-\frac{1}{2}} & \lambda_{N-\frac{1}{2}} + \alpha\Delta x \end{bmatrix},$$

to be precise $\underline{\mathbf{A}}$ combines equations 3.5, 3.6 and 3.7. $\mathbf{Q}$ is the input vector, through which energy is added to the system.

With this formulation we can solve the system for the unknown variable $\mathbf{T}$. After this we use the BDF formula from equation 3.8, rewriting it a bit to arrive at a matrix representation which allows us to step in time through

$$\begin{gathered} \left(\rho c_p + \Delta t \lambda \frac{\partial^2}{\partial x^2}\right) T(x)^{(n+1)} = \rho c_p T(x)^{(n)} + \Delta t Q(x)^{(n+1)}, \\ \text{or}: \quad \underline{\mathbf{A}}\mathbf{T}^{n+1} = \mathbf{B}^n. \end{gathered} \tag{3.10}$$

Where $\mathbf{T}^{n+1}$ is the temperature vector at time $n+1$ with length N, $\underline{\mathbf{A}}$ is the operator matrix, and $\mathbf{B}$ is the solution matrix, made up of Q^{n+1} and T^n. With this system, we again make use of the Scipy library.

3.2.2 Neutronics

In neutronics we are interested in a 1 dimensional representation of the HTGR. We take an arbitrary position z of the core shown in 2.3b, at this z we extend from the centre to the edge of the reflector. In our model this representation is made up of the core, consisting of a homogenized core and a reflector. From chapter 2, the core consists of fuel assemblies. We have further divided the fuel assembly into fuel blocks, whose shape are as given in Section 3.2.1, which will become important once we couple temperature and neutronics. From this we can create the mesh shown in Figure 3.6:

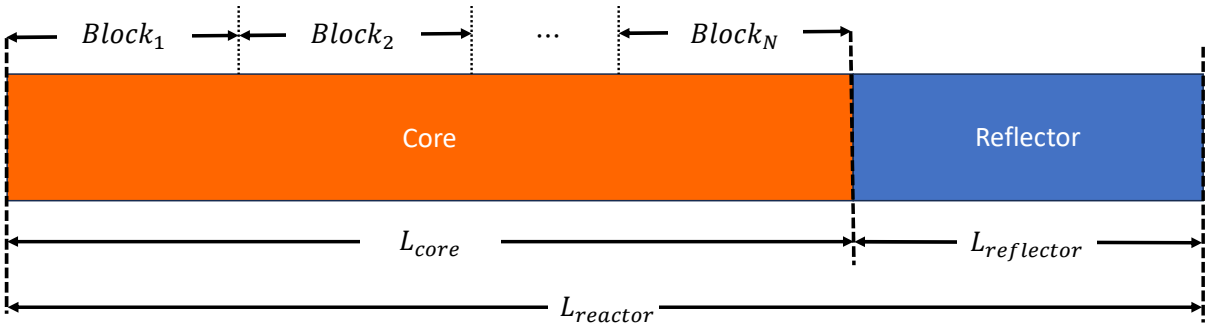


Figure 3.6: The spatial mesh of the reactor. The core is of length L_{core} made up of N blocks consisting of fuel, graphite, and coolant channels. Added to the core is a reflector, of length $L_{reflector}$. Reflector and core combined give the full reactor geometry.

Figure 3.6 shows the geometry used in the neutronics calculation. The core consists of N_{blcoks} fuel blocks, whose composition is given in Figure 3.5. Here any $Block_i$ has length L_{Block} from Figure 3.5. This neutronics mesh is subsequently divided into $N_n = L_{reactor}/\Delta x$ control volumes.Δx being the same as in the temperature mesh.

In neutronics our domain is $D_N \in [0, L_{reactor}]$. In reality the reactor is symmetric around 0, because of the symmetry of the reactor, we only have to model half the reactor, using a homogeneous Neumann boundary at the left side ($x = 0$). At the right side ($x = L_{reactor}$) we use an extrapolated boundary condition [20] : $\frac{\phi_g}{4} + \frac{D_g}{2}\frac{d\phi_g}{dx}\Big|_{x=L_{reactor}} = 0$. This extrapolated boundary is used because there is only an outgoing flux at this point. Mathematically seen it is a version of the mixed boundary, as given in equation 3.7.

We assume a homogenized fuel, graphite, and coolant mixture in the core, for which we use homogenized cross-sections from SERPENT [30].

As with temperature, we respectively deal with a spatial and a temporal solution. When we look at the spatial solution of the neutronics, we take $\frac{d\phi}{dt} = 0$, as such we determine the steady state of the reactor. This is done through a power iteration scheme.

Power iteration

The direct analytical expression for the steady-state neutron fluxes strictly only has a solution if the multiplication factor (k_{eff}) is equal to 1. This is however not always directly the case. Therefore we add the factor of $\frac{1}{k_{eff}}$ to our equation 2.3, and use a power iteration scheme to successively solve the thermal and fast neutron populations until we have reached a solution for the multiplication factor that we choose. We take a modified 2 group neutronics equation (equation 2.3),

$$
\begin{aligned}
\textit{Fast}: \quad & -D_1\frac{d^2\phi_1(x,t)}{dx^2}+\Sigma_{a_1}\phi_1(x,t)+\Sigma_{s_{12}}\phi_1(x,t) && = \frac{1}{k_{eff}}\left[\nu_1\Sigma_{f_1}\phi_1(x,t)+\nu_2\Sigma_{f_2}\phi_2(x,t)\right], \\
\textit{Thermal}: \quad & -D_2\frac{d^2\phi_2(x,t)}{dx^2}+\Sigma_{a_2}\phi_2(x,t) && = \Sigma_{s_{12}}\phi_1(x,t).
\end{aligned}
\tag{3.11}
$$

Here ϕ is the 2 group energy dependent neutron scalar flux, with the subscript denoting whether it is fast (1) or thermal (2). All neutrons from fission are born in the fast group. When these fast neutrons have scattered, they are moderated to the thermal group. It is important to note that up-scattering from the thermal group to the fast group ($\Sigma_{s_{21}}$) is possible, but the probability is an order of magnitude smaller, and is thus not taken into account in this thesis.

In solving this system, we can solve the fast and thermal fluxes for any k_{eff}. This is however not directly the true k_{eff} of the system, for that we use a power iteration scheme,

$$
\begin{aligned}
\underline{\mathbf{L_1}}\boldsymbol{\phi_1}^{n+1} &= \frac{1}{k_{eff}^n}\underline{\mathbf{S}}^n, \\
\underline{\mathbf{L_2}}\boldsymbol{\phi_2}^{n+1} &= \Sigma_{s_{12}}\boldsymbol{\phi_1}^{n+1}, \\
\underline{\mathbf{S}}^{n+1} &= \nu_1\Sigma_{f_1}\boldsymbol{\phi}_1^{n+1}+\nu_2\Sigma_{f_2}\boldsymbol{\phi}_2^{n+1}, \\
k_{eff}^{n+1} &= \frac{\int dx\underline{\mathbf{S}}^{n+1}(x)}{\frac{1}{k_{eff}^n}\int dx\underline{\mathbf{S}}^{n}(x)}.
\end{aligned}
\tag{3.12}
$$

We solve separately for the neutron populations at step $(n+1)$ using `scipy.sparse.linalg.spsolve`. In this equation $\underline{\mathbf{L_{1,2}}}$ is the left side of respectively the fast and thermal part of equation 3.11. The $\underline{\mathbf{S}}^n = \nu_1\Sigma_{f_1}\boldsymbol{\phi}_1^n+\nu_2\Sigma_{f_2}\boldsymbol{\phi}_2^n$ matrix is the source for the fast neutrons, defined by the fluxes of the previous step. By solving for the fluxes in this iteration, we find $\underline{\mathbf{S}}^{n+1}$, with which we can determine k_{eff}^{n+1}. This power iteration stops when the difference between k_{eff}^{n+1} and k_{eff}^n is smaller than a given tolerance. The full derivation of this method is given in [20]. By using the power method, we can solve for k_{eff} and the respective fluxes of the system.

Time-dependent

When we deal with time-dependent neutronics, we need to take in mind delayed neutrons, as explained in Section 2.2.2. This means that for 2 group neutronics our equation will be,

$$
\begin{aligned}
&\frac{1}{v_1}\frac{\partial\boldsymbol{\phi_1}(t)}{\partial t}-D_1\frac{\partial^2\boldsymbol{\phi_1}(t)}{\partial x^2}+\Sigma_{a_1}\boldsymbol{\phi_1}(t)+\Sigma_{s_{12}}\boldsymbol{\phi_1}(t)=(1-\beta)[\nu_1\Sigma_{f_1}\boldsymbol{\phi_1}(t)+\nu_2\Sigma_{f_2}\boldsymbol{\phi_2}(t)]+\lambda\boldsymbol{C}, \\
&\frac{1}{v_2}\frac{\partial\boldsymbol{\phi_2}(t)}{\partial t}-D_2\frac{\partial^2\boldsymbol{\phi_2}(t)}{\partial x^2}+\Sigma_{a_2}\boldsymbol{\phi_2}(t)=\Sigma_{s_{12}}\boldsymbol{\phi_1}(t), \\
&\frac{d\boldsymbol{C}(t)}{dt}=-\lambda\boldsymbol{C}+\beta[\nu_1\Sigma_{f_1}\boldsymbol{\phi_1}(t)+\nu_2\Sigma_{f_2}\boldsymbol{\phi_2}(t)],
\end{aligned}
\tag{3.13}
$$

from [20]. In comparison to equation 3.11, the time dependence of the fluxes is added, as well as the velocity of the neutrons v_g. Of importance now is the precursor population, $\mathbf{C}$. In our system all delayed neutrons from precursors are added to the fast neutron group. In this research only 1 group was taken into account. This was done in order to verify with the analytical solutions of the point reactor kinetics equations [20].

For the numerical implementation we take the method derived by Pautz and Birkhofer [34]. This method describes the neutron transport equation of g groups using BDF, with the rewritten precursor population. This means that we only need to solve for the neutron populations at timestep $(n+1)$, instead of also having to solve for the precursor population at timestep $(n+1)$.

Apart from the delayed neutrons, the general method of solving the time-dependent neutron equation closely resembles that of the temperature calculation. Modifying Pautz's transport equation to a diffusion equation of 2 groups of neutrons gives,

$$\begin{aligned}
\left(-D_1\frac{\partial^2}{\partial x^2}+\Sigma_{a_1}+\Sigma_{s_{12}}+\frac{1}{v_1\Delta t}-(1-\beta)\,\nu_1\Sigma_{f_1}\right)\boldsymbol{\phi}_1^{n+1} &= (1-\beta)\,\nu_2\Sigma_{f_2}\boldsymbol{\phi}_2^{n}+\lambda\gamma\frac{1}{\Delta t}\mathbf{C}^n+\frac{1}{v_1\Delta t}\boldsymbol{\phi}_1^n,\\
\left(-D_2\frac{\partial^2}{\partial x^2}+\Sigma_{a_2}+\frac{1}{v_2\Delta t}\right)\boldsymbol{\phi}_2^{n+1} &= \Sigma_{s_{12}}\boldsymbol{\phi}_1^{n+1}+\frac{1}{v_2\Delta t}\boldsymbol{\phi}_2^n,\\
\mathbf{C}^{n+1} &= \frac{1}{\Delta t}\gamma\mathbf{C}^n+\beta\gamma\left(\nu_1\Sigma_{f_1}\boldsymbol{\phi}_1^{n+1}+\nu_2\Sigma_{f_2}\boldsymbol{\phi}_2^{n+1}\right),\\
With: \quad \gamma &= \left(\frac{1}{\Delta t_n}+\lambda\right)^{-1}.
\end{aligned} \tag{3.14}$$

Equation 3.14 comes from equation 3.13, the notable difference is the implementation of the BDF, and the rewriting of the precursor population term $\mathbf{C}$, through the introduction of a γ. To numerically solve this, we discretize spatially using FVM, and rewrite equation 3.14 as a system of matrices,

$$\begin{aligned}
\underline{\mathbf{L_1}}\boldsymbol{\phi}_1^{n+1} &= \underline{\mathbf{S_1}}^n,\\
\underline{\mathbf{L_2}}\boldsymbol{\phi}_2^{n+1} &= \Sigma_{s_{12}}\boldsymbol{\phi}_1^{n+1}+\frac{1}{v_2\Delta t}\boldsymbol{\phi}_2^n,\\
\mathbf{C}^{n+1} &= \frac{1}{\Delta t}\gamma C^n+\beta\gamma\left(\nu_1\Sigma_f^1\boldsymbol{\phi}_1^{n+1}+\nu_2\Sigma_f^2\boldsymbol{\phi}_2^{n+1}\right).
\end{aligned} \tag{3.15}$$

Here $\underline{\mathbf{L_{1,2}}}$ are the operator matrices, being composed of the parameters applied to $\boldsymbol{\phi}_1^{n+1}$ and $\boldsymbol{\phi}_2^{n+1}$ respectively. The $\underline{\mathbf{S_1}}$ matrix compromises the right hand side of the first line of equation 3.15. This formulation allows us to once more use the `scipy.linalg.sparse.spsolve` library, to solve iteratively for $\boldsymbol{\phi}_1^{n+1}$,$\boldsymbol{\phi}_2^{n+1}$, and $\mathbf{C}^{n+1}$ at each time step.

3.2.3 Coupled

In chapter 2, the inherent coupling of temperature and neutronics were discussed. The cross-sections of the materials change depending on their temperature. Within the coupled code we identify two schemes. Finding the power for a steady-state reactor, and a general transient coupled code. In both we use the neutronics and temperature schemes developed in the previous sections.

Mapping between temperature and neutronics

The main difficulty in coupling is the different domains of the temperature and neutronics scheme. In temperature we do not model the coolant channels. So we need to determine a way to map between the temperature mesh and the neutronics mesh. This mapping is done through a number of steps shown below. We take the size of the discretized control volumes to be the same, to ensure that the power we derive from neutronics can be defined per control volume and be directly implemented in the temperature calculation.

For temperature to neutronics we use the following scheme:

1. From our temperature mesh, we determine the temperature per control volume in $T_{Cell,i}$
2. We take the average temperature of the cell ($\overline{T_{Cell}}$) and call this average temperature T_{Block} : $T_{Block,i} = \overline{T_{Cell,i}}$
3. In our neutronics mesh we use $T_{Block,i}$ to find the cross-sections comprising the control volumes in *Blocki*.

This method is visualised in Figure 3.7.

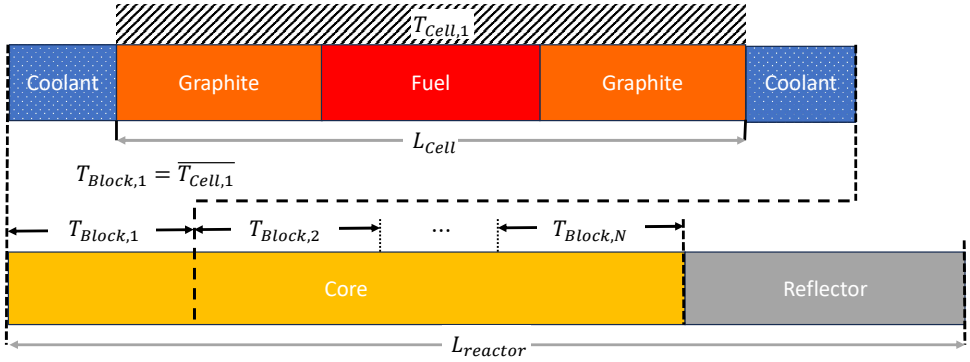


Figure 3.7: In this figure the mapping between the temperature and the neutronics mesh is shown, specifically for the first fuel block. In the temperature mesh the temperature per control volume in T_{cell} is determined. The determined area is the dashed part above the temperature mesh. We take the average of this area, $T_{Cell,1}$, and apply this temperature $T_{Block,1}$ to the control volumes comprising the corresponding block, $Block_1$, in the neutronics mesh. Cross-sections are altered due to this temperature, coupling temperature and neutronics

For neutronics to temperature we take a different approach, because the fuel block is not homogenized. Heat is only added through fission in the fuel channels.

1. We determine which control volumes in the neutronics mesh correspond to fuel channel in the temperature mesh, in Figure 3.8 we denote these areas in red with 'F'.
2. Per block, we add the fluxes in those control volumes comprising the 'F' area together, and determine the power per fuel channel per block
3. We insert the determined power in the fuel channels of the block

This method is visualised in Figure 3.8.

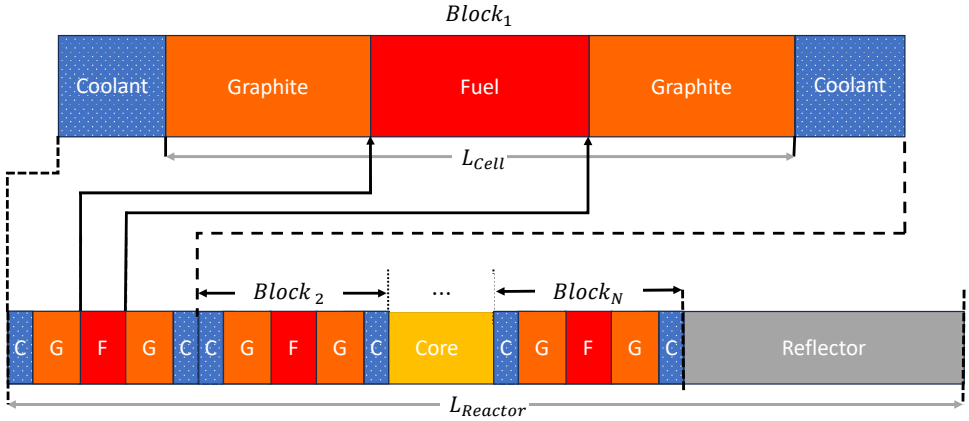


Figure 3.8: In this figure the mapping between the neutronics and the temperature mesh is shown, specifically for the first fuel block. After the flux in the entire reactor is determined, we take the fluxes at those control volumes corresponding to the fuel channel within every block. The fluxes of these control volumes are added together and converted to power. This power is then inserted in the control volumes comprising the fuel channel of the temperature mesh, coupling neutronics and temperature

k_{eff} routine

In Section 2.4 we said that solving for the neutronics in a steady-state only considers the ratios of fluxes. For an actual steady-state reactor, we need to take the temperature into account. We introduce two parameters: the change in multiplication factor due to temperature, $\frac{dk}{dT}$, and the change in temperature due to power, $\frac{dP}{dT}$. With these two parameters, we can iteratively determine which power and temperature, is required in the reactor to reach a steady-state. For this purpose we devise,

$$P^{n+1} = R\frac{dP}{dT}\left(\left(k_{eff,desired} - k_{eff}^{n}\right)\frac{1}{dk/dT} + P^{n}\right). \tag{3.16}$$

In this equation $k_{desired}$ is the multiplication factor we desire, usually equal to 1. k_{eff}^{n} is the multiplication factor at a previous iteration step. We determine the difference between this multiplication factor and the desired one, after which through $\frac{dk}{dT}$ we determine which change in temperature is necessary to achieve that. With $\frac{dP}{dT}$ the power we shall give the system in the next step is determined. Included is a relaxation factor R, to not overshoot the desired k_{eff}, and aid in its convergence.

Power and neutron fluxes are linked through fission,

$$P(x) = \Sigma_{f_1} w_1 \phi_1(x) + \Sigma_{f_2} w_2 \phi_2(x). \tag{3.17}$$

Where $w_{1,2}$ is the energy per fission event for respectively the fast and thermal fluxes.

We solve $\frac{dk}{dT}$ by determining k_{eff} for different temperatures. We apply a `Curve_fit` routine [40] for the function $k_{eff}(T) = \frac{dk}{dT}T + a$, where a is an arbitrary constant, which fits our data using a non-linear least-squares method, thus determining the dependence of k_{eff} on the temperature.

To solve for $\frac{dP}{dT}$ we determine the temperature for a number of different power inputs by using the method we developed to determine the temperature for a given input, after which we apply the same method as with $\frac{dk}{dT}$.

With these parameters we can determine the true steady-state neutron fluxes and temperatures, through equation 3.18 and setting $k_{eff,desired} = 1$. We iteratively solve the individual temperature and neutronics equations, and map between the two as shown in the previous section. For this we use the steady-state solvers for temperature and neutronics,

$$\begin{aligned}
\mathbf{Q}^{n} &= \Sigma_{f_1} w_1 \boldsymbol{\phi}_1^{n} + \Sigma_{f_2} w_2 \boldsymbol{\phi}_2^{n}, \\
\underline{\mathbf{A}}\mathbf{T}^{n+1} &= \mathbf{Q}^{n}, \\
\underline{\mathbf{L_1}}(\mathbf{T}^{n+1})\tilde{\boldsymbol{\phi}}_1^{n+1} &= \underline{\mathbf{S_1}}^{n}(\mathbf{T}^{n+1}), \\
\underline{\mathbf{L_2}}(\mathbf{T}^{n+1})\tilde{\boldsymbol{\phi}}_2^{n+1} &= \Sigma_{s_{12}}(\mathbf{T}^{n+1})\tilde{\boldsymbol{\phi}}_1^{n+1} + \frac{1}{v_2 \Delta t}\tilde{\boldsymbol{\phi}}_2^{n}, \\
F &= P^{n} / \int \left(\Sigma_{f_1} w_1 \tilde{\boldsymbol{\phi}}_1^{n+1} + \Sigma_{f_2} w_2 \tilde{\boldsymbol{\phi}}_2^{n+1}\right), \\
\boldsymbol{\phi}_{1,2}^{n+1} &= F\tilde{\boldsymbol{\phi}}_{1,2}^{n+1}, \\
P^{n+1} &= R\frac{dP}{dT}\left(\left(k_{eff,desired} - k_{eff}^{n}\right)\frac{1}{dk/dT} + P^{n}\right).
\end{aligned} \tag{3.18}$$

We map power density retrieved from fluxes in the previous iteration to our temperature code. We then determine the temperature associated with that power density. We use this temperature to alter the cross-sections and solve for the associated fluxes at that temperature $\tilde{\boldsymbol{\phi}}_{1,2}$. The size of this flux is still arbitrary, so we introduce a scaling F, by dividing the power density from the previous iteration by the power density arriving from the fluxes we had determined. By doing this scaling, we ensure that the power density in the next iteration corresponds to the right fluxes. After scaling the fluxes we get the actual fluxes $\boldsymbol{\phi}_{1,2}$.

With this scheme we can determine the temperature and fluxes for any $k_{eff,desired}$ that we want.

General transient code

The general transient code closely resembles equation 3.18. Now however, we use the time-dependent schemes for the temperature and fluxes. Furthermore, we are no longer interested in finding a power corresponding to a $k_{eff,desired}$. In practice we start from a steady-state with: $\mathbf{T}^0, \boldsymbol{\phi}_1^0, \boldsymbol{\phi}_2^0$ and $\boldsymbol{C}^0$. From this steady-state we start time stepping. We look at the behaviour of the reactor after a change in one of the parameters, to see how the reactor reacts. For this we use,

$$
\begin{aligned}
\mathbf{Q}^n &= \Sigma_{f_1} w_1 \boldsymbol{\phi}_1^n, \Sigma_{f_2} w_2 \boldsymbol{\phi}_2^n, \\
\underline{\mathbf{A}}\mathbf{T}^{n+1} &= \rho c_p \mathbf{T}^n + \Delta t \mathbf{Q}^n, \\
\underline{\mathbf{L_1}}(\mathbf{T}^{n+1})\boldsymbol{\phi}_1^{n+1} &= \underline{\mathbf{S_1}}^n(\mathbf{T}^{n+1}), \\
\underline{\mathbf{L_2}}(\mathbf{T}^{n+1})\boldsymbol{\phi}_2^{n+1} &= \Sigma_{s_{12}}(\mathbf{T}^{n+1})\boldsymbol{\phi}_1^{n+1} + \frac{1}{v_2 \Delta t}\boldsymbol{\phi}_2^n, \\
\mathbf{C}^{n+1} &= \frac{1}{\Delta t}\gamma C^n + \beta\gamma\left(\nu_1 \Sigma_f^1(\mathbf{T}^{n+1})\boldsymbol{\phi}_1^{n+1} + \nu_2 \Sigma_f^2(\mathbf{T}^{n+1})\boldsymbol{\phi}_2^{n+1}\right), \\
\mathbf{P}^{n+1} &= \Sigma_{f_1} w_1 \boldsymbol{\phi}_1^{n+1} + \Sigma_{f_2} w_2 \boldsymbol{\phi}_2^{n+1}.
\end{aligned}
\tag{3.19}
$$

We can solve for any transient. This scheme gives the temperature, power, fluxes, and precursor population for all x an t.

Chapter 4

Reduced Order Model

Neutronics and thermo-dynamics are dynamical systems, mathematically represented with PDEs. In nuclear physics we want to simulate these dynamical system to test reactor designs. These tests are to assess the safety of the reactor in risk and reliability assessments. To simulate the dynamical system we require numerical implementations of the dynamical systems. Numerical solutions to PDEs in complex geometry require discretization in space and time, where the accuracy of the solution is a function of the step-size in both space and time [28]. In order to create a high-fidelity model these time- or space-steps need to be sufficiently small, resulting in large computational costs.

This makes these so-called full order models unwieldy, especially in iterating novel designs or in computing statistical errors in for example cross-section data [39]. To alleviate this we want to create a reduced order model (ROM).
In this chapter we shall introduce the mathematical concept of reduced order modelling. Furthermore we shall discuss the different methods of creating a ROM and provide the method that was chosen in this research, Operator Inference (OpInf). Special attention shall be given to the mathematical framework of OpInf.

4.1 Reduced Order Modelling

Reduced order modelling is a mathematical method, in which we go from a high dimensional space (in the FOM) to a low dimensional space (in the ROM), losing as little accuracy as possible in the process [16]. For example, we can go from a system with 10^5 degrees of freedom, to just 10 [14].
Because the resulting ROM is of a lower dimensionality, it can solve the dynamical system in a fraction of the time, for example in [14] we go from 150CPU hours in the FOM to just 5s in the ROM. In this thesis we are interested in projection-based reduced order modelling, which allow for easy mapping between the high and low dimensionality.

Projection-based ROMs have two distinct steps [42]:

- Offline Step: Finding the low-dimensional representation of the FOM
- Online Step: Solving unknowns using the resulting ROM

We discern two types of ROM in the offline step: *intrusive* and *non-intrusive*. The *intrusive* type requires the full physical model, in the form of the exact operators of the governing equations, or in the form of direct access to the FOM and the ability to manipulate the FOM. *Non-intrusive* on the other hand is data-driven. It only requires access to output data (trajectories) to construct a ROM.

Of these two types, *non-intrusive* has our preference. This type does not require governing equations, making it especially useful if these governing equations are not known or hard to manipulate. This is the case when we want to apply the ROM to high-fidelity codes that are inaccessible or too large to manipulate, such as SCALE [4].

In this research we want to construct a non-intrusive predictive ROM. In literature review we found numerous ROM methods, such as applying POD to a system [31] [36], these methods though predictive are intrusive. Alternatively we found (partitioned) DMD [25] [21] to be non-intrusive, but lacking in its predictive capabilities. We found two possible predictive and non-intrusive methods, Operator Inference by Willcox and Peherstofer [35] and Sparse identification of Non-linear Dynamics by Kutz [27]. Of these two Operator Inference has been implemented. In the following sections the mathematical framework of OpInf will be presented, after which the implementation of OpInf to our representative model will be shown.

4.2 Proper Orthogonal Decomposition

OpInf uses the Proper Orthogonal Decomposition (POD) as a mathematical framework to construct a low dimensional subspace of the underlying dynamics. An in-depth visual of POD is given in [23]. Our FOM produces specific trajectories for specific inputs. For example, in our temperature scheme we retrieve trajectories for different types of heat input, these trajectories are determined with m control volumes in space. In POD we take k snapshots of these trajectories in time and store them in a snapshot matrix, $\mathbf{X} \in \mathbb{R}^{k \times m}$. $\mathbf{X}$ thus contains the snapshots of the trajectories and their dynamical evolving in time. For clarity we have chosen to represent matrices without the underline in the following section.

On the snapshot matrix we preform the Singular Value Decomposition (SVD),

$$\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^*, \tag{4.1}$$

SVD is the factorization of a matrix into three other special matrices [16]: $\mathbf{U}$, a unitary matrix, $\mathbf{\Sigma}$, a diagonal with positive real numbers, and $\mathbf{V}^*$ the complex conjugate another unitary matrix $\mathbf{V}$. This decomposition can be visualised as in Figure 4.1.

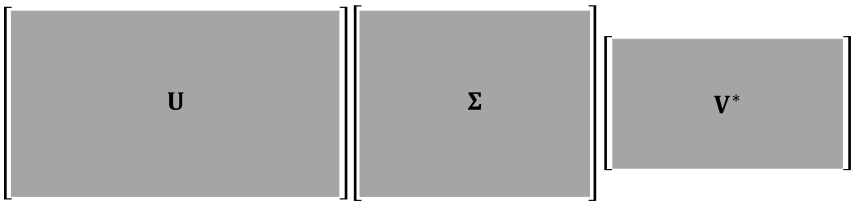


Figure 4.1: A visual representation of the three matrices of the SVD. On the left the unitary matrix $\mathbf{U}$*, in the middle the diagonal* $\mathbf{\Sigma}$*, and on the right the complex conjugate of the other unitary matrix* $\mathbf{V}$

The matrices $\mathbf{U}$, $\mathbf{\Sigma}$, $\mathbf{V}^*$ are called the left singular, singular, and right singular matrices respectively.

The entries of $\mathbf{\Sigma}$ are called the singular values of $\mathbf{X}$, and are unique to $\mathbf{X}$. Furthermore the columns of $\mathbf{U}$ and $\mathbf{V}^*$ are orthonormal. The columns of $\mathbf{U}$ correspond to the spatial dynamics of the trajectories in $\mathbf{X}$ and the rows of $\boldsymbol{V}^*$ corresponds to the time dynamics of the trajectories [16].

If the snapshot matrix $\mathbf{X}$ has shape $k \geq m$, than the matrix $\mathbf{\Sigma}$ has at most m non-zero elements. Thus we can economize the SVD by reducing the $\mathbf{U}$ and $\mathbf{\Sigma}$ matrices, disregarding the rows of $\mathbf{\Sigma}$ that are zero, and the columns of $\mathbf{U}$ that would have multiplied with those zero elements, as visualised in Figure 4.2.

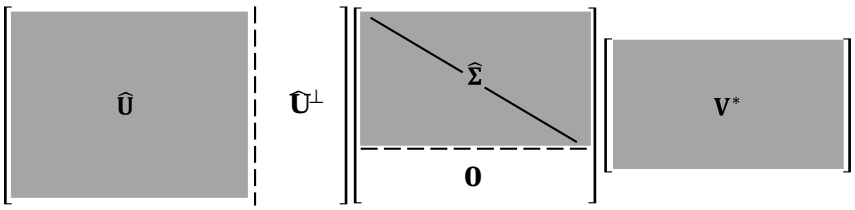


Figure 4.2: The economization of the SVD. We take away the rows containing only zeros of the diagonal matrix $\mathbf{\Sigma}$*, resulting in* $\hat{\mathbf{\Sigma}}$*. This means we also can reduce the amount of columns in the unitary matrix* $\mathbf{U}$*, giving* $\hat{\mathbf{U}}$*. The columns of* $\mathbf{U}$ *we do not use are denoted as* $\mathbf{U}^{\perp}$*. The* $\mathbf{V}^*$ *matrix remains unchanged.*

Columns of $\mathbf{U}$ (the modi of the system) that hold the most important spatial dynamics of the trajectories are those that are multiplied with the highest singular values. The time progression of those trajectories is held within the rows of $\mathbf{V}^*$ that are multiplied with the same singular value. The idea in ROM is to only select r columns (or modi) that hold the most important trajectories, discarding the rest. We do this by reducing $\mathbf{\Sigma}$, keeping only the r highest singular values, and subsequently reducing $\mathbf{U}$, keeping only r columns and in the same way $\mathbf{V}^*$, keeping only r rows. This decreases the size of the SVD matrices, while keeping the most important spatial dynamics of the trajectories. The resulting matrices are known as truncated matrices. Figure 4.3 gives a visualisation of this process.

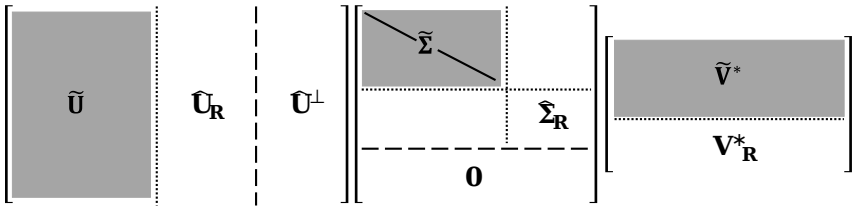


Figure 4.3: This figure shows the so-called truncated SVD, where only the first r modi corresponding to the most important dynamics of the system are left over. This results in $\tilde{\mathbf{U}}$, which only has r columns, $\tilde{\mathbf{\Sigma}}$ holding r singular values, and $\tilde{\mathbf{V}}^*$ having only r rows.

We now have the SVD describing the trajectories of the snapshot matrix,

$$\mathbf{X} = \mathbf{U\Sigma V} = \hat{\mathbf{U}}\hat{\mathbf{\Sigma}}\hat{\mathbf{V}} \approx \tilde{\mathbf{U}}\tilde{\mathbf{\Sigma}}\tilde{\mathbf{V}}^*, \tag{4.2}$$

We use $\tilde{\mathbf{U}}$ to project any state into a lower dimensional subspace, $\tilde{\mathbf{x}} = \tilde{\mathbf{U}}^\intercal \mathbf{x}$.

To illustrate how the POD can be used, we show its intrusive use with Galerkin projection (from [16]). We take an arbitrary dynamical system defined as,

$$\frac{d}{dt}\mathbf{x}(t) = \mathbf{A}\mathbf{x}(t). \tag{4.3}$$

With $\mathbf{A}$ an operator acting upon the states $\mathbf{x}$, which change in time t. From this system we have learned the $\tilde{\mathbf{U}}$ matrix through a snapshot matrix. If we know the operator $\mathbf{A}$ of equation 4.3, we can reduce the operator $\mathbf{A}$: $\tilde{\mathbf{A}} = \tilde{\mathbf{U}}^\intercal \mathbf{A}\tilde{\mathbf{U}}$. We can use this reduced matrix to determine for any initial state $\mathbf{x}(t=0)$ the progression of that state in time. We reduce the order of this initial condition, and apply the reduced operator to the reduced initial condition,

$$\tilde{\mathbf{x}} = \tilde{\mathbf{U}}^\intercal \mathbf{x},$$
$$\frac{d}{dt}\tilde{\mathbf{x}}(t) = \tilde{\mathbf{A}}\tilde{\mathbf{x}}(t),$$

to step in time in the reduced space. Once finished we can transform back with $\mathbf{x} = \tilde{\mathbf{U}}\tilde{\mathbf{x}}$.

Applying POD in such a manner works, but is intrusive, as we need to have the operator $\mathbf{A}$ to reduce it. OpInf from Willcox et al. [35] uses a different way to achieve a ROM using the POD.

4.3 Operator Inference

OpInf does not work with the direct operators of a dynamical system, it is non-intrusive. However, some knowledge about the underlying physics is necessary. We need to know if the system is linear, quadratic, or cubic. OpInf can work with a system up to a cubic relation, anything beyond that and the resulting reduced model becomes too large, making reduction redundant.

For example, take the heat equation from chapter 2,

$$\frac{\partial}{\partial t}T(x,t) = \lambda \frac{\partial^2}{\partial x^2}T(x,t) + Q(x,t), \tag{4.4}$$

where T is the temperature in K, λ is the thermal conductivity in $\frac{\mathrm{W}}{\mathrm{cmK}}$, and Q the power put into the system in W. This system can be described as,

$$\frac{d}{dt}\mathbf{T}(t;\lambda) = \mathbf{A}(\lambda)\mathbf{T}(t;\lambda) + \mathbf{B}\mathbf{Q}(t). \tag{4.5}$$

Here $\mathbf{T} \in \mathbb{R}^n$ is the temperature with n the amount of control volumes in space. In this equation the spatial derivative $\frac{\partial^2}{\partial x^2}$ is described in the operator $\mathbf{A}$ with parameter λ, the added heat to the system is described through $\mathbf{BQ}$. This system has a linear term of $\mathbf{T}$. The only information we need to accurately infer the operators is that the term is linear.

The goal of OpInf is to infer a reduced operator, instead of retrieving a reduced operator by projecting the SVD matrix $\tilde{\mathbf{U}}$ on a known operator, as in the previously described Galerkin projection. In OpInf we first construct a POD basis from snapshots of the system. With this POD basis we can compress the snapshots $\mathbf{T}$, by multiplying it with $\tilde{\mathbf{U}}^\intercal$,

$$\mathbf{T}_{compressed} = \tilde{\mathbf{U}}^\intercal \mathbf{T} = \tilde{\mathbf{T}} \in \mathbb{R}^r. \tag{4.6}$$

Where r correspond to the amount of modi we kept in the truncated matrix $\tilde{\mathbf{U}}$ and n the amount of control volumes in space. In OpInf we can use a number of different types of bases, however the POD base is advised, as it gives the least error in an l_2 norm [16]. Along with the compressed snapshots we need the compressed time derivatives of the snapshots. This is done by applying a BDF scheme, from Section 3.1.3, to the provided snapshots, giving us $\frac{d}{dt}\mathbf{T}$, which we compress in the same way as the snapshots.
We can now infer the operators of equation 4.5 through ,

$$\min_{\tilde{\mathbf{A}}\in\mathbb{R}^{r\times r},\tilde{\mathbf{B}}\in\mathbb{R}^{r\times n}} \left\| \frac{d}{dt}\tilde{\mathbf{T}} - \tilde{\mathbf{A}}\tilde{\mathbf{T}} - \tilde{\mathbf{B}}\mathbf{Q} \right\|^2. \tag{4.7}$$

From this minimization we get the reduced operators. Using these reduced operators we can, just as with the intrusive projection scheme, determine $\mathbf{T}(t)$ for any initial condition $\mathbf{T}(t=0)$ and power input $\mathbf{Q}(t)$, after we have successfully reduced the initial condition and input. To map back to the original dimensionality, we multiply $\tilde{\mathbf{U}}\tilde{\mathbf{T}}(t)$ to get $\mathbf{T}(t)$.

If the system is quadratic or cubic in nature, the description of the system of equation 4.5 changes. We add operators for the quadratic or cubic dynamics. Subsequently the minimization problem is also extended to include those operators.

It should be noted that OpInf does not require equidistant Δt in its state vectors nor its derivatives. If this is not the case, we use the discrete Operator Inference, with which we can have non-equidistant Δt. In this research we have worked with the continuous variant, which does have equidistant Δt.

4.4 Reduced Order Model implementation

A specific Python package has been made for OpInf [3]. This Python package provides a number of functions such as: preprocessing data, inferring reduced operators, and predicting future states using those inferred operators. The following section describes the implementation of OpInf in our representative model. First we discuss the necessary preprocessing of the training data. Next we show the use of parameters in OpInf. Following this we explain the application of OpInf to the time-dependent temperature scheme. After which we show the application of OpInf to the time-dependent neutronics scheme. Finally we discuss the application of OpInf to the coupled system.

4.4.1 Preprocessing

Training data of OpInf often needs to be preprocessed, because the functions within OpInf are optimized for data within $[-1, 1]$. Furthermore by preprocessing our data we reduce the likelihood that OpInf over- or underfits its operators. For example in POD we reduce the system to the most important dynamics. A change in neutron flux from 10^{15} to 10^{16} might dominate these dynamics compared to a change in temperature from 600 to 800K, whereas we need to capture both dynamics. This preprocessing often entails shifting and scaling of the training data [3]. The exact extent and degree of preprocessing depends on the specific problem set. Preprocessing the data provides significantly improved results [26], [22]. In this work we preprocess our training data by scaling it.

We determine what the steady-state solutions are for fluxes, precursor concentration and the temperature profile. When training a ROM, we scale the training states with the maximum value of the variable's steady-state. So $\boldsymbol{\phi}_{f,train,} = \boldsymbol{\phi}_{f,train} / max\left(\boldsymbol{\phi}_{f,steady-state}\right)$. The operators are thus learned for the scaled training states. If we want to use the ROM we also have to scale the initial conditions with the same factor. After the ROM has finished time stepping, we descale using the same factor as in scaling, $\boldsymbol{\phi}_f(t_{final}) = \boldsymbol{\phi}_{f,ROM}(t_{final}) \cdot max\left(\boldsymbol{\phi}_{f,steady-state}\right)$

4.4.2 Parameters in OpInf

In equation 4.5 we showed the application of OpInf on the thermal diffusion equation, with a parameter λ. In OpInf such a parameter is denoted as μ. We can train and use parameters in two distinct ways: if $\mu \in \mathbb{R}^1$ we use a 1 dimensional interpolation, and if $\boldsymbol{\mu} \in \mathbb{R}^p$ we use a p dimensional interpolation. As an example we take a system characterized by,

$$\frac{d}{dt}\mathbf{X}(t;\boldsymbol{\mu}) = \mathbf{A}(\boldsymbol{\mu})\mathbf{X}(t;\boldsymbol{\mu}). \tag{4.8}$$

Where $\boldsymbol{\mu} \in \mathbb{R}^p$ is our parameter vector. To explain the interpolation technique, we look at the influence of temperature on neutron flux in a reactor. In principle the parameter vector can denote anything we want. For example fast neutron yield (ν_1) or the absorption cross-section for the thermal group ($\Sigma_{a,2}$).

1 dimensional interpolation

At first we look at the case where we have a homogeneous temperature throughout the core, here $p = 1$ and thus $\mu = T \in \mathbb{R}$, with T the homogeneous temperature in our reactor. Equation 4.8 reduces to,

$$\frac{d}{dt}\mathbf{X}(t;\mu) = \mathbf{A}(\mu)\mathbf{X}(t;\mu).$$

We run our FOM with k snapshots and m control volumes in space. We do this for n different runs with different temperatures μ_i, resulting in n training runs, which we collect in a training set. This training set is used to train our ROM. The training set containing $\mathbf{X}_{\mathbf{train}} \in \mathbb{R}^{k \times nm}$ and $\frac{d}{dt}\mathbf{X}_{\mathbf{train}} \in \mathbb{R}^{k \times nm}$. With this training set we infer n operators, $\mathbf{A}_1, \mathbf{A}_2, ..., \mathbf{A}_n$, where operator $\mathbf{A}_i$ corresponds to training run $\mathbf{X}_{\mathbf{train}i}$ and μ_i. When we want to estimate an operator for a temperature not included in the training set, we interpolate between the individual elements of the operators, $\mathbf{A}_i$. To create an interpolated operator $\mathbf{A}_{Interpolated}$ OpInf uses as a standard Cubic interpolation (through `scipy.interpolate.CubicSpline` [5]) for this operation. Figure 4.4 gives a graphical representation of this `CubicSpline` of the elements within the operator. A separate spline is made for every element within the operator matrices. So if $\mathbf{A} \in \mathbb{R}^{2\times 2}$ there would need to be 4 splines. With the resulting interpolated operator we can apply the same techniques as in the temperature ROM, stepping forward in time using BDF. It should be noted that OpInf can use any interpolation technique, such as a linear interpolator.

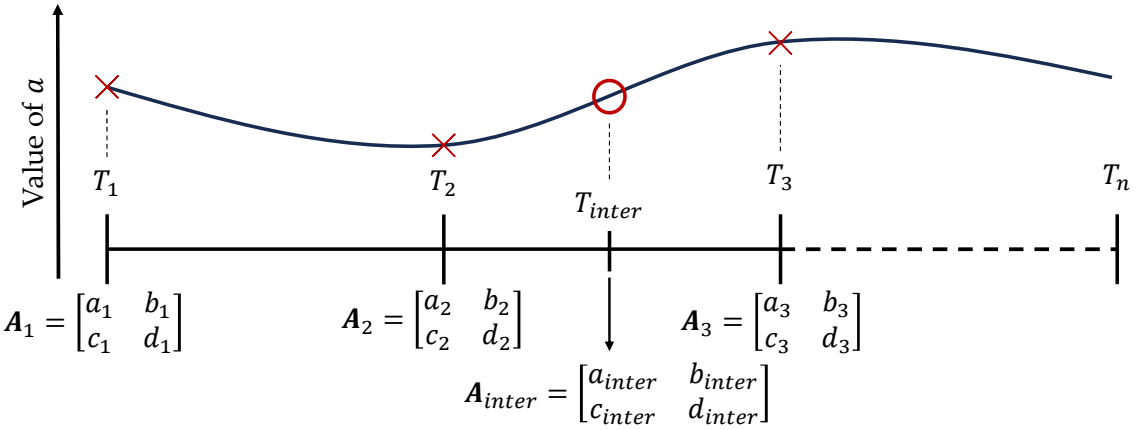


Figure 4.4: A graphical representation of the spline method in 1 dimension. For n different temperatures T_i an associated operator matrix $\mathbf{A}_i$ is inferred. With these matrices we calculate a spline per entry. Here we show an example spline for the first entry a_i. If we want to determine an operator at a temperature not included in training, we use these splines to determine the values of the entries at that temperature, here we for example determine the value of a_{inter}.

p dimensional interpolation

We return to equation 4.8. We now consider $\boldsymbol{\mu} \in \mathbb{R}^2$, noting that the following method works for p dimensional interpolation. The principle with which we use OpInf is similar to the homogeneous case. However, instead of interpolating using a cubic spline between the entries of the associated operators OpInf uses as a standard linear interpolation (through the `scipy.interpolate.LinearNDInterpolator` class [6]) to interpolate in N dimensions. Other interpolation techniques, such as cubic or nearest neighbour, are possible within the OpInf environment. Each method has its own pros and cons, in this thesis we look at cubic and linear interpolation, and determine which works best for our case. To show how OpInf interpolates we take a look at Figure 4.5. In the case that $\boldsymbol{\mu} \in \mathbb{R}^2$, we divide the reactor in two temperature zones, such that $\boldsymbol{\mu} = [T_1, T_2]$, shown in Figure 4.5. We look at the values of the first element a_i of the inferred operators. We infer operators for n training runs with different (T_1, T_2), shown in Figure 4.5 with black dots. If we want to determine an operator for a different temperature, shown as a red-cross, we use `LinearNDInterpolator`, which constructs so-called Delauney triangles, shown in Figure 4.5 with red dashed lines. The value at each point within this triangle can be interpolated through linear barycentric interpolation from the values at the edges, $a_{inter} = \lambda_{1,1}a_{1,1} + \lambda_{1,2}a_{1,2} + \lambda_{2,2}a_{2,2}$, where λ_i are weights assigned to the coordinates in the triangle, with $\Sigma\lambda_i = 1$. For example if we interpolate for the coordinates of point $a_{1,1}$ than $\lambda_{1,1} = 1$, and the others 0. Such a 2D space is thus constructed for every element entry of the operators. So if $\mathbf{A} \in \mathbb{R}^{2\times 2}$ there would need to be 4 interpolation planes.

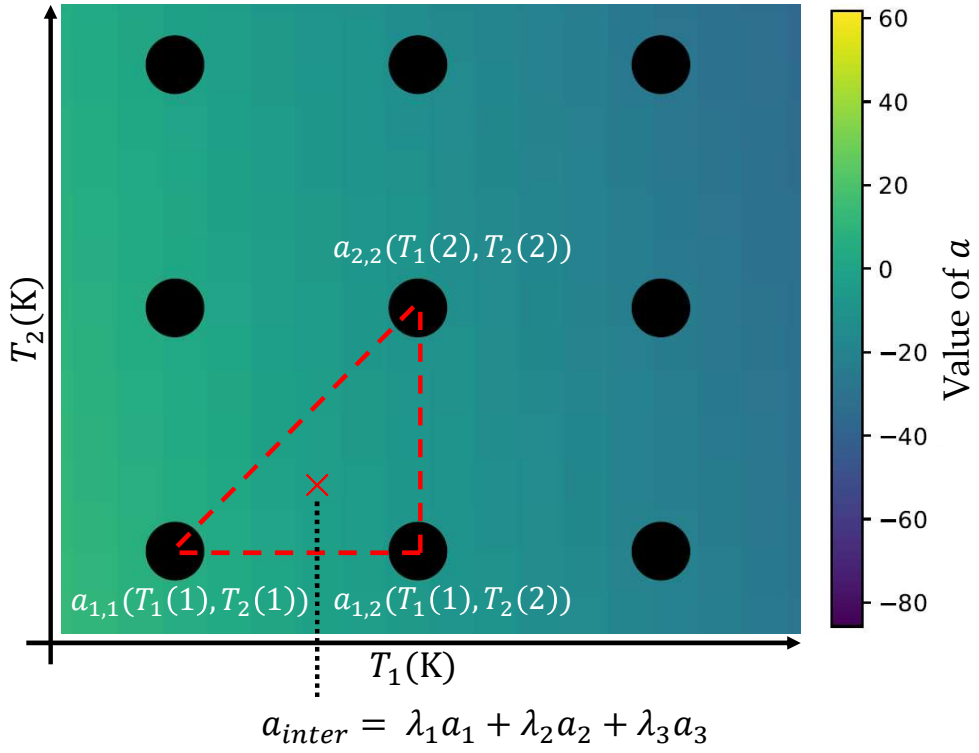


Figure 4.5: Visualisation of OpInf interpolation with two temperature zones. We train with n different temperature combinations (T_1, T_2). For every training temperature coordinate $T_1(i), T_2(j)$ an associated operator matrix $\mathbf{A}_{i,j}$ is inferred with the top left element called $a_{i,j}$. For every entry of this matrix a 2 dimensional grid is made, like the one shown in this figure, which would be for the upper left element. If we want to determine an operator at a temperature not included in training `LinearNDInterpolator` performs a linear barycentric interpolation on a constructed Delauney triangle from points $a_{1,1}, a_{1,2}$ and $a_{2,2}$. Through this interpolation the value of any point within the triangle a_{inter} can be determined through $a_{inter} = \lambda_{1,1}a_{1,1} + \lambda_{1,2}a_{1,2} + \lambda_{2,2}a_{2,2}$, provided that $\Sigma_i\lambda_i = 1$.

In this example we chose $\boldsymbol{\mu}$ to be two temperature zones in the reactor. We can choose $\boldsymbol{\mu}$ to hold many different sorts of parameters. For example, we can define variable 5 temperature zones and define two regions with variable fast neutron yield (ν_f), this would result in $\boldsymbol{\mu} \in \mathbb{R}^7$.

4.4.3 Implementing OpInf for temperature

Recall the heat equation (equation 2.6),

$$\rho c_p \frac{\partial}{\partial t} T(x,t) = \lambda \frac{\partial^2}{\partial x^2} T(x,t) + Q(x,t). \tag{4.9}$$

In OpInf we need to describe this PDE in a matrix formulation. To see how this applies, we rewrite equation 4.9 by dividing by ρc_p,

$$\frac{\partial}{\partial t} T(x,t) = \frac{\lambda}{\rho c_p} \frac{\partial^2}{\partial x^2} T(x,t) + \frac{1}{\rho c_p} Q(x,t). \tag{4.10}$$

From expression 4.10 we see that the matrix implementation becomes,

$$\frac{d}{dt}\mathbf{T}(t;\boldsymbol{\mu}) = \mathbf{A}(\boldsymbol{\mu})\mathbf{T}(t;\boldsymbol{\mu}) + \mathbf{B}(\boldsymbol{\mu})\mathbf{Q}(t). \tag{4.11}$$

This expression contains the parameter $\boldsymbol{\mu}$. This parameter contains the different parameters in the heat equation ρ, c_p, λ. In our implementation we are concerned in what way the acquired ROM can predict effects of different heat sources $\mathbf{Q}(t)$ and not the direct effect of these parameters in the heat equation. For our temperature ROM we take the heat equation parameters to be constant. As such we are interested in a slightly altered form of equation 4.11,

$$\frac{d}{dt}\mathbf{T}(t) = \mathbf{A}\mathbf{T}(t) + \mathbf{B}\mathbf{Q}(t). \tag{4.12}$$

In this expression the state vector $\mathbf{T}(t)$ contains the temperature of each control volume in the temperature mesh. Using this expression we can train a ROM using OpInf. To do this we use the packages provided by Willcox et al. [3].

With our representative model we create n different training cases with different heat sources. For every case this gives us $\mathbf{T_i} \in \mathbb{R}^{k\times m}$ and$\mathbf{Q_i} \in \mathbb{R}^{k\times m}$, with k amount of snapshots and m amount of control volumes in the FOM space. We preprocess the $\mathbf{T_i}$ state matrices by dividing by the steady-state temperature profile. From the preprocessed temperature profile we determine $\frac{d}{dt}\mathbf{T_i} \in \mathbb{R}^{k\times m}$ through first order BDF. We combine the n different preprocessed training runs in a single training set $\mathbf{T_{train}} \in \mathbb{R}^{k\times nm}$, $\mathbf{Q_{train}} \in \mathbb{R}^{k\times nm}$ and the different time derivatives $\frac{d}{dt}\mathbf{T_{train}} \in \mathbb{R}^{k\times nm}$,

$$\begin{aligned}
\mathbf{T_{train}} &= [\mathbf{T_1}, \mathbf{T_2}, ..., \mathbf{T_n}]^\mathsf{T}, \\
\mathbf{Q_{train}} &= [\mathbf{Q_1}, \mathbf{Q_2}, ..., \mathbf{Q_n}]^\mathsf{T}, \\
\frac{d}{dt}\mathbf{T_{train}} &= \left[\frac{d}{dt}\mathbf{T_1}, \frac{d}{dt}\mathbf{T_2}, ..., \frac{d}{dt}\mathbf{T_n}\right]^\mathsf{T}.
\end{aligned}$$

Note that the input vectors $\mathbf{Q}_{train}$ are not preprocessed. From the combined $\mathbf{T_{train}}$ state matrix we extract the POD basis. Within the OpInf environment this is done through `basis.PODbasis().fit()` function. We also tell OpInf how many modes of the POD base we want to keep. It is possible at this stage to assess how many modes are necessary, by plotting the decay of the singular values of the $\mathbf{\Sigma}$ matrix of equation 4.1. However, this only gives a rough estimate as the values of the singular values are dependent on the training, whereas the quality of our ROM is determined in test cases outside of the training data. Therefore the actual necessary amount of modes for an accurate ROM is assessed later, by comparing it to a FOM.

With the reduced base we can create a ROM. We first instantiate the ROM we want to make with OpInf. It is at this stage that we tell what sort of model we use, continuous if there is a constant Δt, or discrete if we want to work with varying Δt. Furthermore we provide OpInf with the types of operators we have (constant, linear, quadratic, cubic, and whether we have an input). With this knowledge OpInf can solve the minimization problem of equation 4.7, as long as we provide the reduced training matrix $\tilde{\mathbf{T}}_{\mathbf{train}}$, time-derivative matrix $\frac{d}{dt}\tilde{\mathbf{T}}_{\mathbf{train}}$, and the full input matrix $\mathbf{Q_{train}}$.

From this training OpInf extracts the two inferred operators $\tilde{\mathbf{A}}$ and $\tilde{\mathbf{B}}$. We can extract these inferred operators and use them manually in a time-stepping scheme, such as BDF. We can also use the inbuilt OpInf class of `predict` to do the same thing. `Predict` uses first order BDF as a standard, but can use any technique from [1].

4.4.4 Implementing OpInf for neutronics

For the implementation of OpInf in neutronics, recall equation 2.8,

$$\begin{aligned}
\frac{1}{v_1}\frac{\partial \boldsymbol{\phi_1}(t)}{\partial t} - D_1(\mathbf{T})\frac{\partial^2 \boldsymbol{\phi_1}(t)}{\partial x^2} + \Sigma_{a_1}(\mathbf{T})\boldsymbol{\phi_1}(t) + \Sigma_{s_{12}}(\mathbf{T})\boldsymbol{\phi_1}(t) &= (1-\beta)[\nu_1\Sigma_{f_1}(\mathbf{T})\boldsymbol{\phi_1}(t) + \nu_2\Sigma_{f_2}(\mathbf{T})\boldsymbol{\phi_2}(t)] \\
&\quad + \lambda \boldsymbol{C}(t), \\
\frac{1}{v_2}\frac{\partial \boldsymbol{\phi_2}(t)}{\partial t} - D_2(\mathbf{T})\frac{\partial^2 \boldsymbol{\phi_2}(t)}{\partial x^2} + \Sigma_{a_2}(\mathbf{T})\boldsymbol{\phi_2}(t) &= \Sigma_{s_{12}}(\mathbf{T})\boldsymbol{\phi_1}(t), \\
\frac{d\boldsymbol{C}(t)}{dt} &= -\lambda \boldsymbol{C}(t) + \beta[\nu_1\Sigma_{f_1}(\mathbf{T})\boldsymbol{\phi_1}(t) + \nu_2\Sigma_{f_2}(\mathbf{T})\boldsymbol{\phi_2}(t)].
\end{aligned} \tag{4.13}$$

Here the cross-sections are a function of temperature, which is the coupling between temperature and neutronics. As with the temperature, we rewrite the equation to get to an OpInf expression,

$$\begin{aligned}
\frac{\partial}{\partial t}\boldsymbol{\phi_1}(t) &= v_1\left[\left(D_1(\mathbf{T})\frac{\partial^2}{\partial x^2} - \Sigma_{a_1}(\mathbf{T}) - \Sigma_{s_{12}}(\mathbf{T}) + (1-\beta)\nu_1\Sigma_{f_1}(\mathbf{T})\right)\boldsymbol{\phi_1}(t) + \nu_2\Sigma_{f_2}(\mathbf{T})\boldsymbol{\phi_2}(t) + \lambda \boldsymbol{C}(t)\right], \\
\frac{\partial}{\partial t}\boldsymbol{\phi_2}(t) &= v_2\left[\left(D_2(\mathbf{T})\frac{\partial^2}{\partial x^2} + \Sigma_{a_2}(\mathbf{T})\right)\boldsymbol{\phi_2}(t) + \Sigma_{s_{12}}(\mathbf{T})\boldsymbol{\phi_1}(t)\right], \\
\frac{d}{dt}\boldsymbol{C}(t) &= -\lambda \boldsymbol{C}(t) + \beta\nu_1\Sigma_{f_1}(\mathbf{T})\boldsymbol{\phi_1}(t) + \beta\nu_2\Sigma_{f_2}(\mathbf{T})\boldsymbol{\phi_2}(t)].
\end{aligned} \tag{4.14}$$

Expression 4.14 can be rewritten as,

$$\frac{d}{dt}\mathbf{X}(t;\boldsymbol{\mu}) = \mathbf{A}(\boldsymbol{\mu})\mathbf{X}(t;\boldsymbol{\mu}). \tag{4.15}$$

In equation 4.15 the state vector $\mathbf{X}(t) \in \mathbb{R}^{3m}$ denotes the combined state vector of the fast and thermal fluxes, as well as the precursor population: $\mathbf{X}(t) = [\boldsymbol{\phi_1}(t), \boldsymbol{\phi_2}(t), \mathbf{C}(t)]^\intercal \in \mathbb{R}^{3m}$. In neutronics we work with the parameter $\boldsymbol{\mu}$, in this case $\boldsymbol{\mu}$ denotes the average temperature of p zones in the reactor, $\boldsymbol{\mu} \in \mathbb{R}^p$. If we treat the temperature as homogeneous throughout the core $p = 1$, if we define temperature in 10 parts in the reactor $p = 10$. We work with temperature as a parameter, because we are interested in the effect of temperature on neutronics.

4.4.5 ROM of coupled temperature and neutronics code

In this thesis we want to investigate whether we can create a predictive non-intrusive ROM of an HTGR. In this research we specifically look at whether this can be achieved for a coupled temperature and neutronics representative model. Two ways of creating a coupled ROM will be tested.

- a merged ROM (mROM): In this approach we merge the state vectors of temperature, fast and thermal fluxes, and precursor population in one single state vector and train a ROM from that state vector
- a sequential ROM (sROM): In this approach we train a separate ROM for the temperature ROM_T and for the neutronics ROM_N, from which we can construct a coupled ROM in the same manner as we coupled temperature and neutronics in the representative model, as given in Section 3.2.3

mROM

For the mROM the system of equations,

$$\begin{aligned}
\frac{1}{v_1}\frac{\partial \boldsymbol{\phi_1}(t)}{\partial t} - D_1(\mathbf{T})\frac{\partial^2 \boldsymbol{\phi_1}(t)}{\partial x^2} + \Sigma_{a_1}(\mathbf{T})\boldsymbol{\phi_1}(t) + \Sigma_{s_{12}}(\mathbf{T})\boldsymbol{\phi_1}(t) &= (1-\beta)[\nu_1\Sigma_{f_1}(\mathbf{T})\boldsymbol{\phi_1}(t) + \nu_2\Sigma_{f_2}(\mathbf{T})\boldsymbol{\phi_2}(t)] \\
&\quad + \lambda \boldsymbol{C}(t), \\
\frac{1}{v_2}\frac{\partial \boldsymbol{\phi_2}(t)}{\partial t} - D_2(\mathbf{T})\frac{\partial^2 \boldsymbol{\phi_2}(t)}{\partial x^2} + \Sigma_{a_2}(\mathbf{T})\boldsymbol{\phi_2}(t) &= \Sigma_{s_{12}}(\mathbf{T})\boldsymbol{\phi_1}(t), \\
\frac{d\boldsymbol{C}(t)}{dt} &= -\lambda \boldsymbol{C} + \beta[\nu_1\Sigma_{f_1}(\mathbf{T})\boldsymbol{\phi_1}(t) + \nu_2\Sigma_{f_2}(\mathbf{T})\boldsymbol{\phi_2}(t)], \\
\rho c_p \frac{\partial \boldsymbol{T}(t)}{\partial t} &= \lambda \frac{\partial^2 \boldsymbol{T}(t)}{\partial x^2} + (w_1\Sigma_{f_1}(\mathbf{T})\boldsymbol{\phi_1}(t) + w_2\Sigma_{f_2}(\mathbf{T})\boldsymbol{\phi_2}(t))
\end{aligned} \tag{4.16}$$

is a coupled system. The cross-sections are a linear function of temperature. This means that we are no longer dealing with a linear system, but with a quadratic system. Furthermore the diffusion coefficient is defined as $\frac{1}{3(\Sigma_a+\Sigma_s)}$, in principle a non-polynomial relation. To account for this we determine the relationship of $D(T)$ within the temperature range in our reactor, and use the resulting polynomial to substitute for the division. We found that for $T \in [294, 2500]$ K D can be approcimated as a linear function of T. For this quadratic system our matrix representation in OpInf is given as,

$$\frac{d}{dt}\mathbf{X}(t;\boldsymbol{\mu}) = \mathbf{A}(\boldsymbol{\mu})\mathbf{X}(t;\boldsymbol{\mu}) + \mathbf{F}(\boldsymbol{\mu})\mathbf{X}(t;\boldsymbol{\mu})^2. \tag{4.17}$$

Here $\mathbf{X}(t) = [\boldsymbol{\phi_1}(t), \boldsymbol{\phi_2}(t), \mathbf{C}(t), \mathbf{T}(t)]^\intercal \in \mathbb{R}^{3m_N+m_T}$, with m_N the amount of control volumes in the neutronics model and m_T the amount of control volumes in the temperature model, which are not equal as explained in Section 3.2.3. Furthermore $\boldsymbol{\mu}$ can be any sort of parameter, such as the fast neutron yield (ν_1).

Equation 4.17 uses the syntax as given in [35]. In this syntax $\mathbf{X}(t;\boldsymbol{\mu})^2$ is a combination of all the state vectors multiplied with themselves without repeating entries ($x_1x_2 = x_2x_1$, only 1 is included),

$$\boldsymbol{X}(t;\boldsymbol{\mu})^2 = \begin{bmatrix} \boldsymbol{X}^{(1)}(t;\boldsymbol{\mu}) \\ \boldsymbol{X}^{(2)}(t;\boldsymbol{\mu}) \\ \cdot \\ \boldsymbol{X}^{(N)}(t;\boldsymbol{\mu}) \end{bmatrix},$$

with,

$$\boldsymbol{X}^{(i)}(t;\boldsymbol{\mu}) = \boldsymbol{X}_i(t;\boldsymbol{\mu}) \begin{bmatrix} \boldsymbol{X}_1(t;\boldsymbol{\mu}) \\ \cdot \\ \boldsymbol{X}_i(t;\boldsymbol{\mu}) \end{bmatrix},$$

resulting in,

$$\boldsymbol{X}(t;\boldsymbol{\mu})^2 = \begin{bmatrix} \boldsymbol{X}_1(t;\boldsymbol{\mu})\boldsymbol{X}_1(t;\boldsymbol{\mu}) \\ \boldsymbol{X}_2(t;\boldsymbol{\mu})\boldsymbol{X}_1(t;\boldsymbol{\mu}) \\ \boldsymbol{X}_2(t;\boldsymbol{\mu})\boldsymbol{X}_2(t;\boldsymbol{\mu}) \\ \boldsymbol{X}_3(t;\boldsymbol{\mu})\boldsymbol{X}_1(t;\boldsymbol{\mu}) \\ \cdot \\ \boldsymbol{X}_N(t;\boldsymbol{\mu})\boldsymbol{X}_N(t;\boldsymbol{\mu}) \end{bmatrix},$$

such that $\mathbf{X}^2 \in \mathbb{R}^S; S = r(r+1)/2$, with r the amount of modi we want to keep.

With this expression we can train a ROM using the necessary state matrices, in the same manner as before.

sROM

In the sROM we couple the two ROMs from Sections 4.4.3 and 4.4.4. These ROMs use the same training as described there.

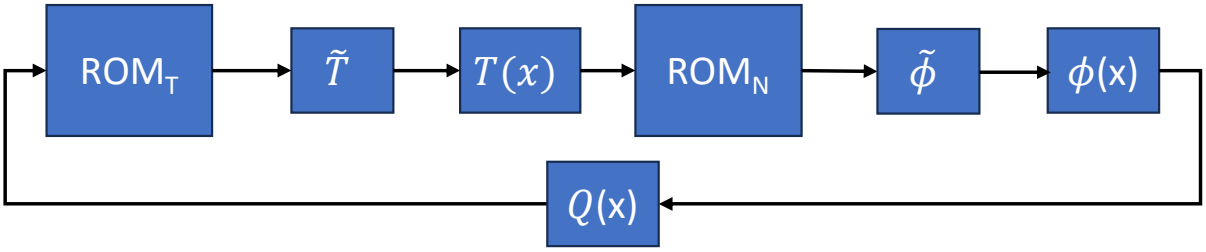


Figure 4.6: This figure shows the sROM flow. We create a ROM_T and a ROM_N. The results from these ROMs ($\tilde{\mathbf{T}}$,$\tilde{\boldsymbol{\phi}}$) are mapped back to their original dimensions ($T(x)$,$\phi(x)$), and are fed into the ROM_N and ROM_T respectively. Thus resolving temperature and neutronics.

Figure 4.6 shows the flow of the sROM, and in what way the ROM_N and ROM_T feed into each other. At every step we have to transform the results of the ROM back to the original dimension. Wwe feed the temperature as a parameter ($T(x)$) into the ROM_N. The ROM_N then estimates the operators associated with that temperature through interpolation in 1 dimension if the temperature is homogeneous and in p dimensions if we define p temperature zones in the reactor, as described in Section 4.4.2. With these parameters the ROM_N predicts the flux profiles. The resultant flux profile are mapped back to the original dimension. From these flux profiles, we extract the power and use the power as a spatial input ($Q(x)$), a heat source, in the ROM_T. The ROM_T than predicts the temperature from that power input, as described in Section 4.4.3. The resultant temperature is mapped back to the original space, and used as an input for the ROM_N. This is done once for every time step, just as in Section 3.2.3.

Chapter 5

Results

This chapter presents the results of this thesis. We first give the validation of the representative high-fidelity models, checking the convergence of the RMS error upon mesh refinement. Next we consider the accuracy of the ROMs created in this thesis (a temperature ROM (ROM_T), a neutronics ROM (ROM_N) and our coupled ROMs (mROM and sROM) compared to our representative model. Parameters used in tests are as given in appendix A.2 unless explicitly mentioned.

5.1 Validation of the representative high-fidelity model

The first part of this thesis was creating a representative high-fidelity model (FOM). This model has been validated using the expected RMS error convergence of the discretization methods of chapter 3, looking at convergences of the numerical methods to analytical solutions or known high-fidelity solutions.

We first look at the representative temperature model from Section 3.2.1, after which the neutronics model from Section 3.2.2 will be investigated. In the end we determine whether the coupled model created in Section 3.2.3 is accurate.

5.1.1 Representative temperature model

In this section we first discuss the spatial and temporal convergences of our representative model from Section 3.2.1.

Spatial

First we look at the steady-state temperature convergence to an analytical solution. For this we use the steady-state heat equation 3.3,

$$-\lambda \frac{\partial^2}{\partial x^2} T(x) = Q(x),$$

for which [13] provides the analytical solution as,

$$\begin{aligned} T(x) &= T_0 + \frac{Q(x)L}{\lambda} x && x \in (0, L) \\ T(x) &= T_0 + \frac{2Q(x)L}{\lambda} x - \frac{Q(x)}{2\lambda}\left(x^2 + L^2\right) && x \in (L, 2L). \end{aligned} \tag{5.1}$$

This solution gives the temperature $T(x)$ in K, for a bar of length $2L$ in which the right half produces heat, as stored in $Q(x)$, with the heat production in W. is The right boundary is subject to a homogeneous Neumann boundary condition, from equation 3.6, while the left boundary is subject to a mixed boundary condition with T_0 the reference temperature, as in equation 3.7.
We implement this using the necessary boundary conditions as explained in chapter 3. Figure 5.1

shows how our numerical solution approaches the analytical solution, with Figure 5.2 showing the RMS error as a function of Δx between the analytical and numerical solutions, as calculated with equation 3.2.

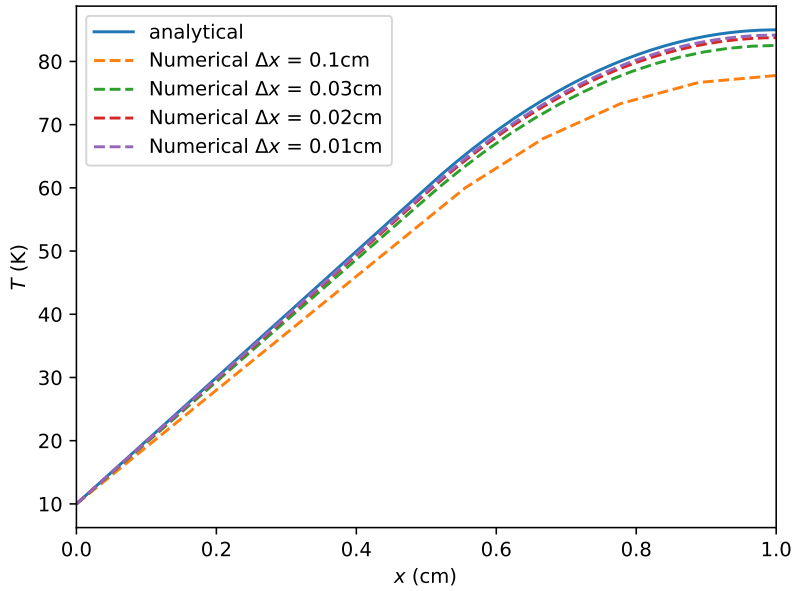


Figure 5.1: Temperature convergence for a selection of Δx. As Δx decreases the numerical solution approaches the analtyical solution as provided by [13].

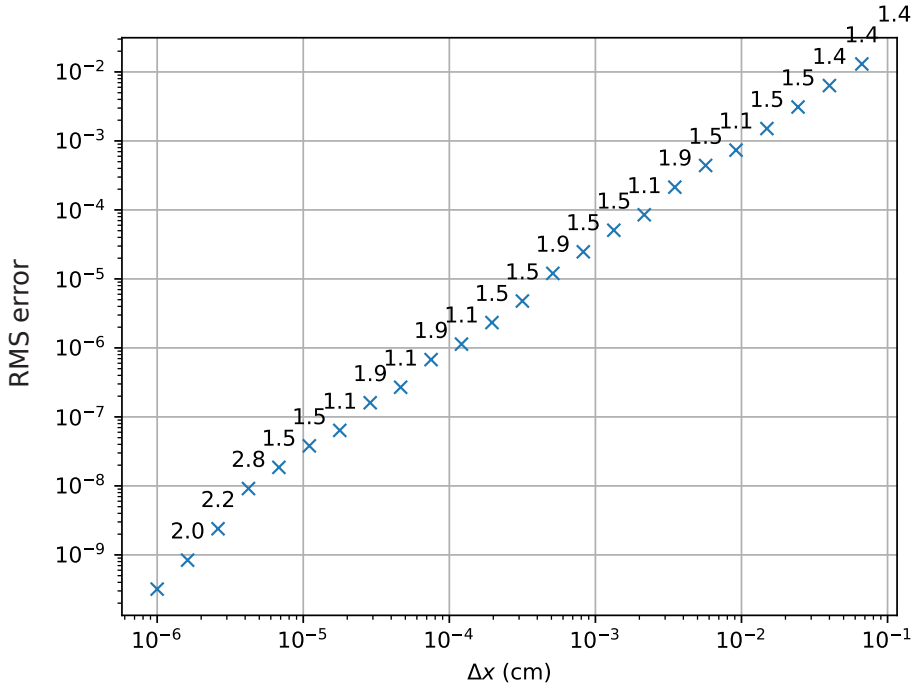


Figure 5.2: RMS error between the numerical and analytical solutions of $T(x)$ for differing Δx. Above the cross the slope to the next point is given. Here a slope of 2 means that the convergence goes with Δx^2.

From Figure 5.1 we can see that the numerical solution converges to the analytical expected temperature profile. From FVM we expect this convergence to go with Δx^2. In Figure 5.2 we first focus on $\Delta x < 10^{-5}$ cm. At this point the convergence goes with Δx^2. If Δx is higher the convergence is

considerably lower, going with 1.5 instead of 2. This is likely due to the fact that for larger Δx the temperature profile is not yet resolved.

Temporal

We create a pseudo analytical solution for our temperature profile in time. This pseudo analytical solution is made by running the solver for temperature at a very small Δt, in this case $\Delta t = 10^{-7}$ s, after which we compare the temperature solver for different Δt to that pseudo analytical solution. We run the temporal temperature model for 1 s, inducing a transient by changing the thermal diffusivity coefficient λ by 0.1, a 7.5% deviation from the steady state. In this run we use 6 fuel blocks positioned next to each other, each being 3.81 cm long, using $\Delta x = 0.127$ cm. Using BDF of order 1 (section 3.1.3) we expect the RMS error to decrease as a function of Δt.

Figure 5.3 shows the RMS error between the pseudo analytical solution and our numerical solution for varying Δt.

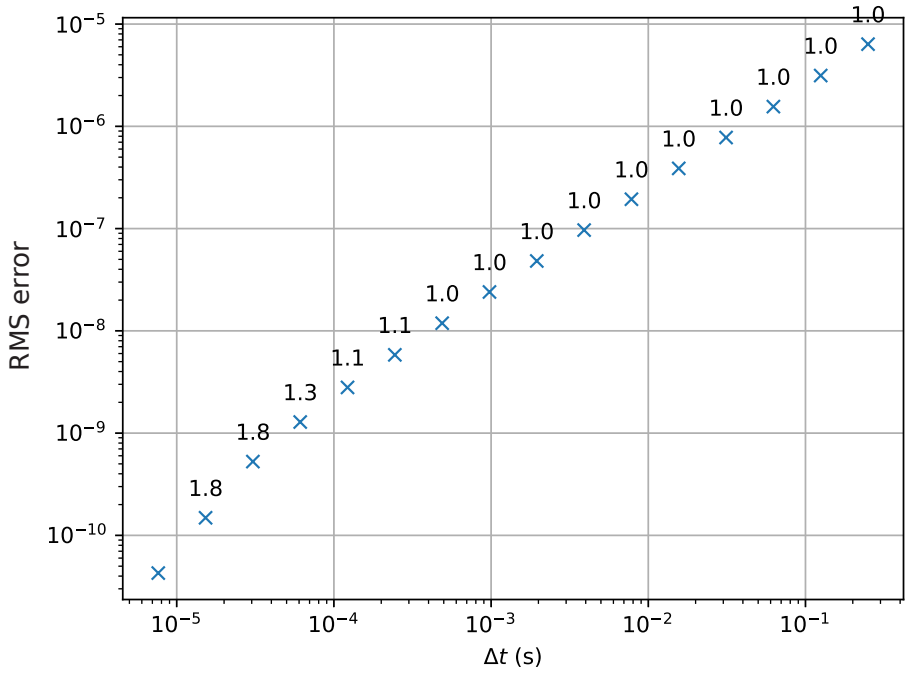


Figure 5.3: RMS error between the pseudo analytical temperature and the numerical temperatures. Starting from the top right every blue cross shows the RMS error at that Δt. Above the crosses the slope to the next cross is shown. Here 1 means that the slope goes with $\sim \Delta t$.

We see that the error in temperature is a linear function of Δt. We can also see, that as Δt approaches the pseudo analytical case the rate of convergence increases. We expect the RMS error between the pseudo analytical case and the numerical case to be within machine precision once their Δt is the same. As the numerical solution approaches the Δt of the pseudo analytical solution the rate of convergence increases.

5.1.2 Representative neutronics model

Spatial

The spatial neutronics approach is the same as the temporal temperature approach. We create a pseudo analytical function of the two group neutron diffusion equation 3.13, to which we compare various Δx, and we observe the convergence of the numerical solutions to the pseudo analytical solution. The pseudo analytical solution has been calculated for $\Delta x = 10^{-6}$ cm, for a bare core reactor of size 1 cm, subject to a homogeneous Neumann boundary on the left sides and an extrapolated boundary on the right, as given in Section 3.6 and 3.7. In our neutronics we calculate the two group neutron diffusion equation 3.13, from this we retrieve the fast and thermal fluxes, as well as the value of k_{eff}. We can thus observe the spatial convergence of these fluxes. Figure 5.4 shows the convergence of the fast fuxes to this pseudo-analytical solution. The convergence of the thermal flux is included in the appendices in Figure A.3.

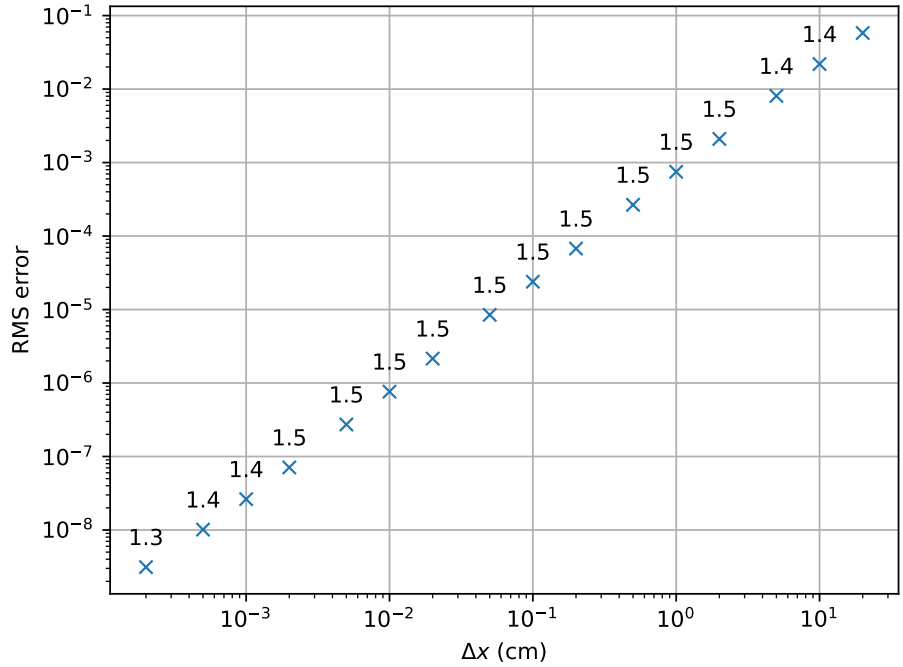


Figure 5.4: Convergence of the fast flux of numerical solutions with various Δx to a pseudo analytical solution. Above the value the slope to the next value is shown. Here 2 means that the slope goes with $\sim \Delta x^2$.

The convergence of k_{eff} is shown in Figure 5.5.

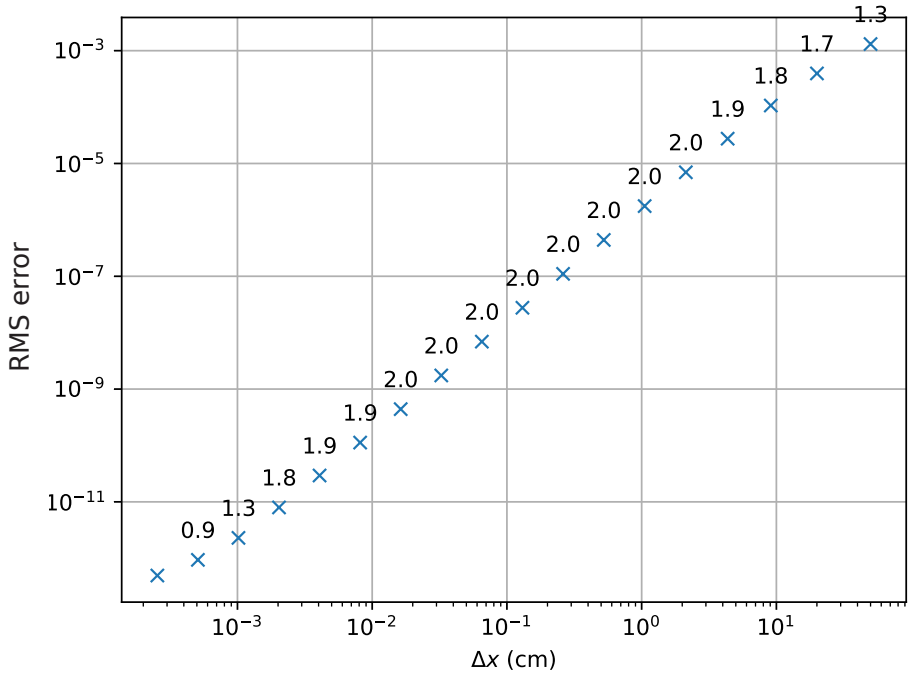


Figure 5.5: Convergence of the k_{eff} of numerical solutions with various Δx to a pseudo analytical solution. Above the value the slope to the next value is shown. Here 2 means that the slope goes with $\sim \Delta x^2$.

We observe that the convergence of k_{eff} is a function of Δx^2, as expected from FVM. However, the convergence of both the fast and thermal fluxes is lower, we observe that the convergence goes with $\Delta x^{1.5}$, considerably lower than the expected Δx^2. The representative neutronics model does not show the expected convergence.

This fact does not prevent us from using the model to further develop a ROM. Our high-fidelity model does not converge as we would expect, but it does converge. So if we take Δx small enough we retrieve the correct solution.

Temporal

We present the Δt convergence of the 2 group neutronics of the representative model. Some preliminary tests with 1 energy group and an infinite homogeneous reactor had been done first and are included in appendix A.3.

Just as with the temporal thermal solution, we present the convergence to a pseudo analytical solution. This pseudo analytical solution was made by running our temporal neutronics solver at $\Delta t = 10^{-7}$ s, after which we compare the neutronics solver for different Δt to that pseudo analytical solution. Through BDF (section 3.1.3) we expect the RMS error to decrease as a linear function of Δt.

The case we solved was a reactor in transient. We start with a steady-state reactor. We change the fast fission cross-section from 3.67×10^{-4} to 3.68×10^{-4} and the fast absorption cross section from 8.988×10^{-4} to 8.998×10^{-4}, lastly we change the fast neutron yield from 2.44 to 2.45. This results in a transient, which we run for 1 second.

We compare the RMS error as calculated from equation 3.2 of the fast fluxes and the thermal fluxes. Figure 5.6 shows this convergence for the fast flux. The thermal flux converges in a similar manner and is included in the appendices, in Figure A.4.

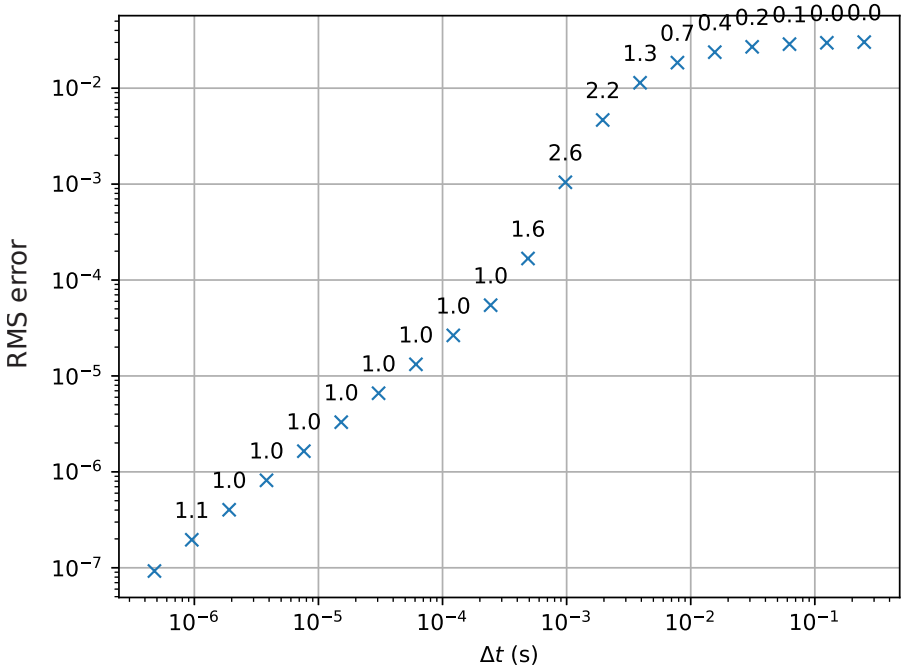


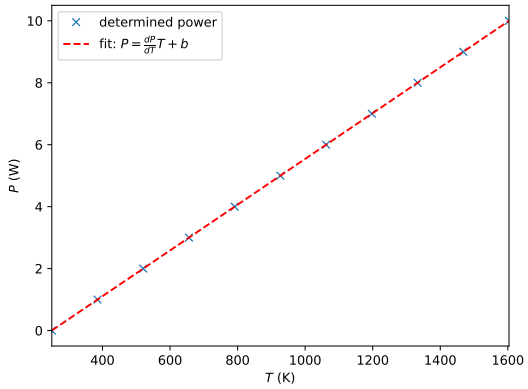
Figure 5.6: RMS error between the numerical and pseudo analytical fast fluxes. Starting from the top right every blue cross shows the RMS error at that Δt. Above the value the slope to the next value is shown. Here 1 means that the slope goes with $\sim \Delta t$.

In Figure 5.6 we see that for $\Delta t < 10^{-3}$ s the RMS error between the calculated fast flux and the pseudo analytical fast flux does not converge at all. Contrary to the temperature model, at these high Δt the neutronics solution is not yet resolved. This is due to the prompt jump not being fully resolved at such high Δt. The prompt jump is a quick change in fluxes due to the direct effect of the prompt neutrons. At $\Delta t < 10^{-3}$ s the numerical solution can resolve the fluxes. We see here, just like with the temperature that as Δt approaches the pseudo analytical case the rate of convergence increases, as exemplified after $\Delta t < 10^{-6}$ s. Looking at $\Delta t \in [10^{-3}, 10^{-6}]$ s wee see that the RMS error converges as a linear function of Δt.

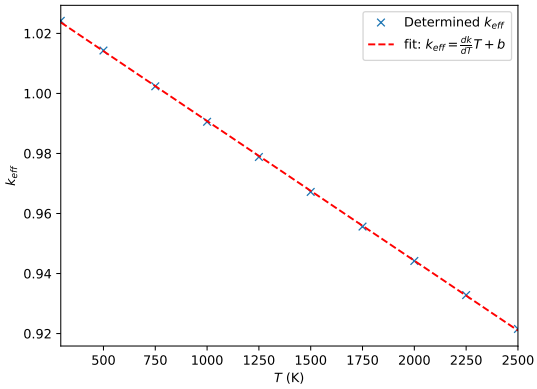
5.1.3 Representative coupled model

In the coupled model we perform two tests. First we perform a preliminary test, after which we determine the Δt RMS error convergence. The core is 101.6 cm long with 30 cm of graphite reflector added to the right side of the reactor. It is subject to a homogeneous Neumann boundary on the left, and an extrapolated boundary on the right. Our mesh has size $\Delta x = 1.31$ cm. When initialising the cross-sections we chose the cross-section in such a way that $k_{eff} = 1$ if there is a homogeneous temperature of 800 K in the core. In our preliminary test we run the coupled code, and determine whether the average temperature approximates this 800 K. In the Δt test we induce a transient in the reactor and determine the convergence to a pseudo analytical case.

Before we ran these tests we determined the $\frac{dP}{dT}$ and $\frac{dk}{dT}$ of the reactor as explained in Section 3.2.3. The resulting fits are shown in figures 5.7a and 5.7b.



(a) This figure shows the fit of $\frac{dP}{dT}$. The temperature in the reactor has been determined for a number of power input. The measurement points are shown with blue crosses, and the fit through the data is shown as a red dashed line. The fit was performed using the `Curve_fit` routine [40], this routine fitted the function $P = \frac{dP}{dT}T + b$ where b is a constant. The fit returned $\frac{dP}{dT} = 0.07 \pm 1.4 \times 10^{-18}\ \frac{\mathrm{W}}{\mathrm{K}}$.



(b) This figure shows the fit of $\frac{dk}{dT}$. k_{eff} in the reactor has been determined for a number of temperatures. The measurement points are shown with blue crosses, and the fit through the data is shown as a red dashed line. The fit was performed using the `Curve_fit` routine [40], this routine fitted the function $k_{eff} = \frac{dk}{dT}T + c$ where c is a constant. The fit returned $\frac{dk}{dT} = (-4.6 \pm 0.016) \times 10^{-5}\,\mathrm{K}^{-1}$.

Figure 5.7: $\frac{dP}{dT}$ and $\frac{dk}{dT}$ of our representative coupled model.

In Figure 5.7a we see the temperature as a function of power, through fitting the data to $P = \frac{dP}{dT}T + b$ with the `Curve_fit` routine [40], we determined $\frac{dP}{dT} = 0.07 \pm 1.4 \times 10^{-18}\ \frac{\mathrm{W}}{\mathrm{K}}$. For this we used as power input $P = 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10$ W.

In Figure 5.7b we see k_{eff} as a function of temperature. The neutronics model was ran for $T = 294, 500, 750, 1000, 1250, 1500, 1750, 2000, 2250, 2500$ K. Cross-sections were known for temperatures $T = 294, 2500$K. Temperatures in between those two were interpolated using,

$$\Sigma(T_3) = \frac{\Sigma(T_1) - \Sigma(T_2)}{T_1 - T_2} T_3 + \left(\Sigma(T_1) - \frac{\Sigma(T_1) - \Sigma(T_2)}{T_1 - T_2} T_1 \right)$$

Where $T_1 = 294\,\mathrm{K}$, $T_2 = 2500\,\mathrm{K}$, and T_3 the temperature for which we want to interpolate the cross-sections. The fit returned $\frac{dk}{dT} = (-4.6 \pm 0.016) \times 10^{-5}\,\mathrm{K}^{-1}$, which is around the value expected from literature, which is approximately -4×10^{-5}. Therefore, we are confident in our further coupled implementation.

Preliminary test

In the preliminary test we determined whether the average temperature in the core would approximate 800K. For this we ran the k_{eff} routine as described in Section 3.2.3. This test was run in a reactor with two fuel blocks and a reflector. The total length of the reactor was 131.6cm of which 30cm was a graphite reflector.

In Figure 5.8 the resulting temperature and flux profiles are shown.

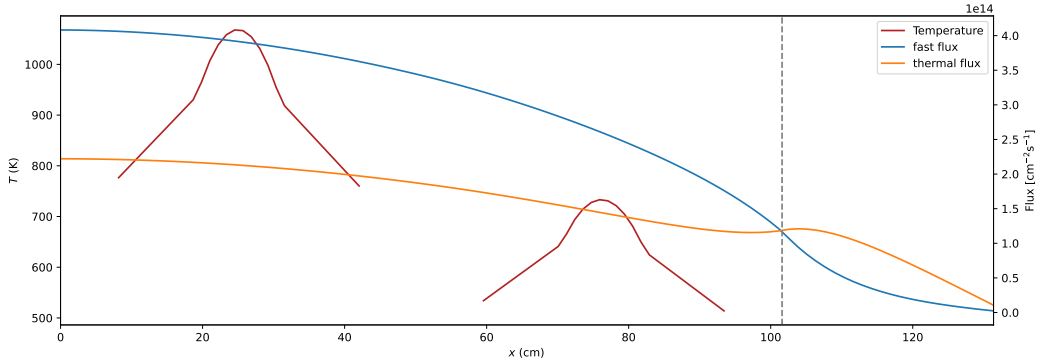


Figure 5.8: Temperature and flux profile at the steady-state $k_{eff} = 1$. In this case there are two fuel blocks, which have been mapped to the reactor core. The coolant temperature, defined at the boundaries of the fuel blocks is (from left to right) $T_{coolant} = 200, 167, 133$, and 100 K, for the fluxes there is a homogeneous Neumann boundary at the left, and an extrapolated boundary at the right. The left y-axis shows the temperature scale in Kelvin, the right y-axis shows the flux scale in $[\text{cm}^{-2}\text{s}^{-1}]$. The grey dotted line represents the change in core structure, to the left of the dotted line is the homogenized core, to the right is a pure graphite reflector.

The fuel blocks of Figure 5.8 have an average temperature as shown in Table 5.1.

Table 5.1: The average temperature over the entire fuel block.The average has been taken over the entire fuel block, using `np.average`.

	T_{Block_1}	T_{Block_2}
T (K)	905.66	619.60

The average of these averages is 762.63K, which is close to, but lower than, the expected value of 800K.

Temporal

To be sure that our coupled system was implemented succesfully we set out to determine the dependency of the RMS error on Δt. We induce a transient in the reactor, and determine a pseudo analyitcal solution. The transient induced is a deviation of 0.01 in both the fast and thermal neutron yields. The pseudo analytical solution has been determined with $\Delta t = 10^{-8}$s.

Figure 5.9 shows the RMS error of the fast flux profile, the thermal profile is similar, and is included in the appendices, in Figure A.5.

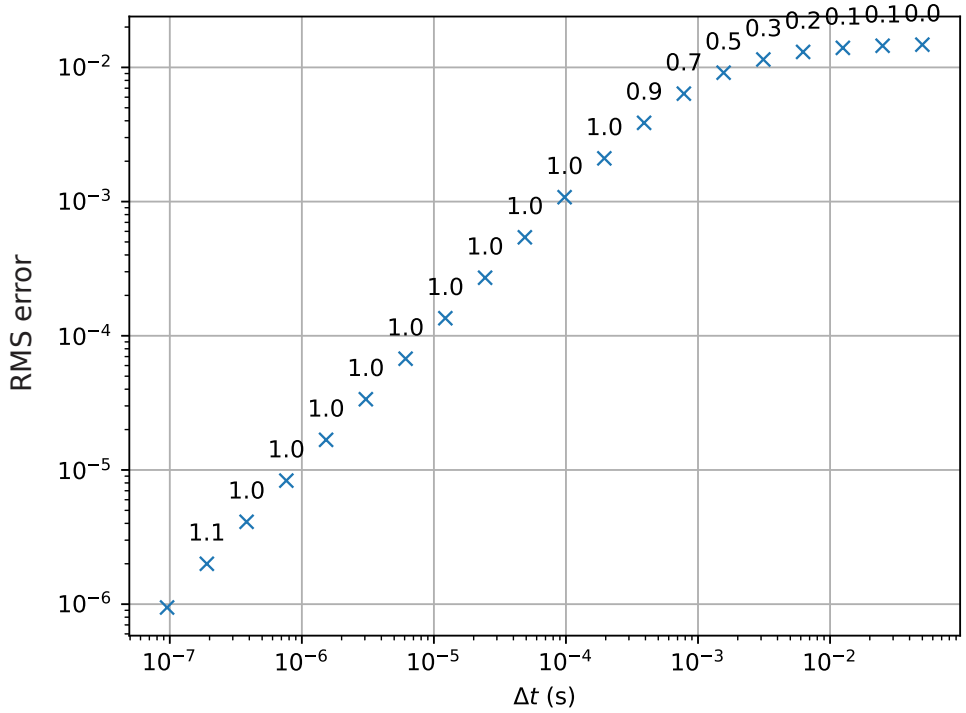


Figure 5.9: RMS error between the numerical and pseudo analytical fast fluxes Starting from the top right every blue cross shows the RMS error at that Δt. Above the value the slope to the next value is shown. Here 1 means that the slope goes with $\sim \Delta t$.

Similar to the neutronics flux profiles we observe that at first the flux shape does not converge at all. At these large Δt our solution is not yet resolved. Only if $\Delta t < 10^{-3}s$ the numerical solution can resolve the fluxes. We see here, that if $\Delta t < 10^{-6}$s the convergence appears to increase slightly. If we focus on the Δt in which we resolve the transient, and which are not too close to the pseudo analytical case, from $\Delta t \in [10^{-4}, 10^{-6}]$s, we see that the convergence is a linear function in Δt as expected from BDF.

5.2 Verification of ROM_T and ROM_N

In this section we discuss the verification of ROM_T and ROM_N, the temperature and neutronics ROMs, created by implementing OpInf on their respective representative models. We do this seperate verification for two reasons: firstly, we can determine the capabilities of OpInf, second we create the necessary ROMs for the coupled sROM procedure of Section 4.4.5.

5.2.1 ROM_T

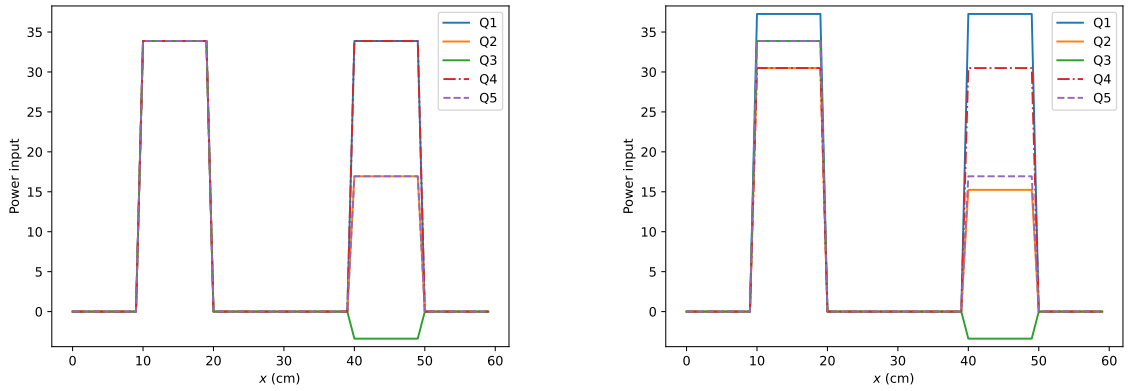
We train ROM_T with different inputs $\mathbf{Q}(t)$ through equation 4.12,

$$\frac{d}{dt}\mathbf{T}(t) = \mathbf{A}\mathbf{T}(t) + \mathbf{B}\mathbf{Q}(t).$$

We test the acquired ROM by running it with another input. We compare the attained temperature profile of the ROM to our representative model. This comparison is done by calculating the RMS error through equation 3.2.

ROM_T is trained with a training set containing different training runs. A training run is a specific power distribution in time and space $\mathbf{Q}(t)$, and the associated $\mathbf{T}(t)$ as determined by our FOM. In these training runs we use a power distribution over two separate fuel blocks in space, shown for example in Figure 5.10a: Fuel block 1 on the left, and Fuel Block 2 on the right. In a training run the temperature starts in a steady state. We then subject the temperature to the power distribution of that training run, from which we collect the corresponding temperature. Every training run starts with a certain power distribution at $t_{initial}$, these power densities than linearly change in time to the distribution at t_{final}. In Table 5.2 the training runs making up the training set for our ROM_T are shown. These power distributions are also visualised in Figure 5.10.

In the training runs we ran our FOM for 1000 s with $\Delta x = 0.13$ cm, $\Delta t = 1$ s.



(a) Power input in fuel blocks at $t = 0$s. Note that in the first fuel block all power inputs are the same. In the second fuel block $Q_1 = Q_4$ and $Q_2 = Q_5$.

(b) Power input in fuel blocks at $t = 4000$s. Note that in the first fuel block $Q_2 = Q_4$ and $Q_3 = Q_5$.

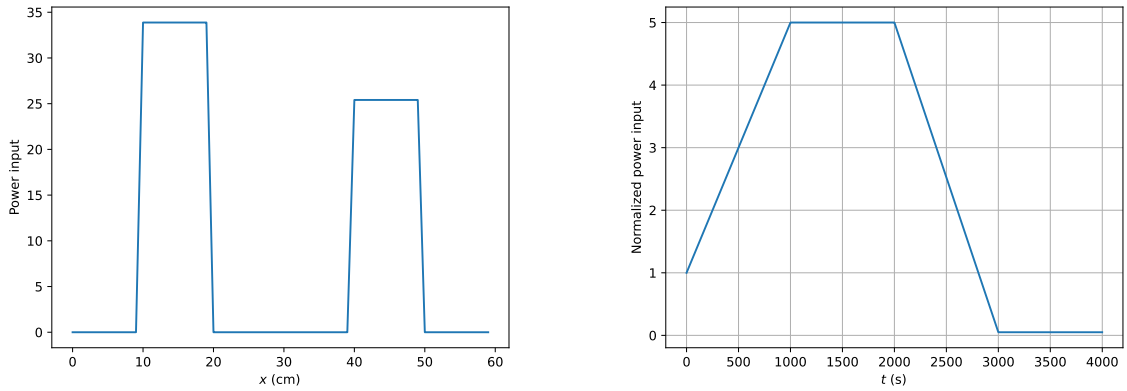
Figure 5.10: The initial and final power inputs with which we trained our ROM_T.

The change in magnitude of the various training sets is shown in Table 5.2.

Table 5.2: Change in magnitude of the 5 training runs used in the training set. The values shown here are the weights with which we multiply the steady state power to retrieve the power of the training run.

Training run	Fuel Block 1 $t_{initial}$	Fuel block 2 $t_{initial}$	Fuel block 1 t_{final}	Fuel block 2 t_{final}
1	1	1	1.1	1.1
2	1	0.5	0.9	0.45
3	1	-0.1	1	-0.1
4	1	1	0.9	0.9
5	1	1	1	0.5

To test the resulting ROM_T we used the testing case shown in Figure 5.11. We started with a temperature profile in steady state. At $t = 0$ s we start with 340 J equally distributed over the control volume Δx in the first fuel block, and 254 J equally distributed over the control volumes in the second fuel block. From $t = 0$ we linearly increase the power, $\mathbf{Q}(t = 1000\text{s}) = 5\mathbf{Q}(t = 0\text{s})$. We keep it 1000 s at that power, and then linearly decrease, $\mathbf{Q}(t = 3000\text{s}) = 0.1\mathbf{Q}(t = 0\text{s})$. Keeping the power at that level for 1000 seconds.



(a) Power input of the testing transient in fuel blocks at $t = 0$s. The second fuel block is 75% of the first fuel block.

(b) Normalized change in power input over time. This change in time is the same for all fuel blocks.

Figure 5.11: The initial power input and the power input in time with which we tested our ROM_T.

We first determined how many of the training runs from Figure 5.10 and Table 5.2 were necessary to successfully create a ROM_T. To that end, we used $r = 12$ modi and $k = 1500$ successive snapshots, each snapshot being of length Δt. We then constructed 5 different ROMs, the first one using only the first training run, the second one the first and the second, and so on. We determined the quality of the resulting ROMs by using the described testing case, calculating the resulting temperature profiles with both these ROMs and our FOM. We determine the RMS error at every time-step and calculate the average RMS error over time.

The average RMS error versus the amount of training is shown in Figure 5.12.

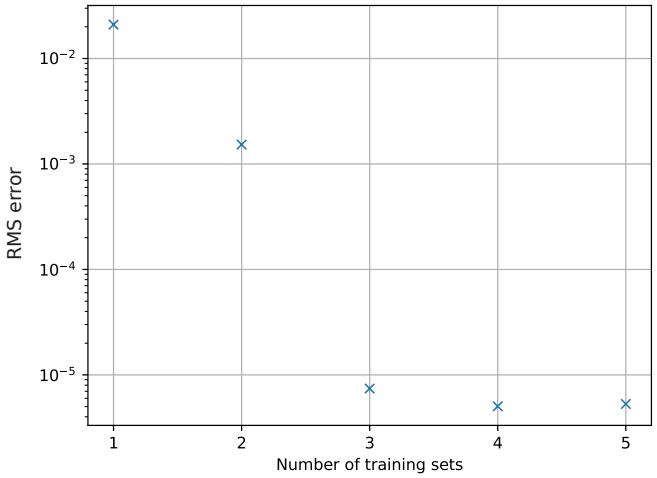


Figure 5.12: Average RMS error between the FOM and our ROM as a function of the amount of training sets used in creating the ROM.

We see that the first three training runs are the most important for the resulting ROM_T, after the third training set the added benefit of more training sets diminishes significantly.

Especially adding the third training run decreases the RMS error significantly. It appears that this is the case because this training run uses a negative weighted power distribution in the second fuel block. This results in the temperature lowering significantly, adding a whole new sort of dynamic with which we can train the ROM_T.

Next we determined how many modi and snapshots are necessary to create a good ROM_T. To that end we used the first three training sets previously described, and varied the amount of modi and snapshots. Using the same case as shown in Figure 5.11 we determined the average RMS error in time between the resulting temperature profile as determined by our FOM and the different ROMs.

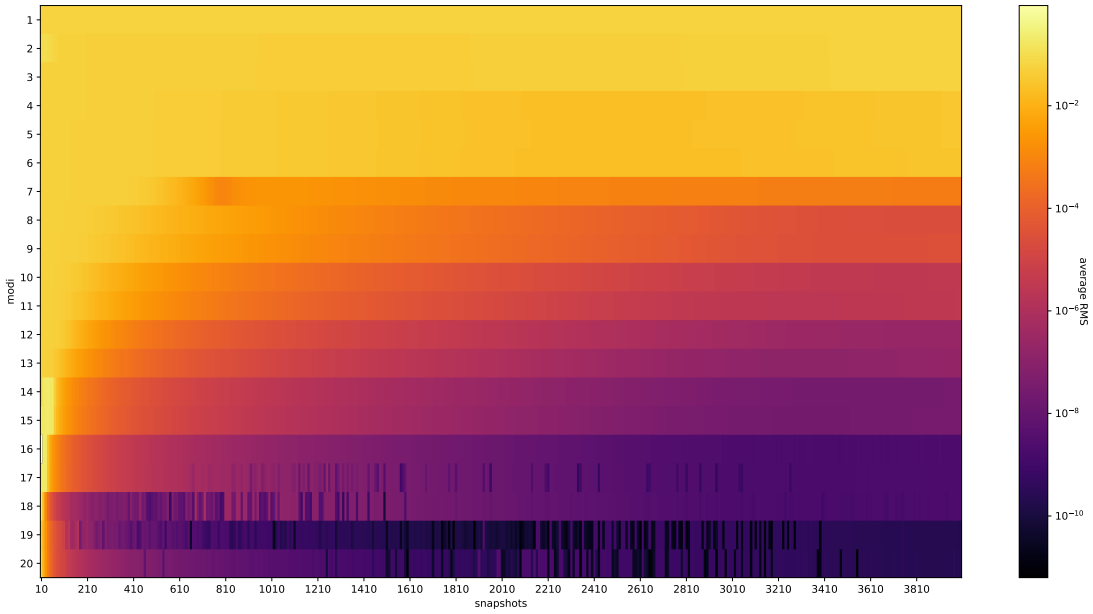
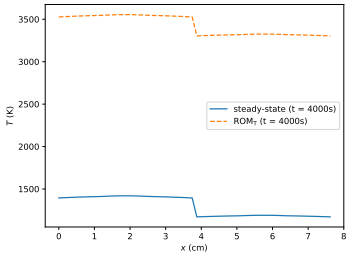


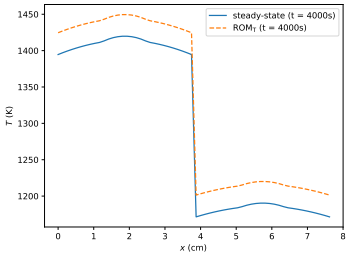
Figure 5.13: Average RMS error in time of ROMs with varying amount of modi (r) and snapshots (k). The snapshots are consecutive to each other. Note that the label on the right should read average RMS error

From Figure 5.13 we see that the RMS error improves when we add more modi and use more snapshots in creating a ROM. We observe an intriguing phenomena in Figure 5.13 at 17 to 20 modi we observe a sort of barcode, where taking more or fewer snapshots results in orders of magnitude difference in the average RMS error. We suspect this is due to the fact that the higher modi are associated with noise. As explained in Section 4.2 the first few modi contain the most important dynamics of a system. Adding more modi generally means adding more noise. The extent to which a mode contains noise is determined by the number of snapshots with which we create the modes. Including more snapshots generally increases the quality of the resulting ROM_T, but some snapshots may only add to the noise.

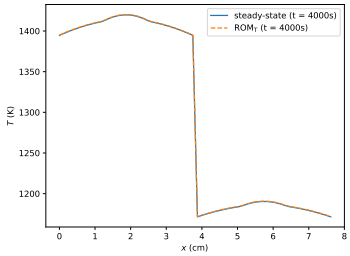
To give more meaning to this RMS error, we show the resulting temperature profile between our FOM and our ROM for three different cases in Figure 5.14, in Figure 5.14a we see the resulting temperature profiles if we use $r = 6$ modi and $k = 1500$ snapshots, Figure 5.14b shows the profiles if we instead use $r = 9$ modi, and Figure 5.14c if we use $r = 12$ modi. These profiles have respective average RMS errors of $3 \times 10^{-2}, 6 \times 10^{-4}, 5 \times 10^{-5}$



(a) Resulting temperature profiles of the FOM and ROM if we use $r = 6$ modi and $k = 1500$ snapshots, average RMS error = 3×10^{-2}.



(b) Resulting temperature profiles of the FOM and ROM if we use $r = 9$ modi and $k = 1500$ snapshots, average RMS error = 6×10^{-4}.



(c) Resulting temperature profiles of the FOM and ROM if we use $r = 12$ modi and $k = 1500$ snapshots, average RMS error = 5×10^{-5}.

Figure 5.14: Resulting temperature profiles for different amount of modi.

In Figure 5.14 we see that the average RMS error gives a rough estimate of the overall quality. If we focus on Figure 5.14a we see that the average RMS error is 3×10^{-2}, while the RMS error at t_{final} is around 5×10^{-1}. In general we remark that the RMS error of the temperature profile starts of very small, as both the FOM and ROM start at the same steady state temperature. Over time this error increases. Because the test case has a varying power input, we decided to look at the average RMS error to give a performance over time.

To ensure these results were robust, the RMS error of 1000 random test cases was determined. In these test cases $\mathbf{Q}(t)$ of the first fuel block was multiplied by a random value between -100 and 100 for all t. Furthermore, $\mathbf{Q}(t)$ of the second fuel block was multiplied by another random value also between -100 and 100 for all t. The time progression for both fuel blocks is as shown in Figure 5.11b. We found that the average RMS error in time was on average 10^{-5}, not exceeding 3×10^{-5}, the resulting graph is included in appendix Section A.5.

From these results we find that we can construct an accurate ROM_T with 12 modi using a training set with 3 distinct training runs, each snapshotted for 1500 s, that can predict for 4000 s,

5.2.2 ROM_T

In the neutronics ROM (ROM_N) we use training sets with different temperatures T to train our ROM, through equation 4.15,

$$\frac{d}{dt}\mathbf{X}(t;\mathbf{T}) = \mathbf{A}(\mathbf{T})\mathbf{X}(t;\mathbf{T}).$$

If we want to find the fluxes for T outside our training set, we need to estimate operator $\mathbf{A}$ by interpolating between the operators associated with the temperatures in the training set, as shown in Section 4.4.4. We test the ROMs by determining the RMS error between the resulting fluxes from our interpolated ROM and our FOM.

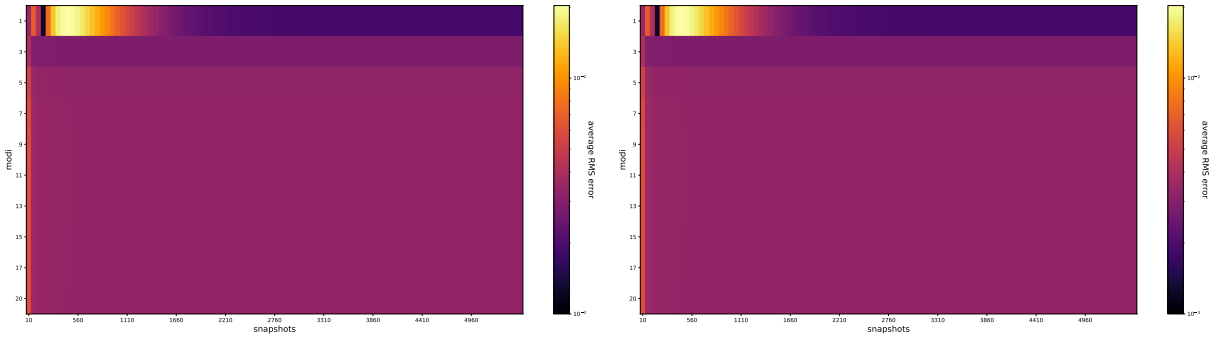
For our neutronics we used temperature as a parameter. From Section 4.4.2 we understand that this is implemented in two distinct ways. Using a 1 dimensional parameter (for our neutronics a homogeneous temperature), and using a p dimensional parameter. This p dimensional parameter is implemented as a reactor with 2 temperature zones, so as a 2 dimensional parameter. Furthermore, for the fluxes we have to preprocess our training set as explained in Section 4.4.1, especially if we consider interpolation in p dimensions.

Homogeneous temperature

As mentioned in Section 5.1.3 we ensure that the reactor is in a steady state ($k_{eff} = 1$) when the homogeneous temperature is 800 K. For our ROM_N we are interested in the transients resulting from a change in temperature. For the homogeneous temperature case, we use a training set containing 4 different temperatures constituting our training runs $T \in [294, 2500]$ K $= [294, 1030, 1765, 2500]$ K, using $\Delta x = 0.13$ cm, $\Delta t = 10^{-4}$ s for 0.5 s.

We first determine what sort of interpolation technique we should employ for a homogeneous temperature. We therefore compare a cubic to a linear interpolation technique.

We train our ROM using the 4 training runs, as the reactor is not in steady state at those temperatures, they induce a transient. Subsequently we test the ROM using $T_{test} = 800$ K predicting to 1 s, using $\Delta t = 10^{-4}$ s. After this we varied the amount of modi's and snapshots used to train the model. We determined the average RMS error in time between the ROM and the FOM. For the cubic interpolation technique this resulted in Figure 5.15a, and for the linear Figure 5.15b.



(a) Average RMS error in time of ROMs with varying amount of modi (r) and snapshots (k) with a cubic interpolation method. The snapshots are consecutive to each other.

(b) Average RMS error in time of ROMs with varying amount of modi (r) and snapshots (k) with a linear interpolation method. The snapshots are consecutive to each other.

Figure 5.15: Comparison of cubic and linear interpolation techniques in ROM_N.

From Figures 5.15a and 5.15b we observe that for both the linear and the cubic interpolation the ROM_N functions well. There is no discernible difference between the cubic and the linear interpolation technique. In the rest of the thesis we have taken the cubic interpolation, as it is the standard function in OpInf.

We also see that both interpolation techniques return good results if we only take 1 mode into account. This is most likely due to the fact that this one mode resembles the steady state shape of the fluxes and delayed neutrons. When we change the temperature homogeneous over the entire core, the shape does not change locally, but increases or decreases globally, maintaining the same shape in space. Adding more modi, adds more noise, which contributes to a larger average RMS error.
It is interesting to note, that if we do not use preprocessing, our average RMS error for $T_{test} = 800$ K, is considerably smaller. This is due to the fact that for this temperature the reactor is in steady state, as such the dynamics do not grow or shrink drastically, meaning that preprocessing the training sets does not add stability to the system, while it does filter out data. Preprocessing increases stability over all test cases, while it does make the ROM less accurate.

In Figure 5.16 we see that if we do not preprocess data, our RMS error at $T = 800$ K is exceptionally low, compared to the RMS error at other temperatures. Furthermore if we compare this profile to that of Figure 5.17, we see that the RMS error over all temperatures, bar those close to 800 K, are lower. In this thesis we have chosen to keep working with preprocessing, as it adds necessary stability to our ROM_N, especially if we go to p dimensional temperature.

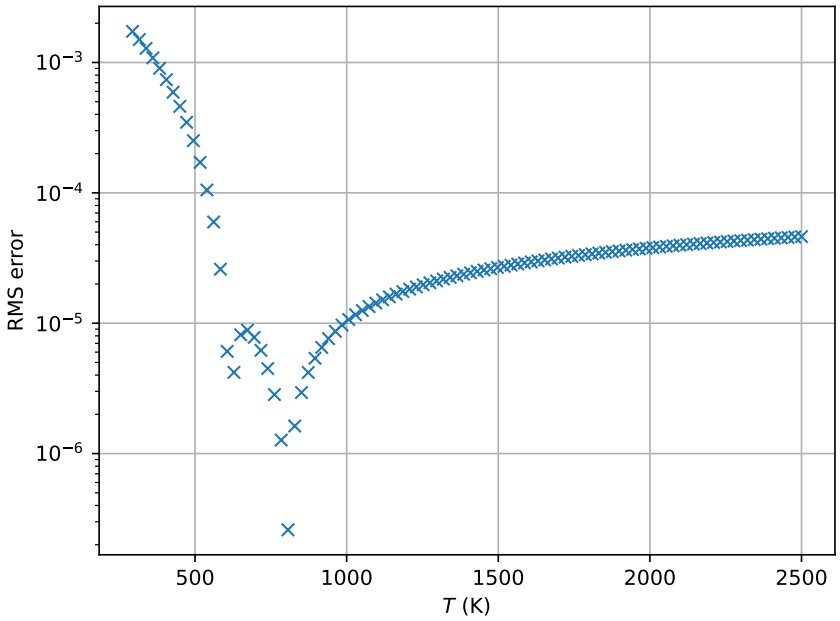


Figure 5.16: RMS error for different temperatures. We retrieve this RMS error if we do not preprocess our training set. This training set consists of 3 temperatures linearly spce between $T = 294$ and 2500 K. We use 7 modi, and 3690 snapshots at $\Delta t = 10^{-4}$ s. We run the ROM_N for 0.5 s at $\Delta t = 10^{-4}$ s.

Next we determined the influence of the amount of training runs in the training set on he RMS error. For this we varied the number of temperatures linearly spaced in between $[294, 2500]$ K which constitute our training runs. We varied this number between 3, 5, 7, and 9. For 9 parameters this means that $T = [294, 570, 846, 1121, 1397, 1673, 1948, 2224, 2500]$ K, for 7 parameters, this means that $T = [294, 662, 1029, 1397, 1765, 2132, 2500]$ K. In the meantime we kept the amount of modi and snapshots constant, using 7 modi and 3960 snapshots. We tested the resulting ROMs by predicting to 0.5 s at $\Delta t = 10^{-4}$ s with 100 different temperatures $T \in [294, 2500]$ K using cubic interpolation, after which we determined the RMS error between the ROM and our FOM at every time step and calculated the average RMS error in time. This resulted in Figure 5.17.

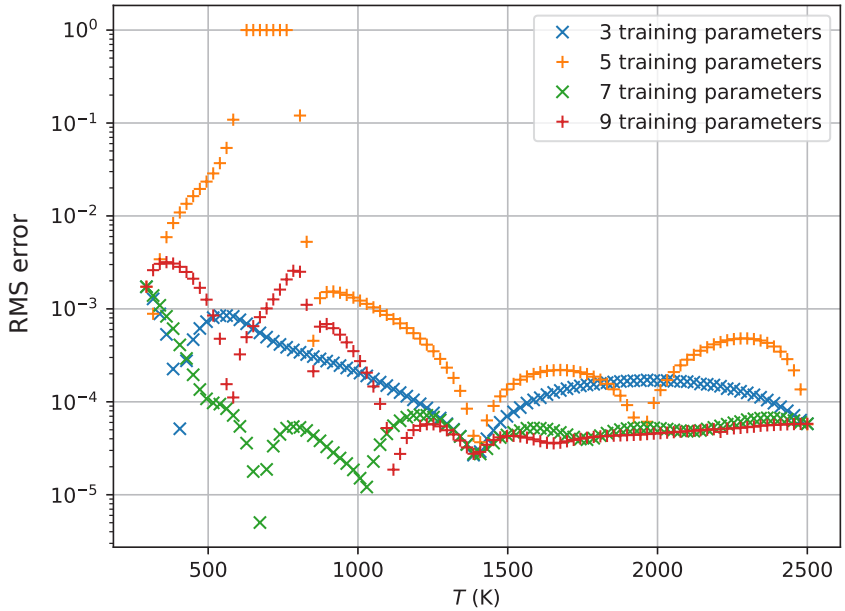
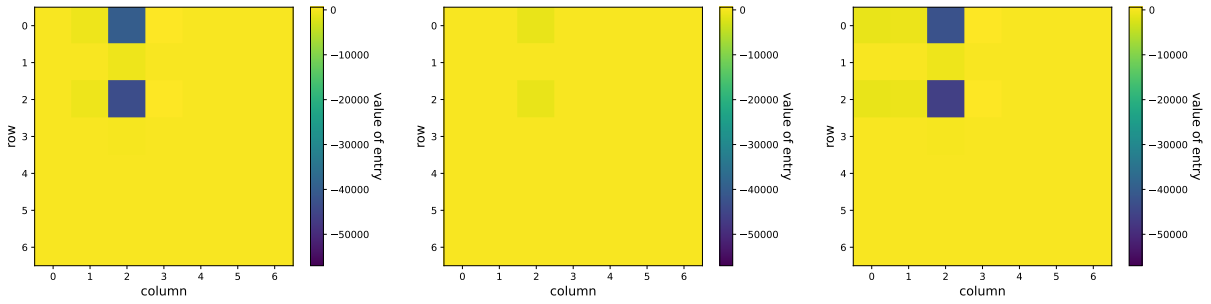


Figure 5.17: RMS error between fast neutron flux as calculated by our FOM and different ROMs. Every ROM had a different amount of training data within $T \in [294, 2500]$. The values at 10^0 correspond to the ROM not being able to timestep for those temperature values.

In Figure 5.17 we see that for $T > 1500$ K the amount of training we use to create the ROM does not drastically change the resulting RMS error. We see that if the test temperature is close to the training temperature the RMS error decreases, as is expected given that the difference between estimation and training temperature is smaller.
We also see that the ROM breaks down for temperatures close to 700K if we use 5 training runs, and gives inaccurate results if we use 9 training runs. The reason for this breakdown can be found in the operators OpInf constructs for the various training runs. Figure 5.18 shows the operators associated with three temperatures. We see that the second operator at $T = 846$ K, which is present if we use 5 or 9 training runs in our training set, has entries with considerably smaller values, due to the fact that it is close to the steady state. By adding this operator, we suspect that ROM_N has trouble correctly estimating for temperatures between $T = 294$ K and $T = 846$ K. If ROM_N interpolates for $T = 600$ K, the value of its nearest training run ($T = 846$ K) influences the interpolation the most. However this training run has a temperature just above the steady state, at that temperature fluxes decrease, while at $T = 600$ K the temperature is below the steady state temperature, so fluxes should increase. This means that the difference between ROM_N and our FOM increases, with the ROM_N breaking down in the end. If we compare 5 to 9 training runs, we see a general increase in the quality of ROM_N, because a training run with $T = 570$ K is included if we our set contains 9 runs. This also explains why we do not see a break down if we use 7 training runs in our training set, as that training set does not include a temperature close to $T = 800$ K. We also observe that the testing cases of $T = 294$ and 2500 K return a high average RMS error. We suspect this is due to the fluxes either blowing up, in the case of $T = 294$ K, or reducing to 0, in the case of $T = 2500$ K, these resultant fluxes are unphysical, and we can therefore not accurately predict them.

Figure 5.18 shows the operators associated with 3 distinct temperatures, $T = 294, 846$, and 1397 K.



(a) ROM_N operator for $T = 294$ K. *(b) ROM_N operator for $T = 846$ K.* *(c) ROM_N operator for $T = 1397$ K.*

Figure 5.18: Resulting ROM_N operators for various homogeneous temperatures.

The final test we ran with a homogeneous temperature was testing the response if we use a different Δt in training and predicting. To this end we trained a ROM with $\Delta t = 10^{-4}$ s and used it to time step using $\Delta t = 10^{-2}$. This test was performed for 0.5 s using 7 modi, 3960 snapshots, and estimating for a temperature of $T = 800$ K. To observe the behaviour of the fast flux in time, we summed the fast flux over the spatial domain and plotted the result of that sum in time.

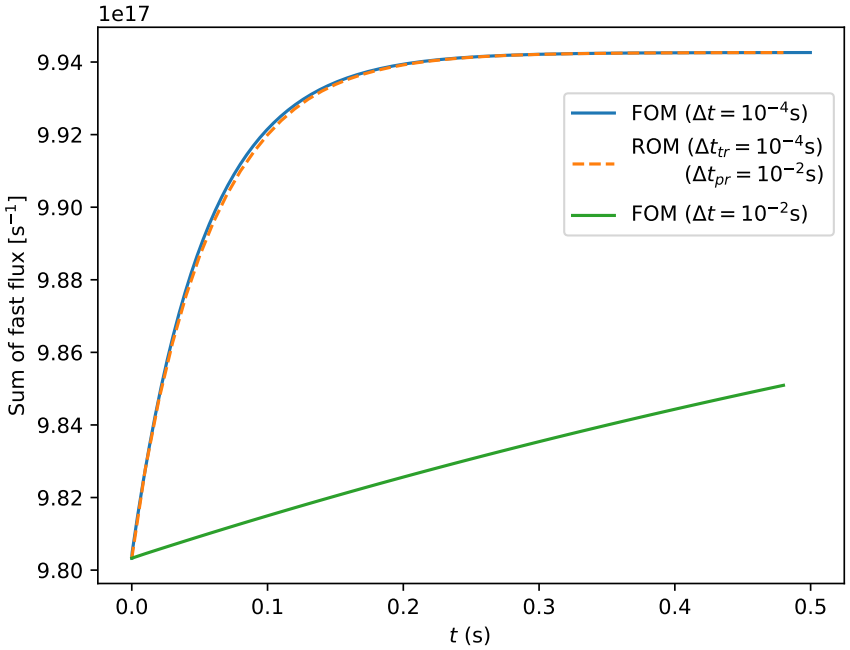


Figure 5.19: Spatially integrated fat flux at $\Delta t = 10^{-4}$ s and $\Delta t = 10^{-2}$ s, shown as respectively blue and green lines, as well as the spatial integration of the fast flu retrieved from a ROM trained at $\Delta t = 10{-}4$ s and timestepping with $\Delta t = 10^{-2}$ s. Δt_{tr} corresponds to the training Δt and Δt_{pr} corresponds to the testing.

We see that if we train a ROM with $\Delta t = 10^{-4}$ s and test it using $\Delta t = 10^{-2}$ s the resulting fluxes correspond to time stepping at $\Delta t = 10^{-4}$ s.

p dimensional temperature

Using the method set forth in Section 4.4.2 we can construct a ROM_N which can determine neutronics while estimating parameters in p dimensions. In our model we use this to discretize the temperature in the reactor, going from a homogeneous temperature to two temperature zones.
We first determined the effect of modi and snapshots on the quality of the ROM_N. To this end we created a training set consisting of 100 training runs. These training runs contained two temperatures, one for each halve of the reactor, with $T_{1,2} \in [750, 80]$ K. The temperatures of the training runs were equidistantly placed in the temperature parameter space between $T = 750$ K and $T = 850$ K. In crating a ROM_N we varied the amount of modi in construction, and the amount of snapshots taken out from the training runs. As in Section 5.1.3 the reactor is in steady state if there is a homogeneous temperature of 800 K in the core. Any temperature deviating from this will induce a transient.
Every training run was done with $\Delta x = 1.3$ cm, $\Delta t = 10^{-4}$ s, running for 0.1 s, with varying amount of modi $r = [5, 15, 25]$ and snapshots $k = [250, , 500, 1000]$. We subsequently tested the acquired ROM by taking 10000 samples in the parameter space, not overlapping with the training. The resulting training runs and testing cases are shown in Figure 5.20. For every test we ran our FOM and our ROM for 1 s at $\Delta t = 10^{-4}$ s, determining the RMS error between them at every time step. We then took the average of that RMS error, resulting in 10000 average RMS error values.

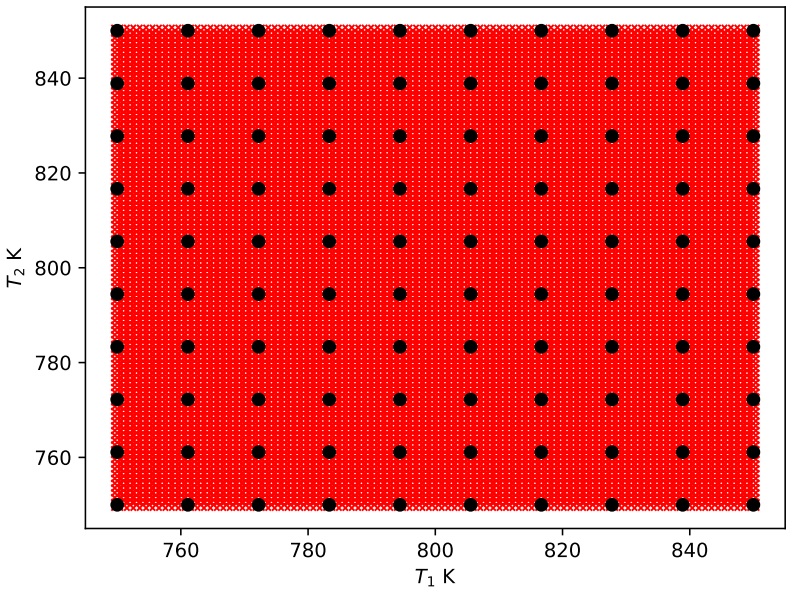


Figure 5.20: Visualisation of the training and testing points within parameter space. The training points are shown with black dots, and the testing points with red crosses.

To determine an overall quality of the ROM we took the average of all those RMS error values. The resulting RMS error averages are shown in Table 5.3.

Table 5.3: Average RMS error values between ROMs with different number of modi (r) and snapshots (k) used in training compared to the FOM. All RMS error values are 10^{-5}.

	$r = 5$	$r = 15$	$r = 25$
$k = 250$	7.0415	7.0397	7.0396
$k = 500$	4.2119	4.2146	4.2144
$k = 1000$	3.5240	3.9633	3.9631

We see that the quality of our ROM is enhanced if we take more snapshots into account. Furthermore we see that modifying the amount of modi does not alter the resulting quality of the ROM that

much. The separate profiles of RMS error at testing points for the different modi and snapshots are included in the appendices, in Figure A.8.
The fact that including more modi does not affect our RMS error that much is expected, if we look at the decay of the SVD values, shown in Figure 5.21.

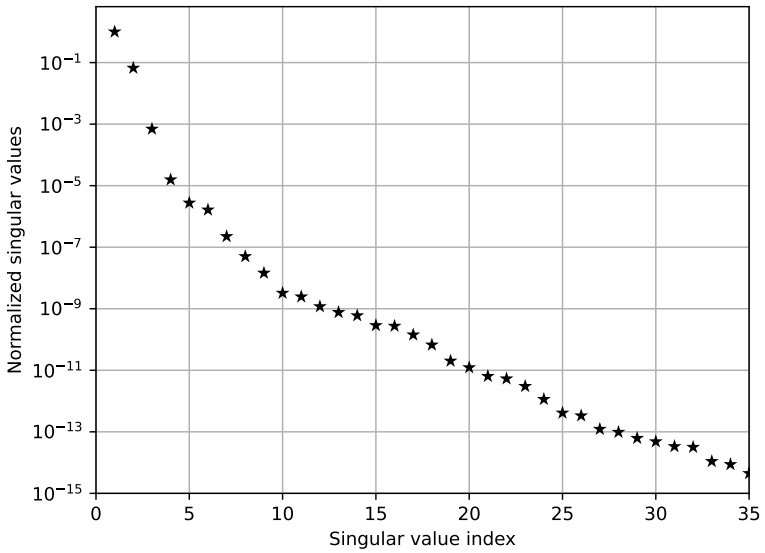


Figure 5.21: Decay of the singular values of the POD basis derived from the training regime.

We see that the normalized singular value already is below 10^{-5} if we take 5 modi into account. All added modi after that only account for relatively insignificant dynamics.
Next we wanted to investigate the role of Δt. For this we varied the Δt with which we trained the ROM. In these investigations the amount of training points and modi were constant, with 100 training points and $r = 15$ modi. Furthermore, the training runs all ran for 1 s. The amount of snapshots changed with Δt, where we snapshotted the entirety of the training run, so 100 for $\Delta t = 10^{-2}$ s, 1000 for $\Delta t = 10^{-3}$ s and 10000 for $\Delta t = 10^{-4}$ s. We determined the average RMS error per testing point for 10000 testing points in the temperature domain of $T_{1,2} \in [750, 850]$ K, we then determined the overall quality of the ROM by averaging over those 10000 RMS errors. The results are shown in Table 5.4.

Table 5.4: RMS error values between ROMs trained with varying Δt and their FOM counterparts.

Δt (s)	RMS error (10^{-5})
10^{-2}	4.0316
10^{-3}	5.3970
10^{-4}	27.172

We see that the RMS error does not change that much between $\Delta t = 10^{-2}$s and $\Delta t = 10^{-3}$s. However it increases an order of magnitude towards $\Delta t = 10^{-4}$s. This can be attributed to the fact that at $\Delta t = 10^{-4}$s the FOM is better resolved. If we decrease Δt our FOM will provide more detail on the prompt jump, which occurs at small time scales, right after the reactor has entered a transient. This means that if Δt is made smaller, a ROM should also account for these dynamics. If a ROM has to account for more types of dynamics, it may need to include more modi.

Knowing the influence of Δt, and of the amount of modi and snapshots, we wanted to see the quality of ROM_N if we use the entirety of the temperature domain $T \in [294, 2500]$ K in our training set, and test cases. The training set for the ROM_N contained 900 training runs, with $T_{1,2} \in [294, 2500]$ K. The temperatures of the training runs were equidistantly placed in the temperature parameter space. We constructed a ROM using $r = 15$ modi and $k = 1000$ snapshots of the training runs with $\Delta t = 10^{-3}$ s. Next we tested the resulting ROM by estimating for 90000 different temperature combinations in the parameter space. Of these points the RMS error was determined between the ROM and the FOM at every time step, after which we calculated the average RMS error over all time steps. In Figure 5.22 we see the resultant RMS error for every temperature. The figure is not symmetric, as the reactor core which was modelled is not symmetric.

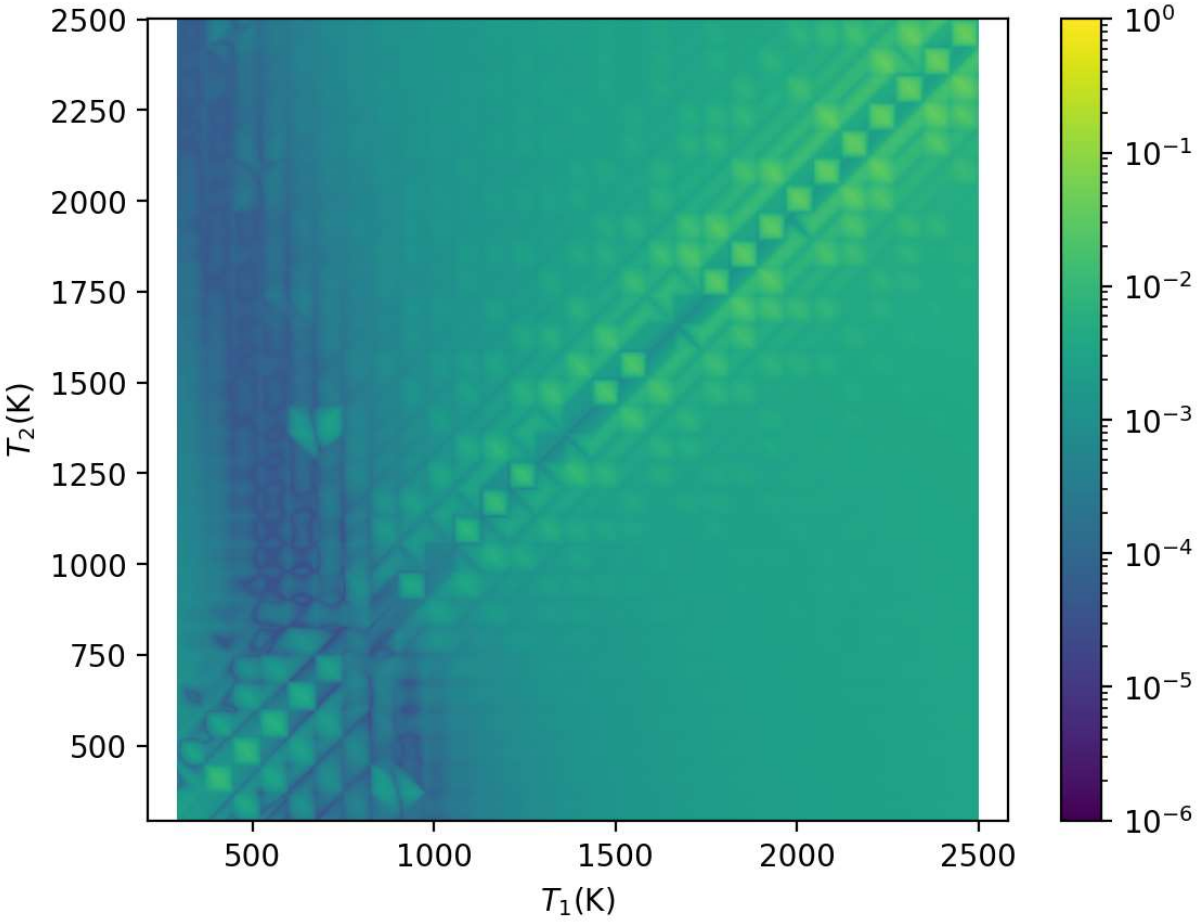


Figure 5.22: RMS error of the tested temperatures. 90000 different temperature combinations as test cases were used to test the ROM. At each of these temperatures the RMS error at every time step between the ROM and our FOM was calculated. The resulting colour is the average of those RMS errors in time.

It should be noted that Figure 5.22 does not capture the behaviour of the RMS error entirely. To visualise the behaviour of RMS error in the parameter space we used a `LinearNDInterpolator` between the acquired RMS error values. Therefore the colour does not correspond one-to-one with the actual RMS error value at that location, but this visualisation does give a good overview of the behaviour of our ROM in parameter space. We see that the ROM taking 2 temperature zones into account performs slightly worse for temperatures far from steady state temperatures, compared to the homogeneous temperature. This is expected, due to interpolation in 2 dimensions being less accurate than in 1 dimension. We further note that the ROM performs well if the temperature resembles a steady state. We do not observe this in Figure 5.17, because for the homogeneous temperature our training sets had fewer training runs around the steady state temperature. In this 2 dimensional case there are 900 training runs in the training set. Meaning that unlike the homogeneous temperature, we are unlikely to overfit the estimation of the operators.

5.3 Coupled ROM of temperature and neutronics

In Section 4.4.5 the two approaches to creating a ROM were discussed, the mROM and the sROM. In this section those two methods will be tested and compared to each other.

In our tests of the two ROMs we initialise a bare core reactor. This reactor has a length of 101.6 cm, with $\Delta x = 1.31$ cm. The neutronics of the reactor is subject to a homogeneous Neumann boundary on the left and an extrapolated boundary on the right. Inside the reactor there are two fuel blocks. The coolant temperature is, from left to right $T_{coolant} = [500, 466, 433, 400]$ K. Figure 5.23 shows the steady state flux and temperature profiles of this reactor.

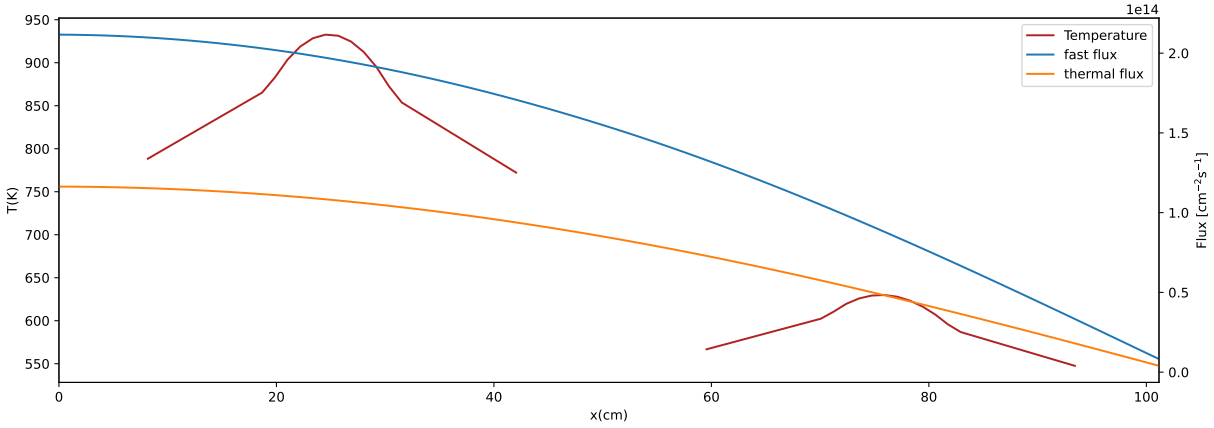


Figure 5.23: The steady state flux and temperature profiles. This reactor has a length of 101.6 *cm, with* $\Delta x = 1.31$ *cm. The reactor is subject to a homogeneous Neumann boundary on the left and an extrapolated boundary on the right. Inside the reactor there are two fuel blocks. The position of the fuel blocks is at the centre of the red lines denoting temperature. The coolant temperature is, from left to right* $T_{coolant} = [500, 466, 433, 400]$ *K.*

In the test and comparative cases we induce the same transient in both the sROM and the mROM. This transient is in the form of a depressurized loss of forced cooling (DLOFC) incident. We chose to simulate a DLOFC, as it can be simulated by a change in the temperature model.
The reason we want a transient to only change the temperature model, is because the neutronics model in the sROM already has to use temperature as a parameter. If we include one more parameter in the neutronics model the ROM_N of the sROM would have to interpolate in 3 dimensions, whereas the mROM would only have to interpolate in 1 dimension. We want to avoid this because we see that the error increases if we increase the amount of dimensions for which we need to interpolate. For example, in Figure 5.13 we see the RMS error if we have two fuel blocks (interpolation in 2 dimensions). In the appendix we include Figure A.7, where we see that the RMS error if we have three fuel blocks (interpolation in 3 dimensions) is orders of magnitude larger compared to 5.13. This means that if the parameter influenced the neutronics model, the mROM would get an unfair advantage over the sROM.

In our model the DLOFC incident is implemented by changing the heat transfer coefficient at the mixed boundaries between the the fuel blocks and the coolant, we recall equation 3.4,

$$-\lambda \frac{\partial}{\partial x} T(x,t)\Big|_{x=B} = \zeta \alpha \left(T(x,t) - T_{ref} \right), \tag{5.2}$$

with T_{ref} is the temperature of the coolant channel, $T(x,t)$ the temperature at the boundary, λ the thermal diffusivity coefficient, and B the x position of the boundary. In our reactor there are 4 such boundaries. Compared to equation 3.4 we have added a factor ζ before the heat transfer coefficient α. In our testing we change this factor from 1, indicating a steady-state, to 0, indicating a full DLOFC incident. This factor is homogeneous throughout the reactor, and is used as a parameter in our ROM, whose implementation is consistent with a 1 dimensional parameter of Section 4.4.2. In our mROM this parameter is applied as a single 1 dimensional parameter to the entire ROM. In our sROM the parameter is implemented as a 1 dimensional parameter for the ROM_T.

5.3.1 Preliminary test

In a preliminary test we train and run the mROM and the sROM and see whether their results resemble the results of our FOM. The training set of both the mROM and sROM contained 3 ζ value training runs: $\zeta = [0, 0.5, 1]$. Subsequently we tested both ROMs with a $\zeta = 0.25$, and compared them to a FOM with $\zeta = 0.25$. In this section all training was done by taking 1000 snapshots of the FOM running for 1s at $\Delta t = 10^{-3}$ s. Afterwards all testing was done by running the ROMs for 4s with $\Delta t = 10^{-3}$, and comparing it to the FOM running for the same amount of time.

Preliminary mROM test

We constructed the mROM using 15 modi. Figure 5.24 shows the resultant fluxes and temperature profiles for both the FOM and the mROM.

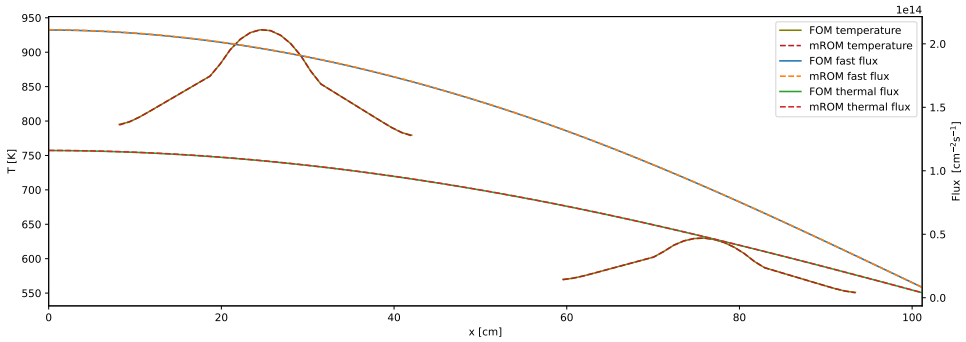
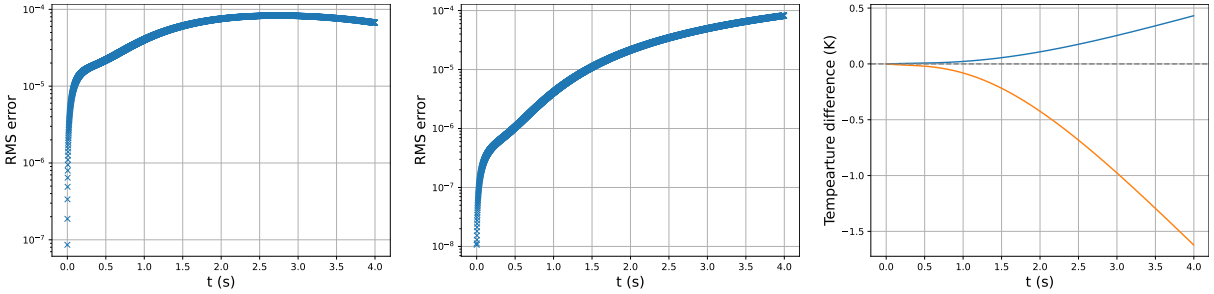


Figure 5.24: Profiles of the fluxes and temperature of the mROM and our FOM at $t = 4$ s. The mROM training set consisted of 3 different heat transfer coefficient values equidistant between 0 and 0.03 $\mathrm{Wcm^{-2}K^{-2}}$ to simulate a DLOFC, all training runs lasted for 1 s at $\Delta t = 10^{-3}$ s, of which we took 1000 snapshots. We tested the mROM with a heat transfer coefficient of 0.0075 $\mathrm{Wcm^{-2}K^{-2}}$, predicting for 4 s with $\Delta t = 10^{-3}$ s.

We see that the mROM fits nearly perfectly on both the fluxes and the temperature. In Figure 5.25 we see the RMS error in time of the fast flux and the temperature as well as the maximum temperature difference in time, both positive and negative, between the mROM and the FOM.



(a) RMS error between the fast flux as determined by the mROM, and our FOM.

(b) RMS error between the temperature as determined by the mROM, and our FOM.

(c) Maximum over- and underestimate of temperature between the mROM and our FOM. The blue line shows the maximum overestimate, and the orange line the maximum underestimate. The grey dotted line signifies that $\Delta T = 0$ K.

Figure 5.25: Analysis of various errors between our mROM and our FOM. The mROM training set consisted of 3 different heat transfer coefficient values equidistant between 0 and 0.03 $\mathrm{Wcm^{-2}K^{-2}}$ to simulate a DLOFC, all training runs lasted for 1 s at $\Delta t = 10^{-3}$ s, of which we took 1000 snapshots. We tested the mROM with a heat transfer coefficient of 0.0075 $\mathrm{Wcm^{-2}K^{-2}}$, predicting for 4 s with $\Delta t = 10^{-3}$ s.

From this preliminary test we see that our mROM is able to compute a transient accurately, tending to underestimate the temperature.

Preliminary sROM test

The sROM is made up of two ROMs, ROM_T and ROM_N, each with their own training sets.
The ROM_T was constructed with 15 modi, using the same training inputs as in Section 5.2.1. The ROM_N was constructed with 15 modi. The ROM_N training set comprised 144 equidistant temperatures in the temperature parameter space $T = (T_1, T_2)$, with $T_{1,2} \in [294, 2500]$K.

Figure 5.26 shows the resultant fluxes and temperature profiles for both the FOM and the sROM.

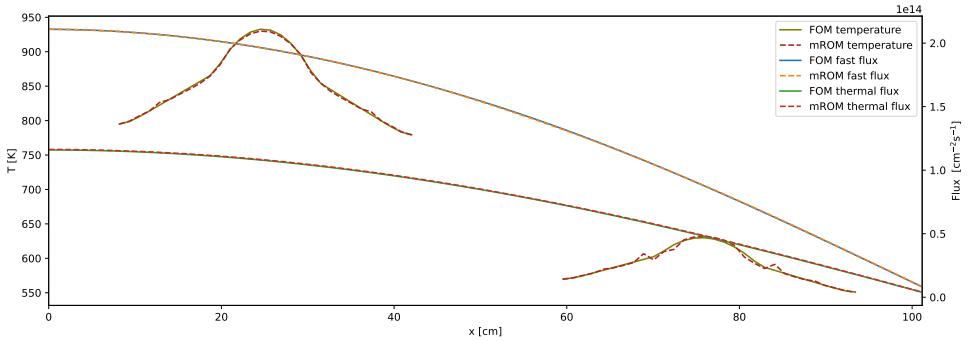
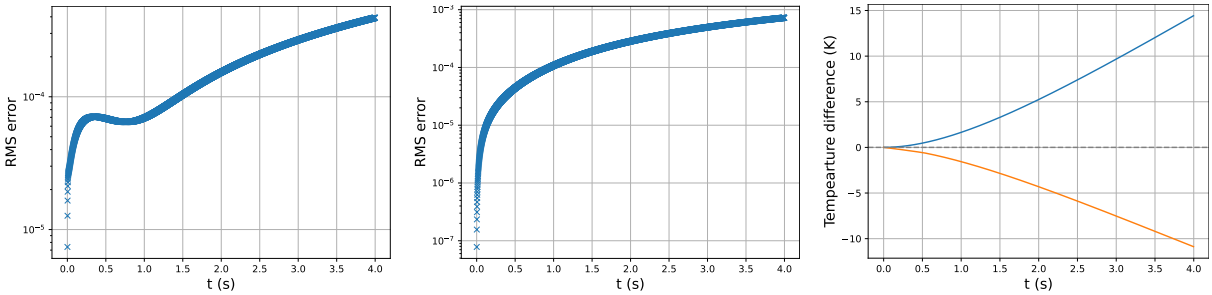


Figure 5.26: Profiles of the fluxes and temperature of the sROM and our FOM at $t = 4$ s. The training set of ROM_T of sROM consisted of 3 different heat transfer coefficient values equidistant between 0 and 0.03 $\mathrm{Wcm^{-2}K^{-2}}$ to simulate a DLOFC, all training runs lasted for 1 s at $\Delta t = 10^{-3}$ s, of which we took 1000 snapshots. We tested the sROM with a heat transfer coefficient of 0.0075 $\mathrm{Wcm^{-2}K^{-2}}$, predicting for 4 s with $\Delta t = 10^{-3}$ s.

We see that the temperature profile of the first fuel block underestimates the temperature at the border between the fuel and the graphite. Furthermore we see that in the second fuel block this border shows noise. In Figure 5.27 we see the RMS error in time of the fast flux and the temperature as well as the maximum temperature difference, both positive and negative, between the mROM and the FOM.



(a) RMS error between the fast flux as determined by the sROM, and our FOM.

(b) RMS error between the temperature as determined by the sROM, and our FOM.

(c) Maximum over- and underestimate of temperature between the sROM and our FOM. The blue line shows the maximum overestimate, and the orange line the maximum underestimate. The grey dotted line signifies that $\Delta T = 0$ K.

Figure 5.27: Analysis of various errors between our sROM and our FOM. The training set of ROM_T of sROM consisted of 3 different heat transfer coefficient values equidistant between 0 and 0.03 $\mathrm{Wcm^{-2}K^{-2}}$ to simulate a DLOFC, all training runs lasted for 1 s at $\Delta t = 10^{-3}$ s, of which we took 1000 snapshots. We tested the sROM with a heat transfer coefficient of 0.0075 $\mathrm{Wcm^{-2}K^{-2}}$, predicting for 4 s with $\Delta t = 10^{-3}$ s.

From this preliminary test we see that our sROM is able to compute a transient but shows errors significantly higher compared to the mROM. We see that the maximum temperature unerestimation is an order of magnitude higher, and the maximum overestimation is even two orders of magnitude higher. Furthermore the RMS errors are an order of magnitude higher.

5.3.2 Training analysis

In training a ROM a number of variables influence the quality of the resulting ROM. These variables include the amount of modi with which the ROM is constructed, the amount of different types of training data which the ROM may use, and the amount of snapshots of those training data. In this subsection the influence of three of these variables on the quality of the ROM are analysed. We look at 2 sizes of the training set. One where we use 3 training runs for the ROM ($\zeta = [0, 0.5, 1]$), and one where we use 5 training runs for the ROM ($\zeta = [0, 0.25, 0.5, 0.75, 1]$). We run our FOM with each of these training runs for 25 s using $\Delta t = 10^{-3}$ s. Subsequently our ROM can use k snapshots in every run to train itself. There are 15 different values of k geometrically spaced between 10 and 25000, $k = [10, 17, 30, 53, 93, 163, 285, 499, 874, 1528, 2673, 4675, 8175, 14296, 25000]$, so at first we only use the the first 0.01 s of the runs to train our ROMs, while at the end we use the full 25 s of the runs. For every value of k we vary the amount of modi r with which we construct the ROM, $r = [4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30]$. For every combination of amount of runs in the training set, snapshots and modi, we determined the RMS error between the ROM and FOM in time. We did this for both the fast flux and the temperature, averaging over the total time steps to retrieve an average RMS error over all time. For all of these combinations, we also retrieved the maximum absolute difference in temperature between the ROM and the FOM.

Training analysis of the mROM

Using a training set with 3 training runs, we determined the maximum absolute temperature difference, and the RMS error of the fast flux and the temperature. Figure 5.28 shows the maximum absolute temperature difference for different amounts of modi and snapshots. The plots for the RMS error of the fast flux and of the temperature tend towards the same conclusions, they differ in precise detail. But in general we see that we need a lot of snapshots, and few modi. These plots are included in appendix A.8. Similarly the plots using 5 training runs show no discernible difference and are included in appendix A.9

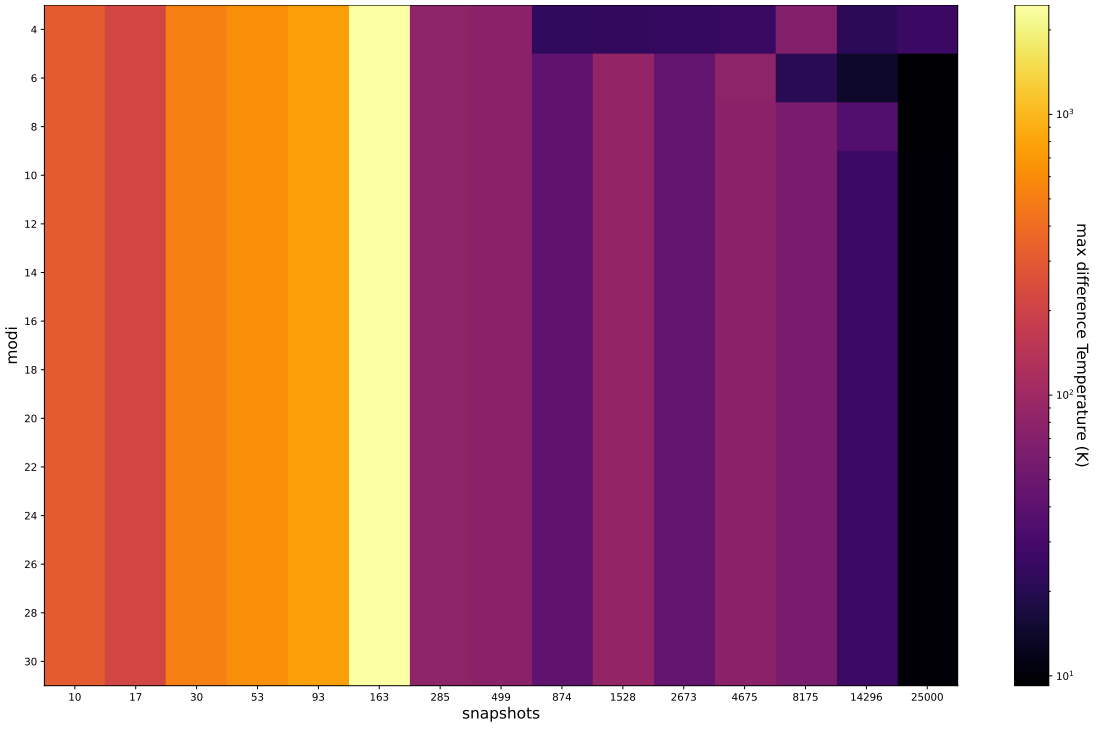


Figure 5.28: The maximum absolute temperature difference in time between our mROM and our FOM. Here we use a training set with 3 different heat transfer coefficient values equidistant between 0 and the nominal value of 0.03 $\mathrm{Wcm^{-2}K^{-2}}$ *to simulate a DLOFC. Every run is 25 s long with* $\Delta t = 10^{-3}$ *s. The resulting max temperature difference is for a test with a heat transfer coefficient of 0.00375* $\mathrm{Wcm^{-2}K^{-2}}$*, running for 100 s with* $\Delta t = 10^{-3}$ *s.*

From this analysis, we determine how many modi, snapshots and training runs we need in our training set to create the optimal ROM. These results are gathered in Table 5.5.

Table 5.5: The smallest maximum absolute difference in temperature between the sROM and our FOM and the lowest RMS error of the fast flux and of the temperature, with the amount of snapshots and modi used to get those values.

	3 Training runs			5 Training runs		
	Value	r	k	Value	r	k
RMS error of Fast flux ($\times 10^{-4}$)	2.5	6	8175	2.4	6	8175
RMS error of Temperature ($\times 10^{-4}$)	1.3	10	25000	0.94	10	25000
Maximum ΔT (K)	9.2	8	25000	7.8	8	25000

We see that we get the best results if we use a small amount of modi and train for a long amount of time. We also note that the maximum temperature difference is considerably larger compared to the average RMS error of the temperature.

Training analysis of the sROM

The mROM comprises two different ROMs. In this test we are looking at the influence of a parameter in our temperature model. Therefore we keep the ROM_N the same as in Section 5.3.1, and change the amount of modi and snapshots in ROM_T. We perform the same analysis as for the mROM. The sROM shows a lot of instability. In figure 5.29 we see the resultant RMS error of the fast flux using 5 training sets. The other results for 3 training sets, and the RMS error for temperature give the same results. If an area is white, it means that the sROM broke down. This breaking down can be attributed to the temperature interpolation step in ROM_N. We identified 3 factors that could contribute to this. First of all, an incomplete training set of temperature for the ROM_N, increasing the amount of training runs in the temperature training set may increase the quality of the ROM_N. Second it may be possible that we selected a sub optimal number of modi in constructing the ROM_N and snapshots in training the ROM_N, resulting in a less qualitative ROM_N, note that Figure 5.29 shows the amount of modi and snapshots for ROM_T. Lastly errors may have occurred in the time stepping in the training analysis. Unfortunately solving these issues was beyond the time scope of this research.

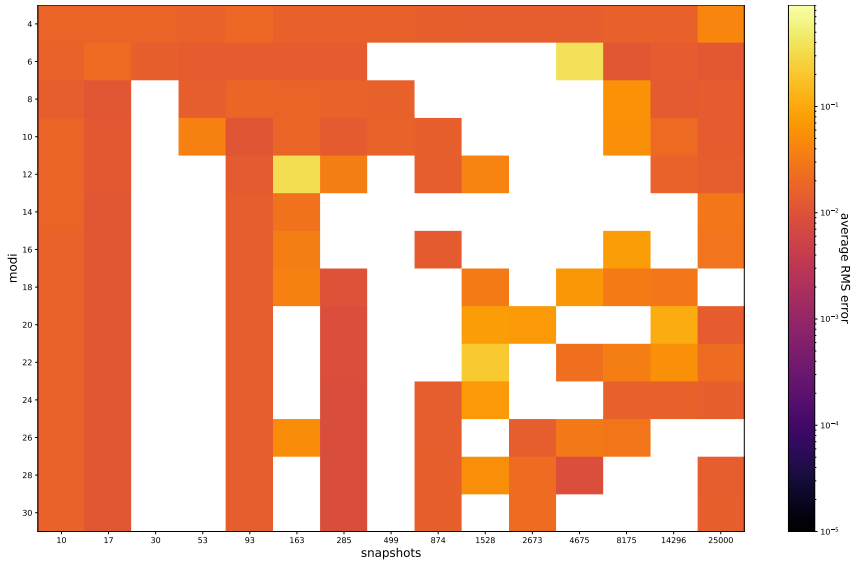


Figure 5.29: The RMS error between the fast flux of the sROM and the fast flux of our FOM. Here we use a training set with 5 different heat transfer coefficient values equidistant between 0 and the nominal value of 0.03 $\mathrm{Wcm^{-2}K^{-2}}$ to simulate a DLOFC. Every run is 25 s long with $\Delta t = 10^{-3}$s. The resulting RMS error in fast flux is for a test with a heat transfer coefficient of 0.00375 $\mathrm{Wcm^{-2}K^{-2}}$, running for 100 s with $\Delta t = 10^{-3}$ s.

Because these results were unsatisfactory, we were unable to retrieve the ideal training case.

5.3.3 Comparative case

From Section 5.3.2 we find the best training set with which we can train our mROM. For the sROM the training was inconclusive. Therefore for our comparative case, we use the same amount of modi and snapshots as with the mROM. In the comparative case we use a training set with 5 training runs ($\zeta = [0, 0.25, 0.5, 0.75, 1]$). The training lasts for 50 s, and we use $\Delta t = 10^{-3}$ s. We test the ability of the ROM to estimate for a ζ not in the training set, by running for 200 s at $\Delta t = 10^{-3}$ s, using $\zeta = 0.125$. We also look at the the abilities of the two ROM methods to predict in time, without having to estimate for a ζ, using $\zeta = 0.25$. The ROMs are constructed with 6 modi and use 50000 snapshots of each training run.From this testing we retrieve the final profile of the fluxes, the RMS error in time of the fast flux and of the temperature, and the maximum temperature differences in time, positive and negative.

Comparative case mROM

Figure 5.30 shows the profiles of the fluxes and the temperature after 200 s.

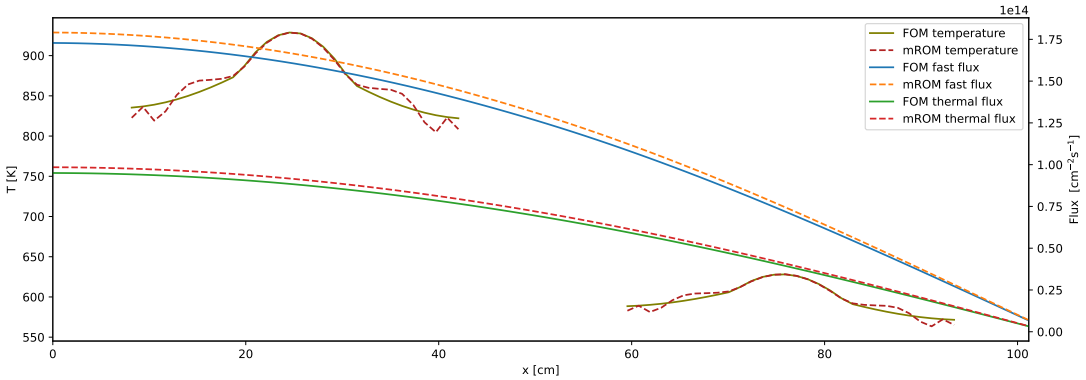
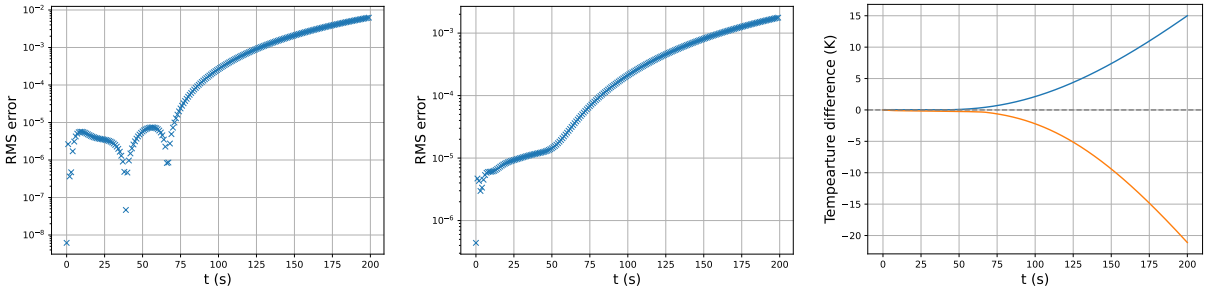


Figure 5.30: Profiles of the fluxes and temperature of the mROM and our FOM. The mROM training set consisted of 5 different heat transfer coefficient values equidistant between 0 and 0.03 $\mathrm{Wcm^{-2}K^{-2}}$ to simulate a DLOFC, all training runs lasted for 50 s at $\Delta t = 10^{-3}$ s, of which we took 50000 snapshots. We tested the mROM with a heat transfer coefficient of 0.00375 $\mathrm{Wcm^{-2}K^{-2}}$, predicting for 200 s with $\Delta t = 10^{-3}$ s.

We see that the mROM can accurately predict the temperature in the fuel region of the fuel blocks, but shows noise in the temperature in the graphite parts. We also see that the mROM overestimates the fluxes. In Figure 5.31 we see the RMS error in time of the fast flux and of the temperature as well as the maximum temperature difference, both positive and negative, between the mROM and the FOM.



(a) RMS error between the fast flux as determined by the mROM, and our FOM.

(b) RMS error between the temperature as determined by the mROM, and our FOM.

(c) Maximum over- and underestimate of temperature between the mROM and our FOM. The blue line shows the maximum overestimate, and the orange line the maximum underestimate. The grey dotted line signifies that $\Delta T = 0$ K.

Figure 5.31: Analysis of various errors between our mROM and our FOM. The mROM training set consisted of 5 different heat transfer coefficient values equidistant between 0 and 0.03 $\mathrm{Wcm^{-2}K^{-2}}$ to simulate a DLOFC, all training runs lasted for 50 s at $\Delta t = 10^{-3}$ s, of which we took 50000 snapshots. We tested the mROM with a heat transfer coefficient of 0.00375 $\mathrm{Wcm^{-2}K^{-2}}$, predicting for 200 s with $\Delta t = 10^{-3}$ s.

The mROM seems to be able to predict rather well in time estimating for a ζ not in its training set. Only after predicting at times beyond 300% of the training time, did the maximum underestimation exceeds 10 K, for the maximum overestimation this was at times beyond 320% of the training time. The computation time of the FOM was 4.78 times longer compared to the computation time of the mROM to compute this comparative case.

Interestingly, focusing on Figure 5.30 we see that the mROM quite accurately predicts the temperature in the fuel channels of the fuel block. If we look at the average RMS of only these fuel channels, shown in Figure 5.32, we see that the resulting average RMS error is an order of magnitude lower compared to the total average RMS error of the temperature. It remains lower until $t = 50$ s, after which it quickly increases to the same levels as in Figure 5.31b, but is in the end trending down. Finding the exact reason for this behaviour unfortunately was beyond the time frame of this thesis.

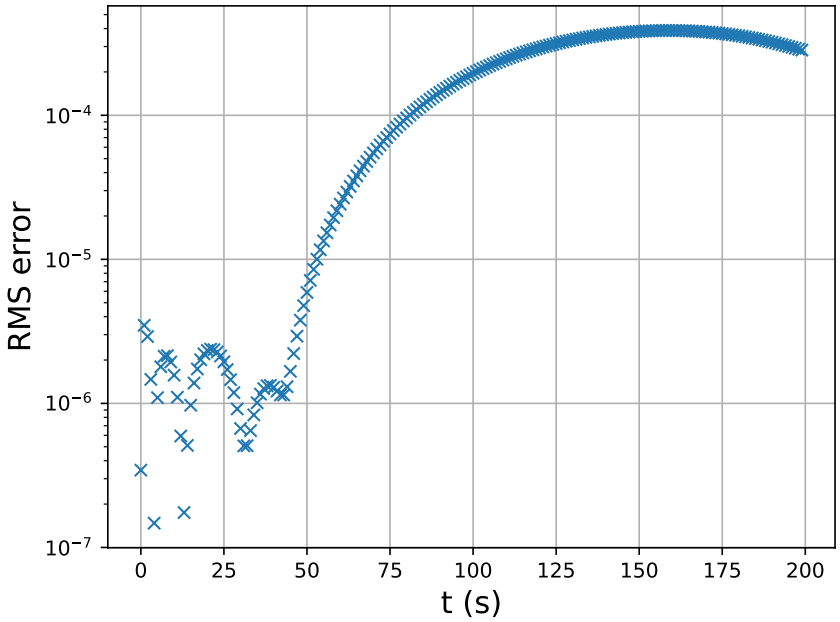


Figure 5.32: RMS error between the temperature of the fuel channel as determined by the mROM, and our FOM. The mROM training set consisted of 5 *different heat transfer coefficient values equidistant between 0 and 0.03* $\mathrm{Wcm^{-2}K^{-2}}$ *to simulate a DLOFC, all training runs lasted for 50 s at* $\Delta t = 10^{-3}$ *s, of which we took 50000 snapshots. We tested the mROM with a heat transfer coefficient of 0.00375* $\mathrm{Wcm^{-2}K^{-2}}$*, predicting for 200 s with* $\Delta t = 10^{-3}$ *s.*

Comparative case sROM

Figure 5.33 shows the profiles of the fluxes and the temperature of the sROM after 200 s.

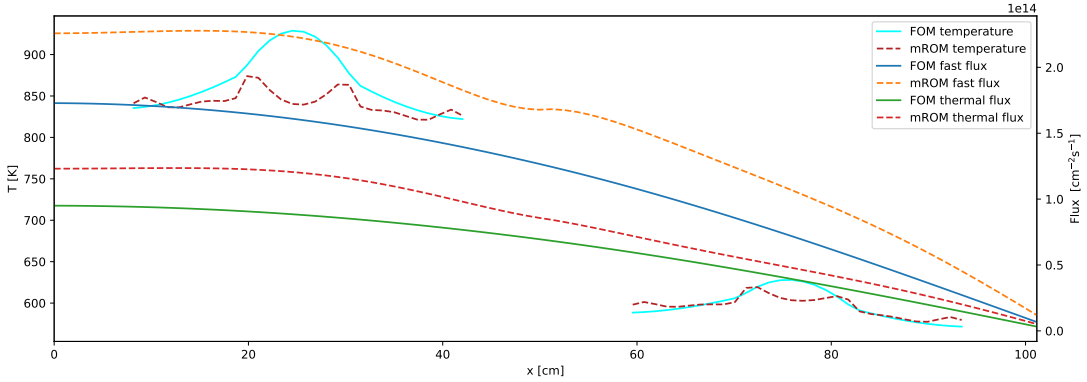
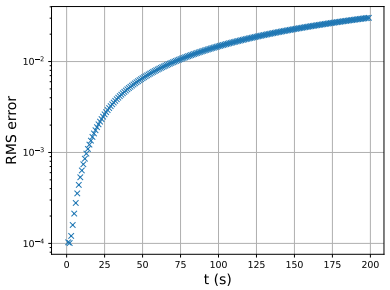
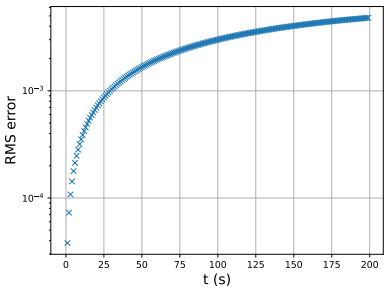


Figure 5.33: Profiles of the fluxes and temperature of the sROM and our FOM. The training set of ROM_T of the sROM consisted of 5 different heat transfer coefficient values equidistant between 0 and 0.03 $\mathrm{Wcm^{-2}K^{-2}}$ to simulate a DLOFC, all training runs lasted for 50 s at $\Delta t = 10^{-3}$ s, of which we took 50000 snapshots. We tested the sROM with a heat transfer coefficient of 0.00375 $\mathrm{Wcm^{-2}K^{-2}}$, predicting for 200 s with $\Delta t = 10^{-3}$ s.

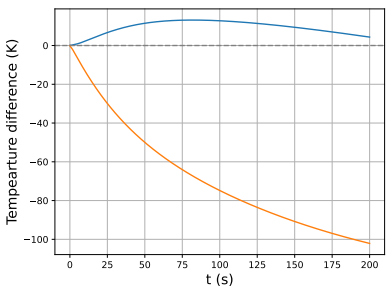
We see that the sROM produces quite a bit of noise in the temperature, especially in the fuel channel of the fuel block. Furthermore we observe that it overestimates the fluxes. In Figure 5.27 we see the RMS error in time of the fast flux and the temperature, determined through equation 3.2, as well as the maximum temperature difference, both positive and negative, between the sROM and the FOM.



(a) RMS error between the fast flux as determined by the sROM, and our FOM.



(b) RMS error between the temperature as determined by the sROM, and our FOM.



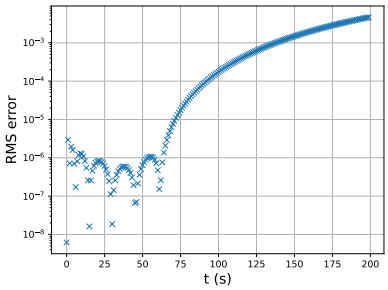
(c) Maximum over- and underestimate of temperature between the sROM and our FOM. The blue line shows the maximum overestimate, and the orange line the maximum underestimate. The grey dotted line signifies that $\Delta T = 0$K.

Figure 5.34: Analysis of various errors between our sROM and our FOM. The training set of ROM_T of the sROM consisted of 5 different heat transfer coefficient values equidistant between 0 and 0.03 $\mathrm{Wcm^{-2}K^{-2}}$ to simulate a DLOFC, all training runs lasted for 50 s at $\Delta t = 10^{-3}$ s, of which we took 50000 snapshots. We tested the mROM with a heat transfer coefficient of 0.00375 $\mathrm{Wcm^{-2}K^{-2}}$, predicting for 200 s with $\Delta t = 10^{-3}$ s.

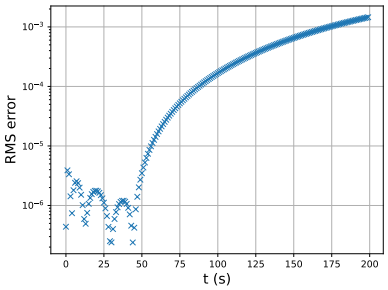
From Figure 5.34c we see that the sROM starts diverge from the FOM immediately, with the maximum temperature difference exceeding 10K after 10s, which is well within the training runs. The computation time of the FOM was 3.97 times longer compared to the computation time of the sROM to compute this comparative case.

Predicting in time with the mROM

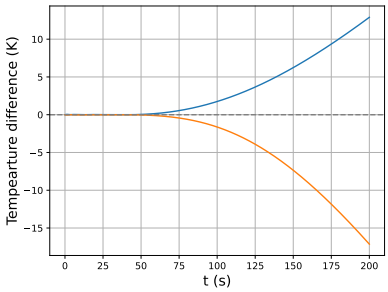
In Figure 5.35 we see the resulting RMS errors and maximum temperature difference if we use a ζ value included in our training for our mROM.



(a) RMS error between the fast flux as determined by the mROM, and our FOM.



(b) RMS error between the temperature as determined by the mROM, and our FOM.



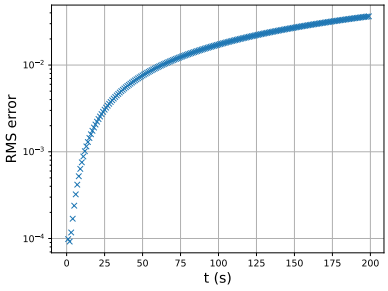
(c) Maximum over- and underestimate of temperature between the sROM and our FOM. The blue line shows the maximum overestimate, and the orange line the maximum underestimate. The grey dotted line signifies that $\Delta T = 0K$.

Figure 5.35: Analysis of various errors between our mROM and our FOM. The mROM training set consisted of 5 different heat transfer coefficient values equidistant between 0 and 0.03 $\mathrm{Wcm^{-2}K^{-2}}$ to simulate a DLOFC, all training runs lasted for 50 s at $\Delta t = 10^{-3}$ s, of which we took 50000 snapshots. We tested the mROM with a heat transfer coefficient of 0.00375 $\mathrm{Wcm^{-2}K^{-2}}$, predicting for 200 s with $\Delta t = 10^{-3}$ s.

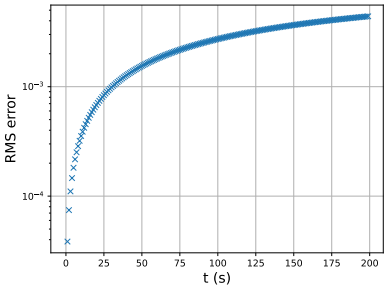
We see that the RMS errors of Figure 5.35 are only slightly lower compared to those in Figure 5.31. This hints towards the error of the mROM not coming from estimation of the heat transfer coefficient, but more from the amount of modi and snapshots used to create the mROM.

Predicting in time with the sROM

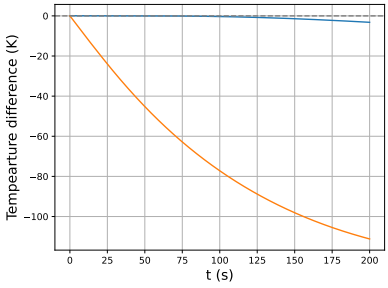
Figure 5.36 shows the resulting RMS errors and maximum temperature difference if we use a ζ value included in our training for our mROM.



(a) RMS error between the fast flux as determined by the sROM, and our FOM.



(b) RMS error between the temperature as determined by the sROM, and our FOM.



(c) Maximum over- and underestimate of temperature between the sROM and our FOM. The blue line shows the maximum overestimate, and the orange line the maximum underestimate. The grey dotted line signifies that $\Delta T = 0K$.

Figure 5.36: Analysis of various errors between our sROM and our FOM. The training set of ROM_T of the sROM consisted of 5 different heat transfer coefficient values equidistant between 0 and 0.03 $\mathrm{Wcm^{-2}K^{-2}}$ to simulate a DLOFC, all training runs lasted for 50 s at $\Delta t = 10^{-3}$ s, of which we took 50000 snapshots. We tested the mROM with a heat transfer coefficient of 0.0075 $\mathrm{Wcm^{-2}K^{-2}}$, predicting for 200 s with $\Delta t = 10^{-3}$ s.

We do not see any improvement comparing Figure 5.34 to Figure 5.36, from which we gather that the error is mostly caused by the interpolation of the temperature field in the ROM_N.

Chapter 6

Discussion

In this thesis we investigated whether it was possible to create a predictive non-intrusive reduced order model of a high-temperature gas-cooled reactor. To accomplish this we first constructed a representative high-fidelity model, with which we could create and test ROMs. To verify that the representative model worked well, numerous tests were performed to verify its convergence to analytical and pseudo analytical solutions. Next, ROMs were made using this representative model. These ROMs were subject to numerous experiments in order to ascertain their accuracy. In this chapter we shall discuss the tests done on the representative model, and subsequently the experiments done with the ROMs.

6.1 Representative High-Fidelity Model

The created representative model from Chapter 3 was subject to the tests of section 5.1. These tests are split between spatial an temporal convergences.

Spatial verification of the representative model

The convergence of the RMS error as a function of Δx for our temperature model is not as expected. Taking Figure 5.2, we see that for $\Delta x < 10^{-5}$ cm the RMS error between the analytical solution and our numerical solution goes with Δx^2. However, at higher Δx this convergence is lower, averaging about 1.5. Moreover the RMS error wobbles, going from 1.1 to 1.9 to 1.5 repeating until Δx has reached 10^{-5} cm. A satisfactory answer for this behaviour has not yet been found. The fact that the RMS error does not converge with Δx^2 for all Δx points towards an error in the implementation of the FVM from Section 3.1.2 was only found towards the end of this thesis, therefore unfortunately there was not enough time to fully investigate the source of this error. It is suspected that the error comes from the fact that the temperature profile is not yet fully resolved at larger Δx.

To verify our FVM implementation of the neutronics model we looked at the convergence of the RMS error for the fast flux profile, and the k_{eff} term. From Figures 5.4 and 5.5 we see that whereas the RMS error convergence of k_{eff} has the expected convergence of Δx^2 until $\Delta x = 10^{-3}$cm, the fast flux RMS error only converges with $\Delta x^{1.5}$. Both start to flatten out if $\Delta x < 10^{-3}$cm. Just as with temperature, this error was only found towards the end of thesis, thereby the full analysis of this error was beyond the time frame of the thesis.

We found that the FVM was not implemented correctly in this thesis. However the results of the thesis in creating a reduced order model still hold. First of all we argue that even though the spatial convergence of the neutronics and temperature models is slower than we expected, they do still converge, meaning that if we choose our Δx small enough, we will retrieve the solution as expected from other high-fidelity codes. Second of all, in creating a ROM we are using our own FOM for training data, and subsequently testing the created ROM with this same FOM. In the end it does not matter too much if the underlying FOM has errors, as long as the ROM produced from the FOM gives the same errors.

Temporal verification of the representative model

In our temperature model the RMS error convergence as a function of Δt corresponds with the theoretical expected RMS error convergence from first order BDF, as explained in Section 3.1.3. In Figure 5.3 we see that the RMS error converges with Δt between $\Delta t = 10^{-1}$ s and 10^{-4} s, as expected. If we decrease Δt further this convergence seems to increase, going with 1.8 between $\Delta t = 10^{-4}$ s and 10^{-5} s. We expect that this is due to the fact that the induced transient is very small, and that our temperature model for smaller Δt can better, and thus more quickly retrieve the actual solution.

We have seen that the Δt RMS error convergence of the fast flux goes with Δt between $\Delta t = 10^{-3}$ s and 10^{-6} s in Figure 5.6. The reason the flux does not converge with the expected Δt before 10^{-3} s, is because at these low Δt the prompt jump as a result of the induced transient is not fully resolved, in other words, at these high Δt our numerical model does not observe a phenomenon that occurs, while at low Δt the model can observe that phenomenon. This is not seen in the temperature model verification, because an induced transient in temperature does not cause a prompt jump.

For the coupled temperature and neutronics model we verified that the RMS error convergence goes linear with Δt in Figure 5.9. Just as with our neutronics model the convergence goes with Δt if $\Delta t < 10^{-3}$ s, which is due to the prompt jump.

6.2 Reduced order model accuracy

For the reduced order model we will discuss the results of the experiments on the four distinct ROMs, ROM_T, ROM_N, mROM, and sROM which we created with OpInf.

$\mathbf{ROM_N}$ accuracy

Using only two fuel blocks we found that our approach to a ROM_T produces very accurate results for power inputs not included in the training set. In Figure 5.13 we have seen that we can achieve an average RMS error of 10^{-10} if we use 20 modi and 2010 snapshots. We take a ROM to be sufficiently accurate if its average RMS error is around 10^{-5}, therefore the created ROM is sufficiently accurate. In Section 4.4.3we have extensively tested this training regime for a single test case. In Section A.5 we tested the created ROM for a 1000 random power inputs. As the ROM_T preformed rather well, not having a RMS error above 3×10^{-5} with 12 modi and 1500 snapshots if we predict 4000 seconds into the future, we are confident that we created a predictive non-intrusive ROM_T.

$\mathbf{ROM_N}$ accuracy

In ROM_N we used temperature as a parameter. ROM_N determined the operators associated with the temperatures in the training set, and estimated using interpolation for temperatures outside this training set. We determined the efficacy of different 1 dimensional interpolation techniques. We see in Figures 5.15a and 5.15b that the accuracy difference between the cubic and linear interpolation is negligible. Therefore we prefer to use the cubic interpolation technique if we have a 1 dimensional parameter. We also note that the best results are retrieved with 1 mode. This is most likely due to the fact that this one mode resembles the steady state shape of the fluxes and delayed neutrons. When we change the temperature homogeneous over the entire core, the shape does not change locally, but increases or decreases globally.

We also saw that if we preprocess the training set, it negatively impact the results for a homogeneous temperature. However, not preprocessing does increase the instability of the resultant ROM_N, especially if we define more temperature zones. Therefore it has been chosen to preprocess the training set, also for homogeneous temperature.

If we change the amount of training runs for the homogeneous temperature we observe two different behaviours. At $T > 1500$ K adding more training slightly reduces the RMS error, as we could expect, because the distance between estimation point and training point decreases. However at $T < 1500$ K

we observe two distinct behaviours. First of all, if our training set contains 5 temperatures our ROM can no longer predict around 700K. Moreover the results for 9 training runs are worse than those for 7 or 3. We suspect the main reason for this is the values within the operators, in the training set with 5 and 9 temperatures, we add an operator at $T = 846$ K which has distinctly smaller entries, due to it being close to the steady state. The resultant ROM_N most likely overfits for this operator, meaning that for temperatures below the steady state temperature, where we would expect the fluxes to increase, the interpolation estimates an operator with decreasing fluxes, because the nearest training run has decreasing fluxes. This is not replicated if we use 7 training runs in our training set, as that training set does not include a temperature close to $T = 800$ K. Furthermore we see that our neutronics produces a steady state around $T = 800$ K. If we go above that temperature the fluxes generally decrease, while if we go below it the fluxes increase. As it is impossible for the fluxes to decrease below zero, a system with decreasing fluxes is more stable, while if the fluxes increase the ROM might overpredict this increase, making results blow up. From this we argue that any ROM_N performs best if the underlying and resultant dynamics are relatively stable.

Of note is the fact that we can observe behaviour associated with low Δt while we run our neutronics at high Δt. From Figure 5.19 we observe that a ROM trained at $\Delta t = 10^{-4}$ s and running at $\Delta t = 10^{-2}$ s produces the results as if it were running at $\Delta t = 10^{-4}$. It observes a prompt jump while a FOM running at $\Delta t = 10^{-2}$ s cannot. The reason for this behaviour was not fully researched, because in the end the computational profit of a ROM comes from its reduction of the spatial domain, not the temporal domain.

In the investigation of ROM_N with two fuel blocks we have found that the accuracy of the ROM_N mostly increases if we use more snapshots, and not necessarily if we use more modi. The reason for this is that most of the dynamics of the system are already caught in the first 5 modi, as seen from Figure 5.21. Adding more modi mostly adds noise to the ROM. Making Δt smaller also increases the RMS error, this is to be expected for a similar reason as with the Δt convergence of the representative model. At higher Δt our ROM_N and the FOM to which we compare the ROM do not yet resolve the prompt jump, whereas at lower Δt they do. If the prompt jump is included ROM_N needs to take more dynamics into account, resulting in higher RMS errors compared to the FOM.

Looking at the entire temperature domain we do not observe similar behaviour as in Figure 5.17, this is most likely because we use more training runs in the entire temperature domain, meaning the ROM_N does not overfit for operators close to a steady state. We observe that the blue region of accuracy in Figure 5.22 corresponds to solutions resembling the steady state. This blue region is not symmetrical, as the reactor itself, and its boundary conditions are not symmetrical.

Coupled ROM

For the coupled ROM we devised two strategies, the merged ROM and the sequential ROM.

From our preliminary test we can conclude that both ROM strategies work, and have workable outputs. We do see that the mROM performs better compared to the sROM. The RMS error for both the fast flux and the temperature is an order of magnitude smaller. Furthermore the maximum temperature difference, both positive and negative is an order of magnitude lower. In Figures 5.25c and 5.27c, we see that the mROM tends to underestimate the temperature, while the sROM tends to overestimate the temperature. We would prefer if our ROM would overestimate the temperature, as this would provide an extra safety margin if we were to use the ROM in real life applications. Lastly we observe more noise in the sROM's temperature profile, especially in the second fuel block.

Looking at the analysis of snapshots and modi we find that the mROM works best if we provide it with more snapshots. It should be noted that the strategy of snapshotting employed here is not very efficient. We snapshotted subsequent time steps. In literature there are other methods such as adaptive sampling [14] and using time steps with different Δt [25]. Using these techniques would likely lower the required amount of snapshots. It is interesting to note that the best RMS error and least difference in temperature is already achieved using only 6 modes. In Figure 5.13 we see that for the

temperature using more modes greatly increases the accuracy of the acquired ROM. This behaviour is not replicated in the coupled model. This may be due to the fact that the results of Figure 5.13 were achieved after predicting for 4000 time steps, whereas the results of Figure 5.5 where obtained after predicting for 100000 time steps. When taking a few time steps noise may tot yet develop a prominent role, whereas when we take more time steps this behaviour becomes more pronounced. Another reason might be that the system is more influenced by the neutronics, rather than temperature. If we look at Figure 5.15a, we see that ROM_N performs better with fewer modi.
The sROM failed to provide useful results in analysing the influence of snapshots and modi on the accuracy of the acquired ROM. The sROM analysis continuously broke down in the step where we estimate temperature not included in the training set. A possible reason for this is the fact that in interpolating the operators we interpolate over all elements of the operator, where the resulting interpolated value may not make sense. If our ROM_N then uses this temperature the resulting flux may blow up, giving rise to ever higher temperatures. As our ROM_N interpolates for temperatures $T \in [294, 2500]$ K, if the temperature exceeds 2500 K, or goes below 294 K the interpolation will return NANs. Another reason is that a mistake was made in the time stepping scheme of the temperature calculation.

When we look at the comparative case, we see that the mROM performs better than the sROM. We observe that the sROM gives great instability in the temperature profile as well as the flux profile, whereas the mROM only shows instability in its temperature profile, moreso in the graphite channel than in the fuel channel. In fact we have found that the RMS error of temperature in the fuel channel is an order of magnitude lower than the RMS error of the temperature in the graphite channels. Why the prediction of the temperature of the fuel channel is more accurate is not yet known. We see that both the mROM and the sROM tend to underestimate the temperature and simultaneously that both overestimate the fluxes. A reason for the comparably worse performance of the sROM may be due to the fact that we could not find, and thus use, an ideal combination of modi and snapshots for the sROM. Interestingly the resultant RMS errors for both the mROM and the sROM were orders of magnitude higher than when we looked at the preliminary test, even though both tests were done predicting in time frames 4 times further than the training regime. We suspect this is because if we use a long time frame, the resultant dynamics of the FOM and the ROM have risen or fallen more compared to a short time frame.

The mROM obtained an increase in computational speed 4.78 faster than the FOM. This is lower than expected. Our FOM needs to do 3 multiplications with matrices of size 78, corresponding to the sizes for the fast and thermal flux, as well as the precursor population, and 1 calculation with a matrix of size 60, corresponding to the spatial size of the temperature model. The mROM needs to do a calculation with a 6 by 6 matrix, and a 21 by 21 matrix, respectively the $\mathbf{A}$ and $\mathbf{F}$ matrices of equation 4.17. A reason for this might be that OpInf uses in build functions, which might add computation time, while these functions may not be ideal or necessary for our problem set.

If we compare the mROM predicting in time, using a heat transfer coefficient of the training set, to predicting in time while also estimating for a heat transfer coefficient not in the training set, we see an order of magnitude improvement of the RMS errors of the fast flux and the temperature. From Figure 5.35b we see that the RMS error in temperature immediately increases if the mROM starts to time step beyond its training data, reaching a value comparable to the RMS error of Figure 5.31b after 75 s, with the RMS error in fast flux showing similar behaviour, lagging slightly behind the RMS error of the temperature. The maximum temperature difference is slightly lower if we do not have to estimate for the value of the heat transfer coefficient. Moreover the maximum temperature starts to diverge right after the training has ended, whereas the RMS error takes a bit longer.
For the sROM we do not observe any improvement in the same comparison, which might be due to the fact that the bottleneck in prediction comes from the interpolation in temperature parameter space, not from estimation of the heat transfer coefficient.

Chapter 7

Conclusions

In this thesis we set out to create a predictive non-intrusive reduced order model of a high-temperature gas-cooled reactor. This has been achieved after creating a representative model. On this representative model we applied and tested Operator Inference in creating separate ROMs of the temperature and neutronics model. In the end we developed and tested two different ways of creating a reduced order model of a coupled neutronics and temperature model, also using Operator Inference.

The representative model showed inconsistencies in its spatial discretization compared to what we would expect from the implementation of the finite volume method. However, we still found that our numerical methods would converge to an analytical, or pseudo analytical solution. Moreover, the temporal discretization produced results which are consistent with what we would expect from the first order backward differentiation formula.

Given these facts, we concluded that our representative model could accurately represent the physics of a high-temperature gas-cooled reactor.

We constructed a reduced order model (ROM) of the temperature code of our representative model using Operator Inference. From our analysis we can conclude that this created ROM accurately predicts in time for any sort of power input, with RMS errors below 3×10^{-5}. This was found to be the case if we used three distinct power inputs in training, snapshotting them for 1500 seconds, after which we trained our ROM with 5 modi and predicted for 4000 seconds in time.

From the analysis of our constructed neutronics ROM, we found that if we have to estimate for a homogeneous temperature not included in training, there is no discernible difference between a cubic and a linear interpolation technique. Next we can conclude that for a homogeneous temperature we need to have training runs the same distance both above and below the steady state temperature. Otherwise the resultant interpolation returns non-physical results. Furthermore we concluded that the acquired ROM could predict a prompt jump with a large time step, even though our representative model using a similar time step could not predict the prompt jump. We found this to be the case as long as the ROM was trained with the prompt jump. We also observed that a neutronics ROM could predict resulting fluxes if it had to estimate for two temperatures within the reactor. We conclude that the resulting predicted flux is of a greater quality close to a steady state temperature distribution, and of a lesser quality far away from a steady state temperature distribution compared to the homogeneous temperature. We found this to be due to the included training runs in the training set.

The goal of this thesis was to create a reduced order model of a coupled neutronics and temperature model. We have tested two different ways of creating such a ROM using Operator Inference, the so-called merged ROM (mROM) and the sequential ROM (sROM). The merged model outperformed the sequential model in every aspect we tested. The created sequential model tended to break down, due to incorrectly estimating for the neutron fluxes of the two temperature zones within the reactor. Looking at individual cases we found that the predicted temperature and flux profiles of the sROM would immediately diverge from the expected FOM profiles. However the resultant mROM could not provide a perfect fit. If we predict beyond our training data the prediction of temperature generally underestimated the actual temperature, and showed noise development in the graphite channel in the fuel block. Furthermore we found that the time saved was significantly less than we would expect either from literature or from looking at the size of the operators. Beyond the error consideration, we conclude that an mROM is more versatile than its sROM counterpart. This is due to the fact that an mROM does not have to estimate for temperatures, meaning that the training of the mROM is simplified, and that it will not break down if it needs to predict using a temperature outside of the temperature with which we can train an sROM. We should however also conclude that the sROM does allow for individual fine-tuning of the ROM_T and ROM_N, which could aid in the quality of the sROM.

In conclusion, it has been found that creating a predictive non-intrusive reduced order model of a high-temperature gas-cooled reactor is possible, if we use Operator Inference in creating a merged reduced order model, and as long as we supply it with sufficient training data.

7.1 Recommendations

In the desire to implement the achieved reduced order model on high-fidelity models both in- and out-house, a number of recommendations for future research are presented.

The first recommendation in further improvement of the merged model is to change the snapshotting method. In this research we snapshotted subsequent time steps. From literature we found iterative snapshot methods, or snapshots methods using snapshots with differing Δt. These could be interesting, as we need more snapshots in the beginning, to account for the prompt jump, after which we can use larger time steps. Such snapshotting schemes have not been investigated in this thesis, but are possible using Operator Inference.

Another recommendation would be to investigate other types of interpolation when interpolating p dimensional parameters. This would mainly benefit the sequential model, whose results may than be on par with the merged model or even exceed the merged model in quality. Another point in which the sROM could be improved, is by investigating whether it is necessary to map the output of the individual ROM_N and ROM_T back to the original dimensionality. If we do not have to map back, we will save computation time. Furthermore, it would be interesting to perform a full investigation into the effect of different input variables for both the ROM_T and the ROM_N, to see if we can fine-tune them to each other, and in that way if it would be possible to create an optimal case in which the sROM works.

We would also recommend to research the length of training time necessary to predict to t_{final}. We have found that training for 1 s and testing for 4 s netted significantly better results than training for 50 s and testing for 200 s, even though for both cases we tested to 400% of the training time. We found this to be due to a difference in the dynamics of the training set. If we want to implement the reduced order model in a high-fidelity model, but are only interested in predicting up to 100 seconds in time, we should find the ideal amount of snapshots and modi to create a ROM for those dynamics.

Another recommendation would be to compare the merged model using Operator Inference to a similar model using Sparse identification of Non-linear Dynamics, to determine which works better for the high-temperature gas-cooled reactor.

Bibliography

[1] Integration and ODEs (scipy.integrate) — SciPy v1.13.1 Manual, . URL https://docs.scipy.org/doc/scipy/reference/integrate.html.

[2] Miip004 u battery for marine propulsion, . URL https://maritiemland.nl/miip004-u-battery-for-marine-propulsion/.

[3] Operator Inference in Python, . URL https://willcox-research-group.github.io/rom-operator-inference-Python3/source/index.html.

[4] SCALE code system | ORNL, . URL https://www.ornl.gov/onramp/scale-code-system.

[5] scipy.interpolate.CubicSpline — SciPy v1.13.0 Manual, . URL https://docs.scipy.org/doc/scipy/reference/generated/scipy.interpolate.CubicSpline.html#scipy.interpolate.CubicSpline.

[6] scipy.interpolate.LinearNDInterpolator — SciPy v1.13.0 Manual, . URL https://docs.scipy.org/doc/scipy/reference/generated/scipy.interpolate.LinearNDInterpolator.html#scipy.interpolate.LinearNDInterpolator.

[7] Statement: U-Battery | News | Urenco, . URL https://www.urenco.com/news/global/2023/statement-u-battery.

[8] Taylor series, . URL https://en.wikipedia.org/w/index.php?title=Taylor_series&oldid=1221317435. Page Version ID: 1221317435.

[9] The Paris Agreement | UNFCCC, 2015. URL https://unfccc.int/process-and-meetings/the-paris-agreement.

[10] International shipping, 2023-07-11. URL https://www.iea.org/energy-system/transport/international-shipping.

[11] How much does the shipping industry contribute to global co2 emissions?, 2023-09-22. URL https://sinay.ai/en/how-much-does-the-shipping-industry-contribute-to-global-co2-emissions/. Section: GHG Emissions.

[12] Heat equation, May 2024. URL https://en.wikipedia.org/w/index.php?title=Heat_equation&oldid=1222010063. Page Version ID: 1222010063.

[13] H. v. d. Akker and R. F. Mudde. *Fysische Transportverschijnselen: Denken in Balansen*. TU Delft OPEN Textbooks, Apr. 2023. ISBN 978-94-6366-683-1. doi: 10.5074/t.2023.002. URL https://textbooks.open.tudelft.nl/textbooks/catalog/book/58. Publication Title: TU Delft OPEN Textbooks.

[14] F. Alsayyari, M. Tiberga, Z. Perkó, J. L. Kloosterman, and D. Lathouwers. Analysis of the Molten Salt Fast Reactor using reduced-order models. *Progress in Nuclear Energy*, 140:103909, Oct. 2021. ISSN 0149-1970. doi: 10.1016/j.pnucene.2021.103909. URL https://www.sciencedirect.com/science/article/pii/S0149197021002729.

[15] W. S. Broecker. Climatic change: are we on the brink of a pronounced global warming? *Science (New York, N.Y.)*, 189(4201):460–463, Aug. 1975. ISSN 0036-8075. doi: 10.1126/science.189.4201.460.

[16] S. L. Brunton and J. N. Kutz. *Data-Driven Science and Engineering: Machine Learning, Dynamical Systems, and Control*. Cambridge University Press, Cambridge, 2019. doi: 10.1017/9781108380690. URL https://www.cambridge.org/core/books/datadriven-science-and-engineering/77D52B171B60A496EAFE4DB662ADC36E.

[17] M. D. Carelli, P. Garrone, G. Locatelli, M. Mancini, C. Mycoff, P. Trucco, and M. E. Ricotti. Economic features of integral, modular, small-to-medium size reactors. *Progress in Nuclear Energy*, 52(4):403–414, May 2010. ISSN 0149-1970. doi: 10.1016/j.pnucene.2009.09.003. URL https://www.sciencedirect.com/science/article/pii/S0149197009001474.

[18] M. Ding and J. Kloosterman. Thermal-hydraulic design and transient evaluation of a small long-life HTR. *Nuclear Engineering and Design*, 255:347–358, 2013. ISSN 0029-5493.

[19] M. Ding, J. L. Kloosterman, T. Kooijman, and R. Linssen. Design of a U-Battery®, Nov. 2011. URL https://www.janleenkloosterman.nl/reports.php.

[20] J. J. Duderstadt, Duderstadt, and E. Hamilton. *Nuclear Reactor Analysis*. New York, Jan. 1976. ISBN 978-0-471-22363-4.

[21] R. Elzohery and J. Roberts. EXPLORING TRANSIENT, NEUTRONIC, REDUCED-ORDER MODELS USING DMD/POD-GALERKIN AND DATA-DRIVEN DMD. *EPJ Web of Conferences*, 247:15019, 2021. ISSN 2100-014X. doi: 10.1051/epjconf/202124715019. URL https://www.epj-conferences.org/articles/epjconf/abs/2021/01/epjconf_physor2020_15019/epjconf_physor2020_15019.html. Publisher: EDP Sciences.

[22] I. Farcas, R. Munipalli, and K. E. Willcox. On filtering in non-intrusive data-driven reduced-order modeling. In *AIAA AVIATION 2022 Forum*. American Institute of Aeronautics and Astronautics. doi: 10.2514/6.2022-3487. URL https://arc.aiaa.org/doi/abs/10.2514/6.2022-3487. _eprint: https://arc.aiaa.org/doi/pdf/10.2514/6.2022-3487.

[23] Fernando Zigunov. Understanding POD: the Proper Orthogonal Decomposition, 2021. URL https://www.youtube.com/watch?v=axfUYYNd-4Y.

[24] N. Ghantous. Can electric boats decarbonise shipping?, Oct. 2022. URL https://www.energymonitor.ai/tech/electrification/can-electric-boats-decarbonise-shipping/.

[25] Z. K. Hardy and J. E. Morel. Partitioned Dynamic Mode Decomposition for Nonlinear or Transient Dynamics.

[26] O. Issan and B. Kramer. Predicting Solar Wind Streams from the Inner-Heliosphere to Earth via Shifted Operator Inference. *Journal of Computational Physics*, 473:111689, Jan. 2023. ISSN 00219991. doi: 10.1016/j.jcp.2022.111689. URL http://arxiv.org/abs/2203.13372. arXiv:2203.13372 [physics].

[27] K. Kaheman, J. N. Kutz, and S. L. Brunton. SINDy-PI: a robust algorithm for parallel implicit sparse identification of nonlinear dynamics. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 476(2242):20200279, Oct. 2020. doi: 10.1098/rspa.2020.0279. URL https://royalsocietypublishing.org/doi/10.1098/rspa.2020.0279. Publisher: Royal Society.

[28] J. v. Kan, G. Segal, and F. Vermolen. *Classical Numerical Methods in Scientific Computing*. TU Delft OPEN Textbooks, July 2023. ISBN 978-94-6366-732-6. doi: 10.59490/t.2023.007. URL https://textbooks.open.tudelft.nl/textbooks/catalog/book/67. Publication Title: TU Delft OPEN Textbooks.

[29] N. S. Kardashev. Transmission of Information by Extraterrestrial Civilizations. *Soviet Astronomy*, 8:217, Oct. 1964. ISSN 0038-5301. URL https://ui.adsabs.harvard.edu/abs/1964SvA.....8..217K. ADS Bibcode: 1964SvA.....8..217K.

[30] J. Leppänen, M. Pusa, T. Viitanen, V. Valtavirta, and T. Kaltiaisenaho. The Serpent Monte Carlo code: Status, development and applications in 2013. *Annals of Nuclear Energy*, 82:142–150, Aug. 2015. ISSN 0306-4549. doi: 10.1016/j.anucene.2014.08.024. URL https://www.sciencedirect.com/science/article/pii/S0306454914004095.

[31] T. Li, Y. Gao, D. Han, F. Yang, and B. Yu. A novel POD reduced-order model based on EDFM for steady-state and transient heat transfer in fractured geothermal reservoir. *International Journal of Heat and Mass Transfer*, 146:118783, Jan. 2020. ISSN 0017-9310. doi: 10.1016/j.ijheatmasstransfer.2019.118783. URL https://www.sciencedirect.com/science/article/pii/S0017931019325219.

[32] M. van den Berg, E. van den Broek, Z. Perko, D. Lathouwers, and J. Kloosterman. High-fidelity and Reduced-Order Modelling of a micro HTGR. - *The International Conference on Mathematics and Computational Methods Applied to Nuclear Science and Engineering*, August 13 – 17, 2023, Aug. 2023.

[33] S. Patel. The Allure of TRISO Nuclear Fuel Explained, Mar. 2021. URL https://www.powermag.com/the-allure-of-triso-nuclear-fuel-explained/.

[34] A. Pautz and A. Birkhofer. Coupling of Time-Dependent Neutron Transport Theory with the Thermal Hydraulics Code ATHLET and Application to the Research Reactor FRM-II. *Nuclear Science and Engineering*, 145(3):320–341, Nov. 2003. ISSN 0029-5639. doi: 10.13182/NSE03-A2386. URL http://www.scopus.com/inward/record.url?scp=0242544090&partnerID=8YFLogxK.

[35] B. Peherstorfer and K. Willcox. Data-driven operator inference for nonintrusive projection-based model reduction. *Computer Methods in Applied Mechanics and Engineering*, 306:196–215, July 2016. ISSN 0045-7825. doi: 10.1016/j.cma.2016.03.025. URL https://www.sciencedirect.com/science/article/pii/S0045782516301104.

[36] D. P. Prill and A. G. Class. Semi-automated proper orthogonal decomposition reduced order model non-linear analysis for future BWR stability. *Annals of Nuclear Energy*, 67:70–90, May 2014. ISSN 0306-4549. doi: 10.1016/j.anucene.2013.11.022. URL https://www.sciencedirect.com/science/article/pii/S0306454913006063.

[37] H. Ritchie and P. Rosado. Energy mix. *Our World in Data*, 2020. https://ourworldindata.org/energy-mix.

[38] unknown. CTU1 Delayed neutrons part 2 Experiment procedure for students, Apr. 2023. URL https://erasmus-plus.ec.europa.eu/projects/search/details/2020-1-CZ01-KA226-HE-094373.

[39] E. van den Broek. Uncertainty Quantification of the Neutronics Model of the U-Battery with Reduced Models. 2022. URL https://repository.tudelft.nl/islandora/object/uuid%3A4fcc0f45-1a20-420a-9373-0ca5acbb0d9c.

[40] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020. doi: 10.1038/s41592-019-0686-2.

[41] W. Wäreborn. *100% Renewable Energy Sourcing for Heavy Industry and the Role of Energy Storage : Renewable energy portfolio optimisation and the evolving role of the industrial energy consumer*. 2022. URL https://urn.kb.se/resolve?urn=urn:nbn:se:uu:diva-490682.

[42] J. B. Zulkeefal Dar and R. Codina. Reduced order modeling, 2024. URL https://deca.upc.edu/en/people/ramon.codina/publications-1/monographs-and-book-chapter/chapweb11.pdf.

Appendix A

Appendix

A.1 Full derivation of the finite volume method

A.1.1 Functional within domain

We integrate over the control volume between $x_{k-1/2}$ and $x_{k+1/2}$.

$$\int_{x_{k-1/2}}^{x_{k+1/2}} \frac{d}{dx}\left(\lambda \frac{dT}{dx}\right) dx = \int_{x_{k-1/2}}^{x_{k+1/2}} Q dx$$

Becomes:

$$-\lambda \frac{dT}{dx}\bigg|_{x_{k+1/2}} + \lambda \frac{dT}{dx}\bigg|_{x_{k-1/2}} = \int_{x_{k-1/2}}^{x_{k+1/2}} Q dx \tag{A.1}$$

Taking central divided differences on the left-hand side, and one point integration on the right hand side:

$$\lambda_{k-1/2}\frac{T_k - T_{k-1}}{\Delta x_k} - \lambda_{k+1/2}\frac{T_{k+1} - T_k}{\Delta x_{k+1}} = \frac{1}{2}(\Delta x_k + \Delta x_{k+1})Q_k + Error$$

Rearranging gives us:

$$\frac{-\lambda_{k-1/2}}{\Delta x_k}T_{k-1} + \left(\frac{\lambda_{k-1/2}}{\Delta x_k} + \frac{\lambda_{k+1/2}}{\Delta x_{k+1}}\right)T_k - \frac{\lambda_{k+1/2}}{\Delta x_{k+1}}T_{k+1} = \frac{1}{2}(\Delta x_k + \Delta x_{k+1})Q_k + Error$$

Allowing us to evaluate the functional T at x_k.

A.1.2 Functional at left side of domain with Neumann boundary condition

At the left boundary equation A.1 becomes:

$$-\lambda \frac{dT}{dx}\bigg|_{x_{1/2}} + \lambda \frac{dT}{dx}\bigg|_{x_0} = \int_{x_0}^{x_{1/2}} Q dx$$

We can substitute the Neumann boundary condition; $\frac{\partial}{\partial x}T(0,t)\Big|_{x=0} = 0$ giving us:

$$-\lambda \frac{dT}{dx}\bigg|_{x_{1/2}} = \int_{x_0}^{x_{1/2}} Q dx$$

Once more we apply central difference method and one point integration to produce:

$$\frac{\lambda_{1/2}}{\Delta x_1}T_0 - \frac{\lambda_{1/2}}{\Delta x_1}T_1 = \frac{1}{2}\Delta x_1 Q_0 + Error$$

A.1.3 Functional at right side of domain with mixed boundary condition

At the right boundary equation A.1 becomes:

$$-\lambda \frac{dT}{dx}\bigg|_{x_N} + \lambda \frac{dT}{dx}\bigg|_{x_{N-1/2}} = \int_{x_{N-1/2}}^{x_N} Q dx$$

We can substitute the Robin boundary condition $a\frac{\partial}{\partial x}T(L,t)\Big|_{x=L} = bT(L,t) + c$ into this equation, giving:

$$\lambda \frac{dT}{dx}\bigg|_{x_{N-1/2}} + bT_N = \int_{x_{N-1/2}}^{x_N} Q dx + c$$

Applying central differences and one point integration:

$$-\frac{\lambda_{N-1/2}}{\Delta x_N} T_{N-1} + \left(\frac{\lambda_{N-1/2}}{\Delta x_N} + b \right) T_N = \frac{1}{2} \Delta x_N Q_N + c + Error$$

A.2 Parameters

A.2.1 Temperature

Table A.1: Table showing the necessary parameters for the temperature calculations. Numbers from [32] and given by M. van den Berg from `CR0213/DLOFC_20/ThermalHydraulics/TH_params_SS`. We note that there is no heat transfer coefficient between the fuel and graphite. Furthermore with the heat transfer coefficient of graphite, we understand the heat transfer coefficient between the coolant and the graphite.

	λ[Wcm^{-1}K^{-1}]	ρ[gcm^{-3}]	c_p[Jg^{-1}K^{-1}]	α[Wcm^{-2}K^{-1}]
Fuel	1.33	2.8	0.25	-
Graphite	0.43	1.89	0.72	0.03

A.2.2 Neutronics

Fast group

Table A.2: Table showing the necessary parameters for the neutronics calculations. Numbers are from `SERPENT` [30]. T_1 is 294K and T_2 is 2500K. 'C' stands for the homogenized core, and 'G' stands for the graphite reflector. Σ_T denotes the total cross-section (absorption + scattering), Σ_f denotes the fission cross-section, Σ_s denotes the scattering cross-section. ν is the neutron yield, the amount of neutrons created per fission event, w is the power generated per fission event, and v is the velocity of the neutrons. For the scattering cross-section the first entry is the inter-group scattering, and the second entry is the fast to thermal group scattering.

	Σ_T [10^{-1}cm^{-2}]	Σ_f [10^{-4}cm^{-2}]	Σ_s [10^{-1}cm^{-2}]		ν	w [10^{-11}J]	v [10^5cms^{-1}]
C (T_1)	3.57043	3.67373	3.52293	0.0349573	2.44655	3.2425339	102.215
C (T_2)	3.57603	3.52896	3.52774	0.0329774	2.44710	3.24263	102.215
G (T_1)	4.41559	0	4.34727	0.0681558	0	0	102.215
G (T_2)	4.41227	0	4.34591	0.0662095	0	0	102.215

The absorption cross-section is calculated as $\Sigma_a = \Sigma_T - \Sigma_s$ the total cross-section minus the scattering cross-section. For the fast group this scattering cross-section is the inter-group scattering added to the fast-thermal scattering.

Thermal group

Table A.3: Table showing the necessary parameters for the neutronics calculations. Numbers are from SERPENT [30]. T_1 is 294K and T_2 is 2500K. 'C' stands for the homogenized core, and 'G' stands for the graphite reflector. Σ_T denotes the total cross-section (absorption + scattering), Σ_f denotes the fission cross-section, Σ_s denotes the scattering cross-section. ν is the neutron yield, the amount of neutrons created per fission event, w is the power generated per fission event, and v is the velocity of the neutrons. For the scattering cross-section the first entry is the inter-group scattering, and the second entry is the fast to thermal group scattering.

	Σ_T $[10^{-1}\text{cm}^{-2}]$	Σ_f $[10^{-4}\text{cm}^{-2}]$	Σ_s $[10^{-1}\text{cm}^{-2}]$	ν	w $[10^{-11}\text{J}]$	v $[10^{5}\text{cms}^{-1}]$
C (T_1)	4.48061	32.5539	4.4272	2.43670	3.2407234	4.48234
C (T_2)	4.50236	30.4173	4.45105	2.43669	3.2407234	4.48234
G (T_1)	4.8757	0	4.34727	4.86611	0	4.48234
G (T_2)	4.87555	0	4.34591	4.86588	0	4.48234

The absorption cross-section is calculated as $\Sigma_a = \Sigma_T - \Sigma_s$ the total cross-section minus the scattering cross-section. For the fast group this scattering cross-section is the inter-group scattering added to the fast-thermal scattering.

A.2.3 Delayed

Table A.4: Table showing the necessary parameters for the delayed neutrons arising from the precursor groups. $t_{1/2}$ denotes the half-life of the precursor, λ denotes the decay constant of the precursor, and β is the fraction of total neutrons that are delayed neutrons, coming from precursors.

$t_{1/2}$ [s]	52.38
λ $[\text{s}^{-1}]$	1.33
β	0.0075

A.3 1 group infinite homogeneous reactor temporal convergence

As an addition to the two group temporal convergence we also investigated the 1 group neutron diffusion equation. We induced a transient and compared the analytical results to our numerical results.

For the prompt jump we take the 1 group neutron diffusion equation from 2.2, and assume the reactor is infinite and homogeneous. We implement this in our representative model by using the expression for the two group equation (3.13), in which we turn off the thermal group. We then apply a Neumann boundary at both the left and right sides, to create an infinite homogeneous reactor. From the resulting expression,

$$\begin{aligned} \frac{1}{v_1}\frac{\partial \boldsymbol{\phi_1}(t)}{\partial t} - D_1\frac{\partial^2 \boldsymbol{\phi_1}(t)}{\partial x^2} + \Sigma_{a_1}\boldsymbol{\phi_1}(t) &= (1-\beta)[\nu_1\Sigma_{f_1}\boldsymbol{\phi_1}(t)] + \lambda \boldsymbol{C} \\ \frac{d\boldsymbol{C}(t)}{dt} &= -\lambda \boldsymbol{C} + \beta[\nu_1\Sigma_{f_1}\boldsymbol{\phi_1}(t)], \end{aligned} \tag{A.2}$$

we can derive an analytical expression for the prompt jump.

When a reactor goes through a transient the neutron fluxes change rapidly at first, this rapid change is due to the change in the prompt neutrons, which quickly react to a change in reactivity. At longer time-scales the change in flux is more related to the delayed neutrons. This rapid change is called the prompt jump and can be analytically determined. We look at ratio of the power deviating from the steady-state $\frac{P}{P_0}$. To this end, we introduce the reactor kinetics equations from[20],

$$\begin{aligned} \frac{P(t)}{P_0} &= n_1 e^{s_1 t} + n_2 e^{s_2 t}, \\ s_{1,2} &= \frac{1}{2\Lambda}\left[-(\beta - \rho_0 + \lambda\Lambda) \pm \sqrt{(\beta-\rho_0+\lambda\Lambda)^2 + 3\Lambda\lambda\rho_0}\right], \\ n_1 &= \frac{\frac{P_0(\rho_0-\beta)}{\Lambda} + \lambda C_0 - P_0 s_2}{s_1 - s_2}, \\ n_2 &= P_0 - n_1. \end{aligned} \tag{A.3}$$

where ρ_0 is the changed reactivity, $\rho_0 = \frac{k_{eff}-1}{k_{eff}}$, with k_{eff} the new multiplication factor if we change the reactivity. λ is the decay constant for the precursor group, β is the delayed neutron fraction, Λ is the mean generation time, and C_0 is the initial precursor density, from [20]. With this expression we can analytically determine the power ratio after a reactivity increase or decrease in a 1 group infinite homogeneous reactor.

We start with a steady-state reactor, next we change the fission cross-section from 3.67×10^{-4} to and 3.68×10^{-4} the absorption cross section from 8.988×10^{-4} to 8.998×10^{-4}, lastly we change the neutron yield from 2.44 to 2.45. From the resulting transient we calculate the analytically expected power ratio, and determine the ratio of power to steady state power at each time step numerically, using equation 2.7. The resulting power ratio's in time are shown in figure A.1.

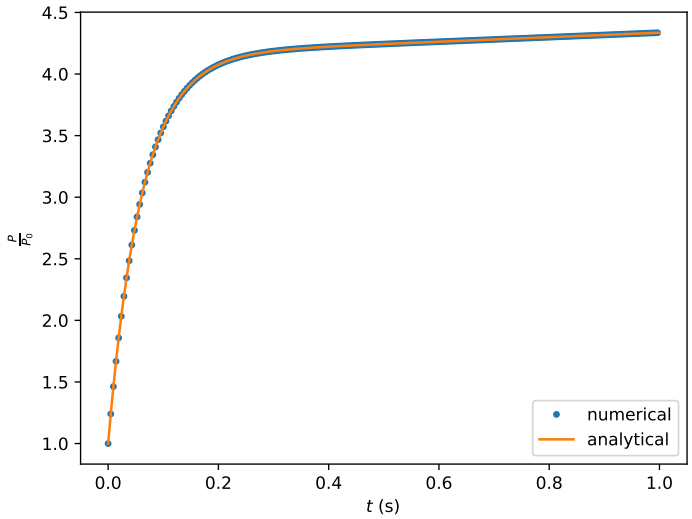


Figure A.1: Numerical and analytical power ratios. Numerically determined power ratio is shown with blue circles. The analytical expression has been plotted throug it with a full orange line. In this experiment $\Delta t = 10^{-7}s$.

In figure A.2 the RMS error (eqation 3.2) between the numerical and analytical power ratio is plotted against Δt.

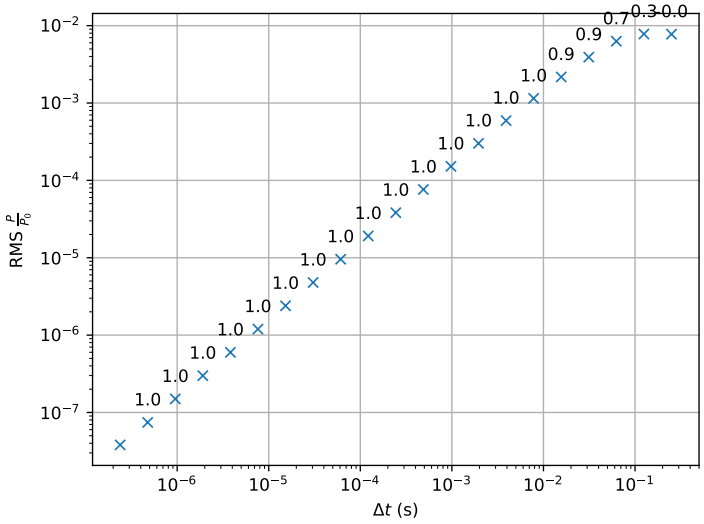


Figure A.2: RMS error between the numerical and analytical power ratios. Starting from the top right every blue cross shows the RMS error at that Δt. *Above the value the slope to the next value is shown. Here* 1 *means that the slope goes with* $\sim \Delta t$.

We see that the temporal convergence goes with Δt, as we would expect from first order BDF.

A.4 Spatial and temporal convergence of the thermal flux of the representative model

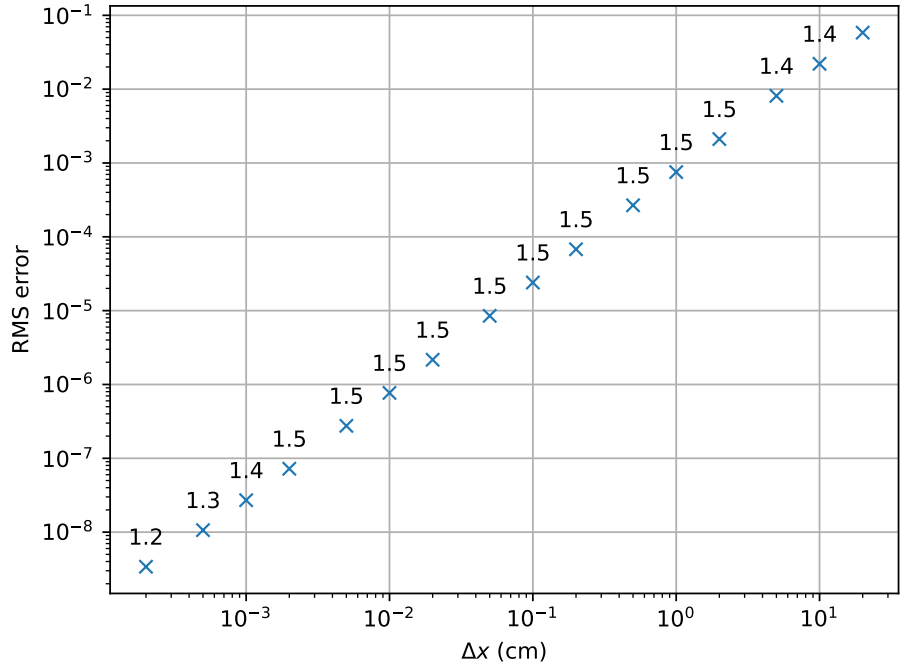


Figure A.3: Convergence of the thermal flux of numerical solutions with various Δx to a pseudo analytical solution. Above the value the slope to the next value is shown. Here 2 means that the slope goes with $\sim \Delta x^2$.

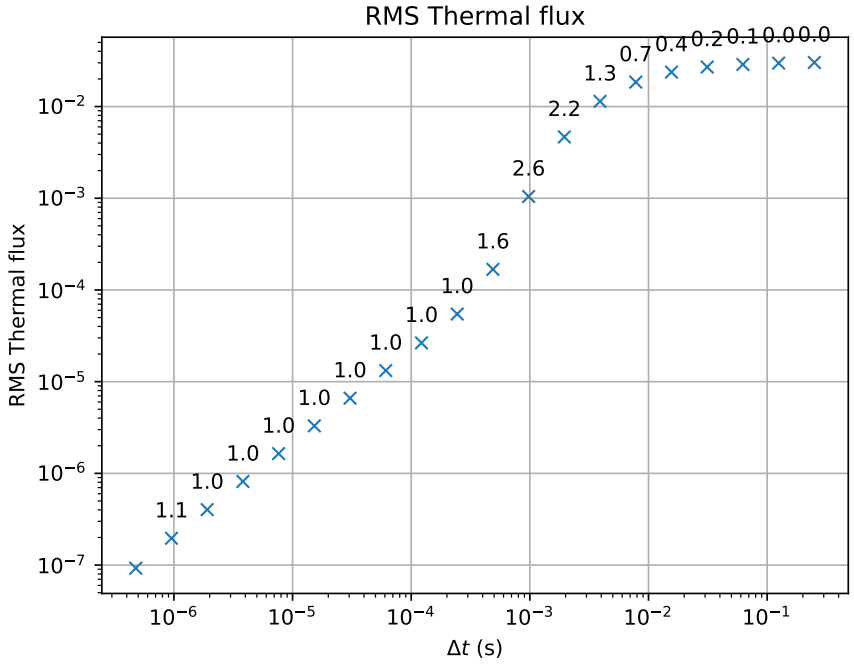


Figure A.4: RMS error between the numerical and pseudo analytical thermal fluxes Starting from the top right every blue cross shows the RMS error at that Δt. Above the value the slope to the next value is shown. Here 1 means that the slope goes with $\sim \Delta t$.

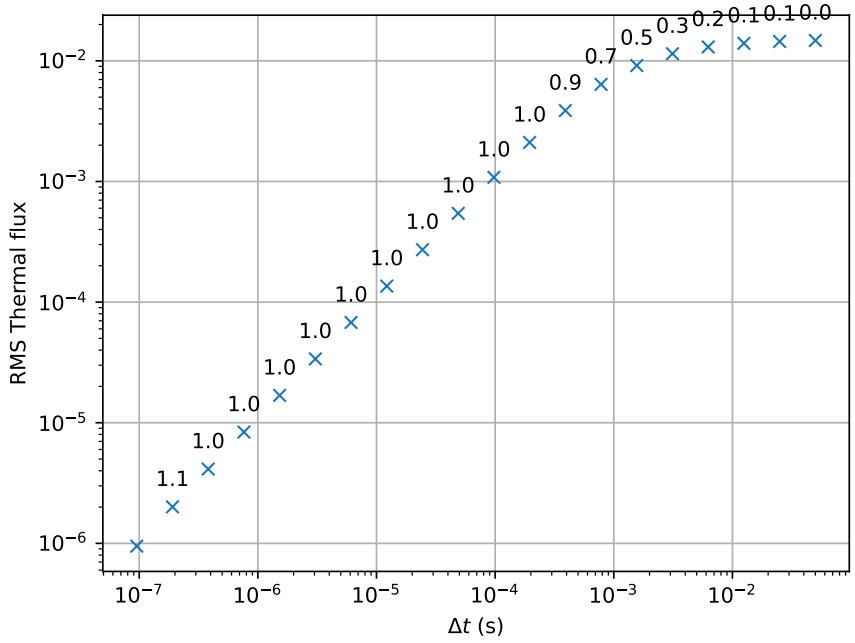


Figure A.5: RMS error between the numerical and pseudo analytical thermal fluxes Starting from the top right every blue cross shows the RMS error at that Δt. Above the value the slope to the next value is shown. Here 1 means that the slope goes with $\sim \Delta t$.

We see that the RMS error convergence of the thermal flux is the same as the fast flux, from Chapter 5.

A.5 ROM_T test cases

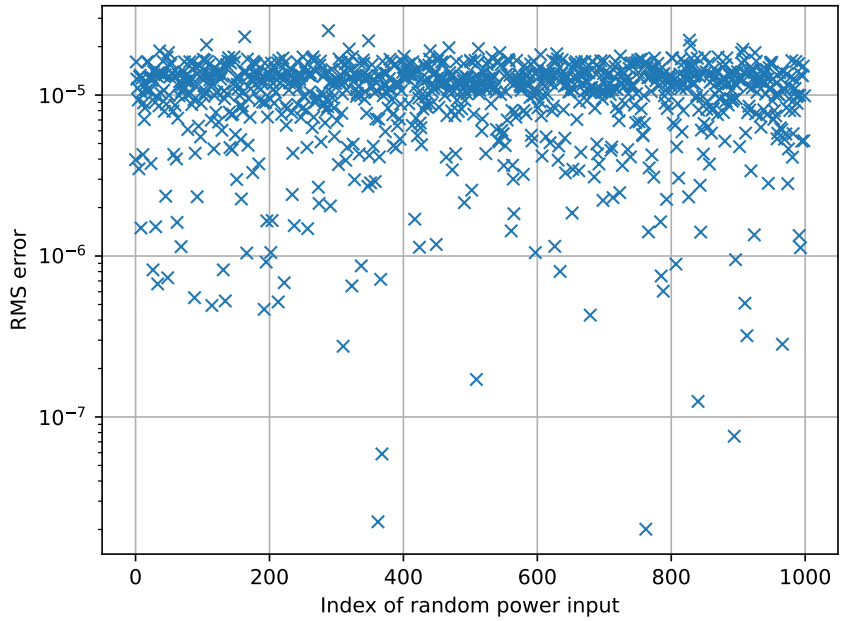


Figure A.6: RMS errors for 1000 random power inputs for the ROM_T. In these cases we multiplied the power of the first fuel block with a random value between -100 and 100, and the power of the second fuel block by another random value also between -100 and 100. We used 12 modi and 1500 snapshots to create the ROM_T.

We see that the RMS error of various random power outputs does not exceed 3×10^{-5}

A.6 Average RMS error of the ROM_T using 3 fuel blocks

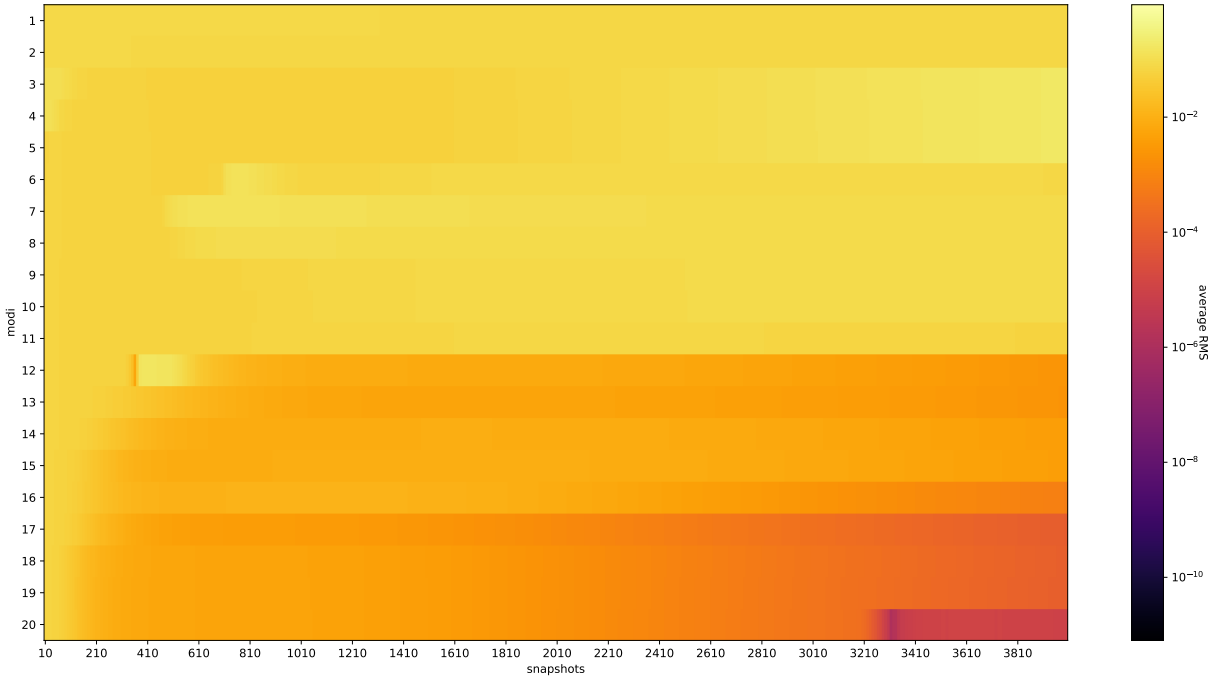
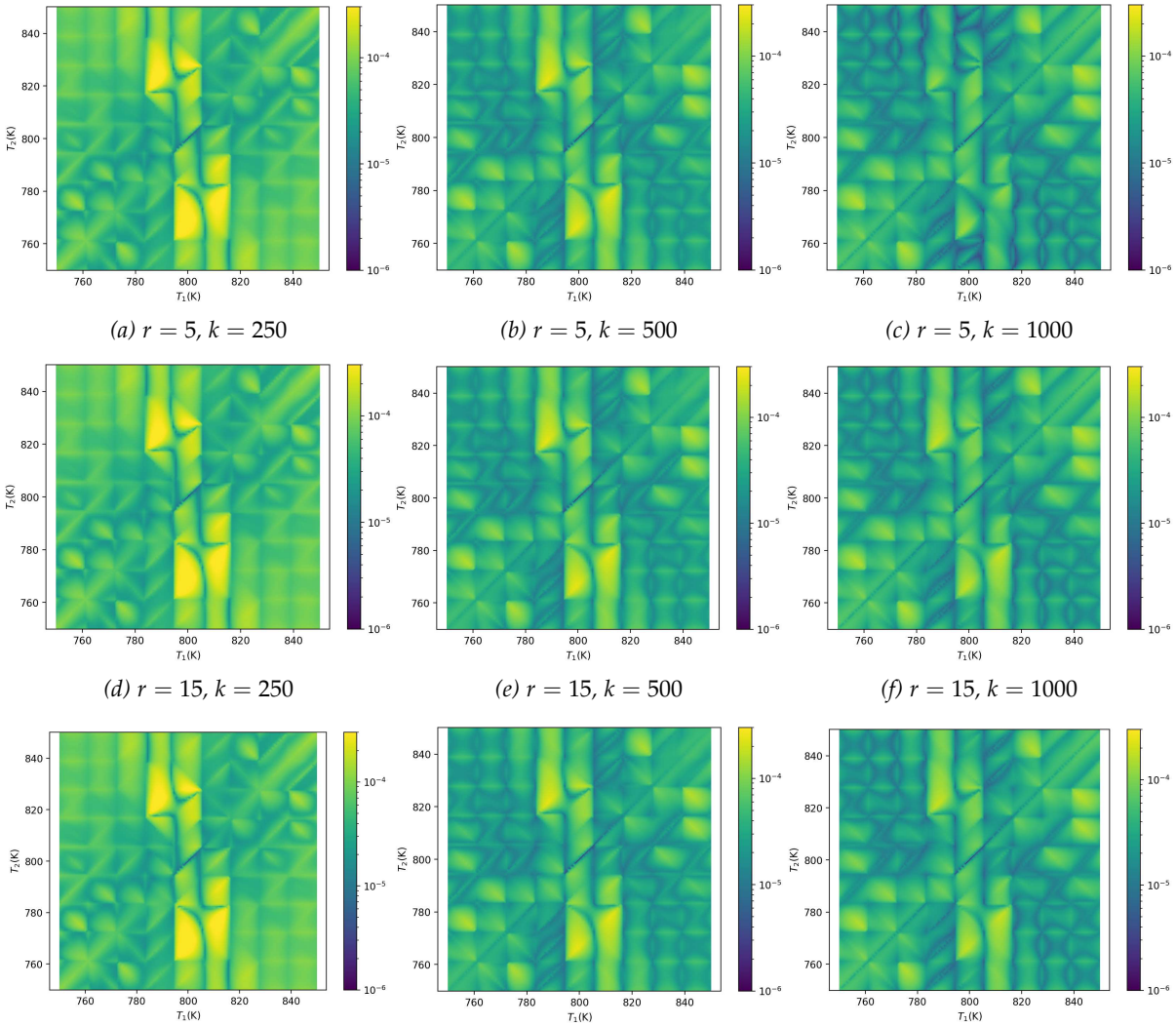


Figure A.7: Average RMS error in time for the ROM_T of ROMs with varying amount of modi (r) and snapshots (k). The snapshots are consecutive to each other. In this test the reactor has 3 fuel blocks

A.7 Profiles of RMS errors in temperature parameter space for different modi and snapshots used in ROM_N



(a) $r = 5$, $k = 250$ (b) $r = 5$, $k = 500$ (c) $r = 5$, $k = 1000$

(d) $r = 15$, $k = 250$ (e) $r = 15$, $k = 500$ (f) $r = 15$, $k = 1000$

(g) $r = 25$, $k = 250$ (h) $r = 25$, $k = 500$ (i) $r = 25$, $k = 1000$

Figure A.8: 2D RMS error profiles for ROMs with varying amounts of modi and snapshots.

A.8 Training analysis of mROM for training set with 3 training runs

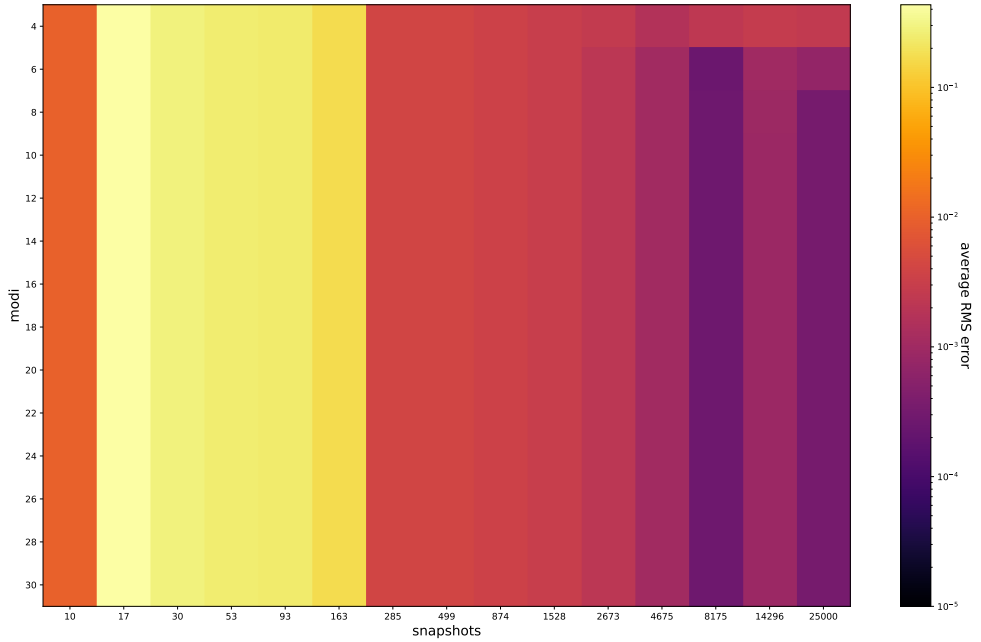


Figure A.9: RMS error of the fast flux between the mROM and our FOM. Here we use a training set with 3 different heat transfer coefficient values equidistant between 0 and the nominal value of 0.03 $\mathrm{Wcm^{-2}K^{-2}}$ *to simulate a DLOFC. Every run is 25 s long with* $\Delta t = 10^{-3}$*s. The resulting max temperature difference is for a test with a heat transfer coefficient of 0.00375* $\mathrm{Wcm^{-2}K^{-2}}$*, running for 100 s with* $\Delta t = 10^{-3}$ *s.*

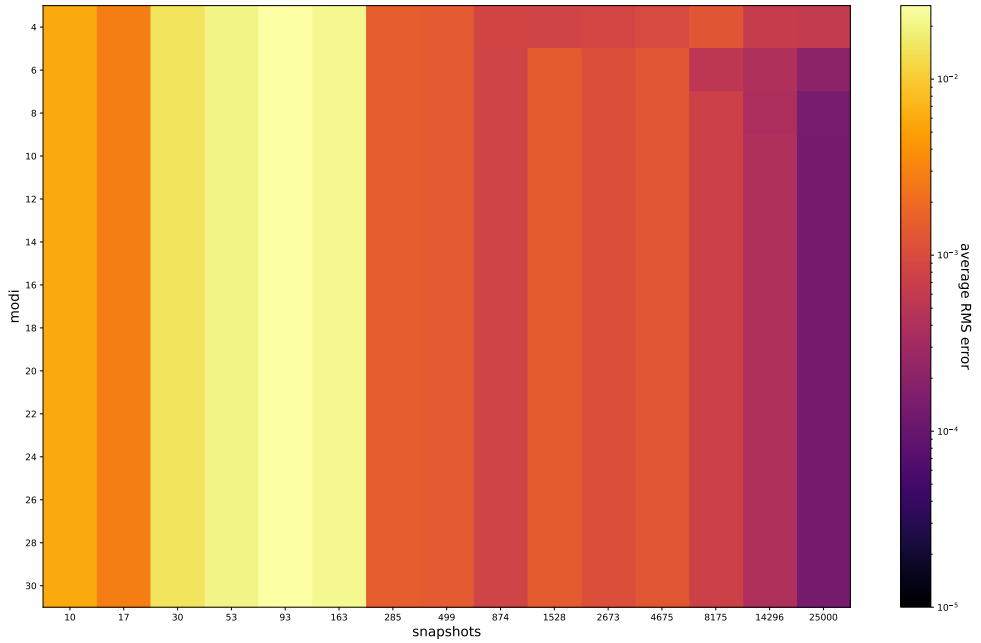


Figure A.10: RMS error of the temperature between the mROM and our FOM. Here we use a training set with 3 different heat transfer coefficient values equidistant between 0 and the nominal value of 0.03 $\mathrm{Wcm^{-2}K^{-2}}$ *to simulate a DLOFC. Every run is 25 s long with* $\Delta t = 10^{-3}$*s. The resulting max temperature difference is for a test with a heat transfer coefficient of 0.00375* $\mathrm{Wcm^{-2}K^{-2}}$*, running for 100 s with* $\Delta t = 10^{-3}$ *s.*

A.9 Training analysis of mROM for training set with 5 training runs

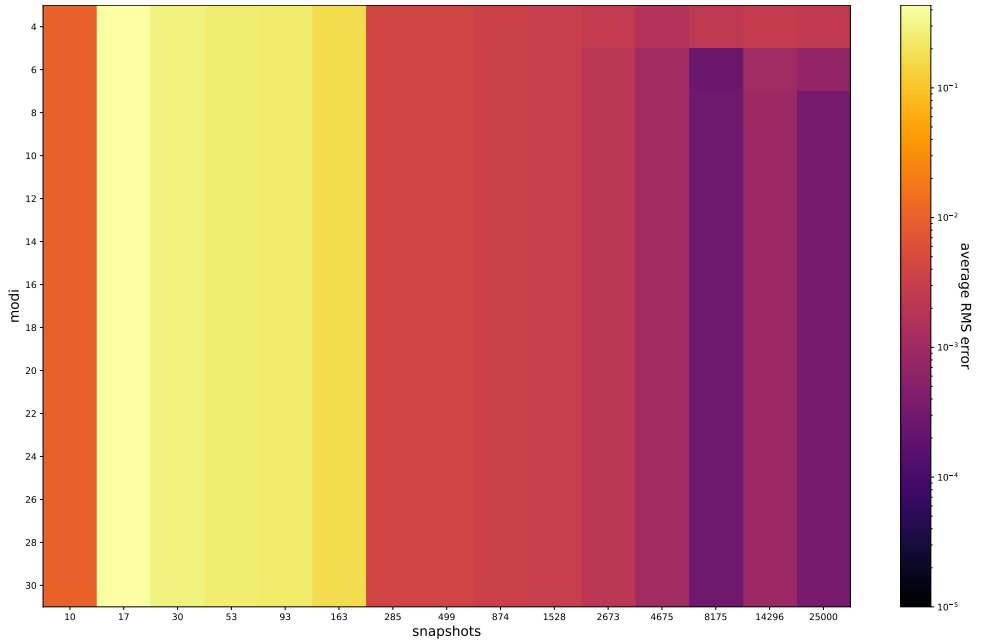


Figure A.11: The maximum absolute temperature difference in time between our mROM and our FOM. Here we use a training set with 5 different heat transfer coefficient values equidistant between 0 and the nominal value of 0.03 $\mathrm{Wcm^{-2}K^{-2}}$ *to simulate a DLOFC. Every run is 25 s long with* $\Delta t = 10^{-3}s$*. The resulting max temperature difference is for a test with a heat transfer coefficient of 0.00375* $\mathrm{Wcm^{-2}K^{-2}}$*, running for 100 s with* $\Delta t = 10^{-3}$ *s.*

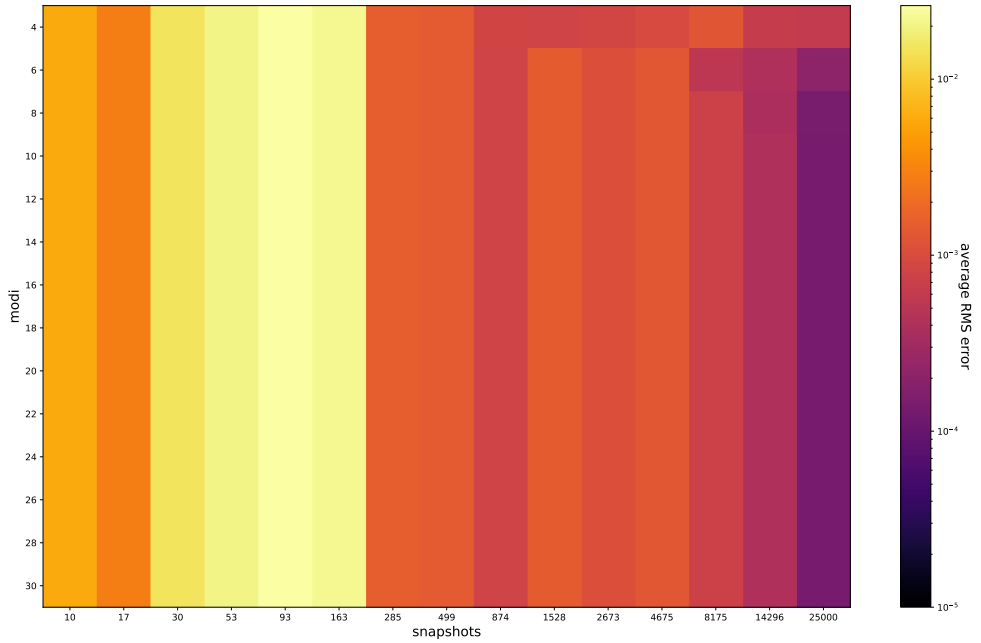


Figure A.12: RMS error of the fast flux between the mROM and our FOM. Here we use a training set with 5 different heat transfer coefficient values equidistant between 0 and the nominal value of 0.03 $\mathrm{Wcm^{-2}K^{-2}}$ *to simulate a DLOFC. Every run is 25 s long with* $\Delta t = 10^{-3}s$*. The resulting max temperature difference is for a test with a heat transfer coefficient of 0.00375* $\mathrm{Wcm^{-2}K^{-2}}$*, running for 100 s with* $\Delta t = 10^{-3}$ *s.*

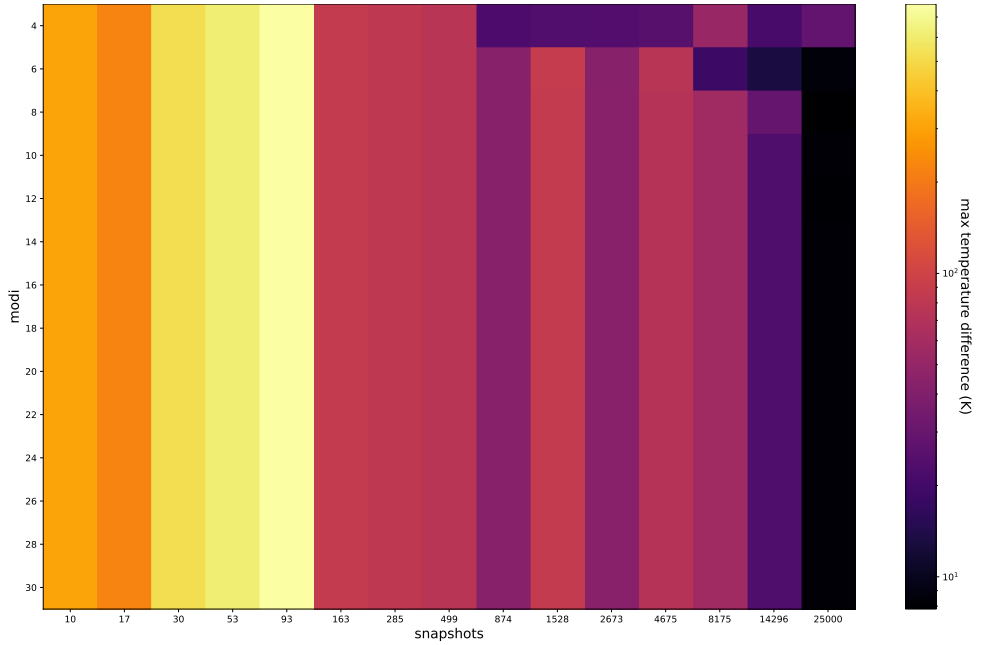


Figure A.13: RMS error of the temperature between the mROM and our FOM. Here we use a training set with 5 *different heat transfer coefficient values equidistant between 0 and the nominal value of 0.03* $\mathrm{Wcm^{-2}K^{-2}}$ *to simulate a DLOFC. Every run is 25 s long with* $\Delta t = 10^{-3}$*s. The resulting max temperature difference is for a test with a heat transfer coefficient of 0.00375* $\mathrm{Wcm^{-2}K^{-2}}$*, running for 100 s with* $\Delta t = 10^{-3}$ *s.*