Imperial College London
Department of Earth, Science and Engineering

A Hierarchical Reduced-Order Model applied to Nuclear Reactors

Toby R. F. Phillips

Supervised by
Prof. Christopher C. Pain
Dr Claire E. Heaney

Mentored by
Prof. Paul N. Smith

Submitted in partial fulfilment of the requirements for the degree of
Doctor of Philosophy, December 2022

Statement of Originality

I herewith certify that all material in this thesis is a result of the research undertaken by me and any which is not my own work has been properly acknowledged.

Toby Phillips

Copyright Notice

Abstract

Modelling the neutron transport of a nuclear reactor is a very computationally demanding task that requires a large number of degrees of freedom to accurately capture all of the physics. For a complete reactor picture, other physics must be incorporated, through coupling, further exacerbating the computational demand. Computational modelling has many benefits: optimisation, real-time analysis, and safety analysis are some of the more important ones. However, nuclear modelling has yet to capitalise on these, and existing approaches are too computationally demanding.

Machine Learning has seen incredible growth over the last decade, but it has yet to be utilised within the nuclear modelling community to the same extent. The frameworks available represent incredibly efficient and optimised code, having been written to run on GPUs and AI computers. Presented here is a physics-driven neural network that solves neutron transport, first for the diffusion approximation and then extended to the whole transport problem.

One method that can potentially reduce the computational complexity is Reduced-Order Modelling (ROM), which is a way to define a low-dimensional space in which a high-dimensional system can be approximated. These established methods can be used with machine learning methods, potentially reducing computational costs further than either method individually. A method to utilise autoencoders with a projection-based framework is also presented here.

The structure of a reactor can be broken down, forming a hierarchy which starts with the reactor core, which is populated by fuel assemblies, which are then populated by fuel rods. This hierarchy means that materials are repeated within a solution, and many existing methods do not capitalise on this and instead resolve the entire global domain. This research presents two ways to utilise this structure with ROM. The first involves combining it with domain decomposition, producing ROMs for the sub-structures. The second presents a hierarchical interpolating method, reducing the number of sub-domains within the solution that need to be resolved.

Acknowledgements

I would first like to thank my supervisors, Christopher Pain and Claire Heaney. Thank you both for all the support, guidance and encouragement you gave throughout the last four years, even during the most turbulent times. I am grateful for the time spent discussing research and non-research topics, and this endeavour would not have been possible without either of you. Thank you for making pursuing research the incredibly enjoyable experience it has been.

I would also like to thank my mentor, Paul Smith, for the advice, guidance and interesting discussions across several topics.

Thanks to the entire Reactor Transport group for providing incredible expertise and support. The weekly meetings always provided new insights and inspiration. Additionally, thanks to the whole Applied Modelling and Computation Group for creating an excellent research environment.

Thank you to the Centre for Doctoral Training in Nuclear Energy for providing the funding and invaluable experiences over the years.

I want to thank Ioannis and my fellow PhD students, especially from the CDT, for the discussions and experiences.

A thank you to all of my friends, whose support was incredible. Jason and Elliot, both of whom had to deal with living with me during stressful times, for more time than intended. James and Stephen, thank you for always showing an interest in the specifics of my research.

Finally, I would like to thank Mafalda. You have an infinite amount of patience, and I am incredibly grateful for your support and kindness during these times.

Contents

Statement of Originality **1**

Copyright Notice **2**

Abstract **3**

Acknowledgements **4**

1 Introduction **12**

1.1 Motivation and Objectives 12

1.2 Publications 14

1.3 Thesis outline 15

2 Literature Review **17**

2.1 Neutron Transport 17

2.1.1 Governing Equations 17

2.1.2 Discretisation of governing equations 18

2.2 Diffusion Approximation 20

2.2.1 Discretisation 21

2.3 Power Method 22

2.4 High-Fidelity Models 24

2.5 Multigrid methods 24

2.6 Reduced-Order Modelling 25

2.6.1 Non-linear systems 26

2.6.2 Non-intrusive ROM 27

2.6.3 ROM applied to neutron transport 28

2.7 Machine Learning 29

2.7.1 Autoencoders 31

2.7.2 Autoencoders used for ROM 32

2.8 Domain Decomposition 34

2.9 Summary 35

3 Diffusion Neural Network Solver **37**

3.1 Introduction 37

3.2 Methodology 39

3.2.1 Discretisation 40

3.2.2 Neural network as a Jacobi Solver 42

3.2.3 Multigrid 47

3.2.4 Multi-group network 50

3.3 Results 52

3.3.1 Fuel Assembly 53

3.3.2 Fuel Assembly - ConvFEM Filters 55
3.3.3 Fuel Assembly - Time comparisons 56
3.3.4 Reactor Core 58
3.3.5 Reactor Core - Time comparisons 60
3.4 Summary 62

4 Neutron Transport Neural Network Solver 64

4.1 Introduction 64
4.2 Methodology 66
4.2.1 Boltzmann Transport Equation 67
4.2.2 Discrete Ordinates 67
4.2.3 Discretisation in Cartesian space 70
4.2.4 Neural network filters 73
4.2.5 Space-Angle Multigrid 75
4.2.6 Optimised Anisotropic Non-Linear Petrov-Galerkin Dissipation 79
4.2.7 Neural Network Filters for the Convolutional Finite Element Method (ConvFEM) 82
4.3 Results 85
4.3.1 Straight Duct problem 85
4.3.2 Fuel Assembly 90
4.4 Summary 93

5 Reduced-Order Modelling 95

5.1 Introduction . . . 95

5.2 Proper Orthogonal Decomposition . . . 95

5.3 POD Basis Functions . . . 97

5.4 Constructing a Reduced-Order Model for Criticality with POD . . . 98

5.5 A Standard Autoencoder . . . 100

5.5.1 Variational Autoencoder . . . 102

5.5.2 SVD-Autoencoder . . . 102

6 Autoencoder-based ROM **104**

6.1 Introduction . . . 104

6.2 Methodology . . . 107

6.2.1 Error measures . . . 110

6.2.2 Homogenising the Material Parameters . . . 112

6.3 1D Slab Reactor . . . 113

6.3.1 POD-based Reduced-Order Model . . . 114

6.3.2 Autoencoder-based Reduced-Order Model . . . 115

6.3.3 Comparison of POD and AE as compression methods . . . 116

6.3.4 Comparison of POD-based and AE-based ROMs . . . 118

6.3.5 Variational autoencoder reduced-order model . . . 121

6.3.6 Reduction to two variables . . . 125

6.4 2D reactor core eigenvalue problem . . . 127

6.4.1 POD-based ROM . . . 129

6.4.2 AE-based ROM 129

6.4.3 Combined Singular Value Decomposition and Autoencoder approach . . 130

6.4.4 Compression using POD, an autoencoder and the SVD-AE 131

6.4.5 Comparing results from the ROMs based on POD, AE and SVD-AE . . 134

6.5 Summary 140

7 Domain Decomposition ROM **142**

7.1 Introduction 142

7.2 Methodology 143

7.2.1 Error Measures 147

7.3 1D Reactor 147

7.3.1 Test Problem and Construction of Snapshot Matrices 147

7.3.2 Construction of Basis Functions 150

7.3.3 Limiting the Number of Snapshots 152

7.3.4 Smart Selection of the Number of Basis Functions 153

7.3.5 POD-Based Results 154

7.3.6 Results for Limiting the Number of Snapshots and Smart Selection of the Number of Basis Functions 157

7.4 2D Reactor 163

7.4.1 Test Problem and construction of snapshot matrices 163

7.4.2 Additional data set and construction of alternative snapshot matrices . . 166

7.4.3 Construction of ROMs 168

7.4.4 Scaling factor for Autoencoder-based ROMs . . . 169
7.4.5 Compression . . . 170
7.4.6 Results . . . 171
7.5 Summary . . . 179

8 Hiearchical ROM **183**

8.1 Introduction . . . 183
8.2 Methodology . . . 185
8.2.1 Determining matrices for sub-domains . . . 185
8.2.2 Determining the Interpolation Matrix . . . 186
8.2.3 Hierarchical Reduced-Order Model for Criticality with POD . . . 186
8.2.4 Adjusted Hierarchical ROM . . . 187
8.2.5 Fuel Assembly Problem and Construction of Matrices . . . 187
8.2.6 Reactor Core problem . . . 191
8.3 Results . . . 192
8.3.1 Error Measures . . . 192
8.3.2 Fuel Assembly Results . . . 193
8.3.3 Reactor core Results . . . 194
8.4 Summary . . . 198

9 Conclusion **200**

9.1 Summary of Thesis Achievements . . . 200
9.2 Future Work . . . 204

9.2.1 Neural Network Solvers 204

9.2.2 Reduced-Order Modelling 205

9.2.3 The Transparent Neural Nuclear Reactor 207

Bibliography **208**

A Appendix **224**

A.1 The Convolution Finite Element Method (ConvFEM) Filters 224

A.1.1 Linear Filters 224

A.1.2 Quadratic Filters 225

A.1.3 Cubic Filters 227

Chapter 1

Introduction

1.1 Motivation and Objectives

Modelling neutron transport within a reactor core presents a significant challenge, even with modern computing technology. The resolution of an entire reactor core, such as the ones currently in use, is computationally expensive and remains beyond the capabilities of many practitioners. The solution to neutron transport problems lies in the seven-dimensional phase space, encompassing space, angle, energy, and time, thus further increasing the complexity of the problem. All discretisations must be detailed enough to accurately capture the physics, which may include absorption, scattering, and fission, as represented by material cross-sections [1]. Fuel Rods have a highly challenging aspect ratio, resulting in the discretisation requiring many degrees of freedom to model accurately. For instance, the reactor core of Sizewell B, the UK's only active PWR [2], contains 193 fuel assemblies. Each contains 264 fuel rods, measuring 3.9m in height and 9.5mm in diameter. To simulate this system, 50,952 fuel rods must be modelled simultaneously. One discretisation of the Westinghouse PWR-900 core with 44 energy groups required over 1.7 trillion degrees of freedom to be modelled accurately [3, 4]. This combination of complex geometry and intricate physics is what makes these problems so difficult to solve.

A high-fidelity model capable of capturing the relevant physics and geometry, and solving the requisite number of dimensions, can require days or even weeks to solve. The time required for this renders simulations unsuitable for optimisation, safety analysis, and real-time analysis, thereby limiting the potential applications of computational modelling since its development.

Nonetheless, utilising the world's most powerful supercomputers, it is possible to solve problems on 10^5 processor cores in 100 minutes [3]. However, such machines are typically accessible to only a few individuals and require a high level of technical expertise. Therefore, most practitioners cannot utilise supercomputers for computational modelling.

Machine learning is a subset of computer science that has experienced substantial growth over the last decade. Algorithms and models that can learn from data without requiring explicit instructions have become more accessible due to the rise of computing power. Fluid dynamics has exhibited the potential of these methods [5], with the most noteworthy advancement being the use of artificial neural networks, or deep learning, in a broad scope of applications. Despite the successes in fluid dynamics, there have been minimal demonstrations of deep learning in neutron transport. The frameworks available for machine learning are typically well-optimised, having been written for use with multiple Graphic Processor Units (GPUs). Neural networks, therefore, can utilise this hardware in a way that traditional code cannot. Typically a neural network will be data-driven, being trained to approximate the entire system. However, the neural network architecture can use pre-determined weights, without training, to produce a physic-driven network. These physics-driven networks allow computationally demanding physics problems, such as neutron transport, to be efficiently run on GPUs.

Reduced order modelling (ROM), also known as Model (order) reduction, is a numerical technique that has influenced how computational science is performed [6]. It is the process of reducing the calculations required by reducing the dimensions of a system to a more manageable level while retaining the detail of a high-fidelity model (HFM). For instance, a reactor system with millions of degrees of freedom may take days to simulate; however, ROM can be applied to the computational modelling to decrease the degrees of freedom to hundreds, thus allowing simulations to take a fraction of a second. If an entire reactor could be modelled with all the detail of a HFM, at a fraction of the computational cost, then nuclear modelling can utilise the previously out-of-reach computational modelling benefits.

Domain decomposition breaks a problem into sub-domains, iterating across the whole domain until convergence, and the internal structures of nuclear reactors lend themselves to domain decomposition. A reactor core typically consists of multiple fuel assemblies populated by fuel rods and control rods. Considering the reactor core as the global domain, the fuel assemblies can be the sub-domains. Each fuel assembly domain can then have the fuel rods be the sub-

domains. Breaking down the structure of a reactor in this way creates a hierarchy of domains to be resolved. Domain decomposition has seen significant use for problems that can be broken down this way, mainly due to the advances in modern computing, which allow for the parallel solving of sub-domains [7]. The reactor structure being so suitable for domain decomposition means that nuclear modelling should be able to capitalise on these methods, increasing the benefits associated with computational modelling for the field.

The first objective of this thesis is to produce a physics-driven neural network for the forward neutron transport model, implementing traditional numerical schemes within machine learning frameworks. The second objective of this thesis is to develop methods by which reduced-order modelling can take advantage of the structure of the reactor, forming a hierarchy of sub-domains. The third objective is investigating how machine learning methods may be utilised within reduced-order modelling. These objectives seek to allow the nuclear industry to enjoy the benefits of computational modelling that has since been inaccessible.

1.2 Publications

Parts of this thesis have been published or are being prepared for submission, unpublished manuscript drafts are included for reference.

Chapter 3 contains a multigrid neural network solver applied to the diffusion approximation that is published in *International Journal for Numerical Methods in Engineering*:
T. R. F. Phillips, C. E. Heaney, B. Chen, A.G. Buchan, and C. C. Pain. Solving the Discretised Neutron Diffusion Equations using Neural Networks. [8]

Chapter 4 contains a space-angle multigrid neural network solver applied to the Boltzmann transport equation that is being prepared to be submitted:
T. R. F. Phillips, C. E. Heaney, B. Chen, A.G. Buchan, and C. C. Pain. Solving the discretised Boltzmann transport equations using neural networks: Applications in neutron transport. [9]

Chapter 6 contains a projection-based autoencoder ROM method published in *International Journal for Numerical Methods in Engineering*:
T. R. F. Phillips, C. E. Heaney, P. N. Smith, and C. C. Pain. An autoencoder-based reduced-order model for eigenvalue problems with application to neutron diffusion. [10]

Chapter 7 contains a domain-decomposition POD approach published in *Energies*:
T. R. F. Phillips, C. E. Heaney, B. S. Tollit, P. N. Smith, and C. C. Pain. Reduced-order modelling with domain decomposition applied to multi-group neutron transport. [11]

Additionally, other articles were written during this research that have been or are in preparation to be published. The first was published in *Applied Sciences*:
T. R. F. Phillips, C. E. Heaney, E. Benmoufok, Q. Li, L. Hua, A. E. Porter, K.Fan Chung, C. C. Pain. Multi-Output Regression with Generative Adversarial Networks (MOR-GANs).[12]

The second is being prepared for submission:
T. R. F. Phillips, C. C. Pain, P. Kumar, C. E. Heaney, Y. Zhou, W. Xua, L. Hua, V. L. S. Silva, C. Dilliway, R. Arcucci, C. Quilodrán-Casasas, A. Cammarano, X. Wu, F. Fang, A. E. Porter, K. Fan Chung. Modelling and Analysis of Air Quality in an Underground Station using Differential Equations and Generative Neural Networks . [13]

1.3 Thesis outline

Chapter 2 presents the relevant literature. An overview of neutron transport is provided, including discretisations. The equations and discretisations for the diffusion approximation are also given. The power method, for eigenvalue problems, is also presented. This is followed by a brief overview of multigrid methods. An overview of reduced-order modelling is given, including their applications to neutronics. Machine learning methods, particularly the use of autoencoders, are reviewed next. Finally, domain decomposition is done, before finishing with a brief literature summary.

Chapter 3 presents a method for a neural network solver for the diffusion approximation to the Boltzmann transport equation. This method is demonstrated on GPUs and compared with a standard code, demonstrating equivalent solutions.

Chapter 4 presents an extension of the neural network solver, established in chapter 3, to work for the full Boltzmann transport equation. Additionally, a method utilising higher-order filters and a Petrov-Galerkin diffusion method for stability is presented.

Chapter 5 contains an overview of existing reduced-order modelling methods. The main focus is on Proper Orthogonal Decomposition (POD), a proven traditional method used within all

subsequent chapters, either as the main component or a comparison of methods. An overview of Autoencoders, including various types, is also given.

Chapter 6 presents a method for incorporating autoencoders within projection-based ROMs. This method is tested on simple 1D and 2D eigenvalue problems, comparing the traditional POD and autoencoders. A unique hybrid approach is presented here, combining POD with an AE.

Chapter 7 presents a method for domain decomposition combined with ROM. A multi-group 1D reactor is shown to be able to be modelled this way, with an analysis in how basis functions may be constructed. Motivated by this, further research is done by applying this to a more complicated 2D problem, including using autoencoders.

Chapter 8 presents a method for a Hierarchical ROM, created by interpolating between sub-domains. This approach is done through POD and is applied to a 2D fuel assembly and reactor problem.

Chapter 9 summarises the thesis accomplishments and discusses possible directions for future work stemming from this.

Chapter 2

Literature Review

This section provides a review of relevant literature. An overview of Neutron transport, along with the discretisation of the governing equations, is given. The diffusion approximation and its discretisation are given next. Following this, an overview of multigrid methods and reduced-order modelling is given, including their applications to neutron transport. Next is an overview of machine learning methods related to physics-based problems, focusing on autoencoders, followed by a review of domain decomposition. Finally, a summary of the literature is given.

2.1 Neutron Transport

2.1.1 Governing Equations

An overview of neutron transport methods was done by Lewis [14]. The research done here focuses on only deterministic methods using the Boltzmann transport equation:

$$\begin{aligned}&\frac{1}{v}\frac{\partial}{\partial t}\phi(r,\Omega,E,t)+\Omega\cdot\nabla\phi(r,\Omega,E,t)+\Sigma_t(r,E)\phi(r,\Omega,E,t)=q_{ex}(r,\Omega,E,t)+\\&\int dE^{'}\int d\Omega^{'}\Sigma_s(r,E^{'}\rightarrow E,\Omega^{'}\cdot\Omega)\phi(r,\Omega^{'},E^{'},t)+\\&\chi(E)(1-\beta)\int dE^{'}\nu\Sigma_f(r,E^{'})\phi(r,E^{'},t)+\Sigma_l\chi(E)\lambda_l C_l(r,t)\end{aligned}\tag{2.1}$$

where on the right-hand side $\frac{1}{v}\frac{\partial}{\partial t}\phi(r,\Omega,E,t)$ is the net streaming of particles through the service, $\Omega.\Delta\phi(r,\Omega,E,t)$ are the losses through collisions inside the volume and $\Sigma_t(r,E)\phi(r,\Omega,E,t)$ are

the gains through the emission of neutrons due to fission, scattering or external sources.
On the left-hand side the first term $q_{ex}(r, \Omega, E, t)$ represents the source of particles emitted into the volume by external sources. The second term $\int dE' \int d\Omega' \Sigma_s(r, E' \rightarrow E, \Omega'.\Omega)\phi(r, \Omega', E', t)$ represents the source of scattered particles emitted into the volume. The third term $\chi(E)(1 - \beta) \int dE' \nu\Sigma_f(r, E')\phi(r, E', t)$ is the prompt neutron source (accounting for delayed neutrons β). The final term $\Sigma_l \chi(E)\lambda_l C_l(r, t)$ represents the delayed neutron source with each prescursor type having the denotation l and the equation:

$$\frac{\partial}{\partial t} C_l(r, t) = \beta_l q_f - \lambda_l C_l(r, t). \tag{2.2}$$

Boundary conditions and initial conditions need to be applied to equation (2.1). Vacuum, or bare surface, and reflective boundary conditions are often used.

2.1.2 Discretisation of governing equations

Equation (2.1) is a continuous equation applied to a finite domain. As explored in [14] this must be discretised to be solved by computational methods. The system must be discretised in space, angle, energy and time.

The finite volume method [15, 16] and the finite element method [17, 18] are two methods that can be used to discretise in space. Although the finite volume method could solve complex geometries like the one here, the finite element method is more commonly used. The finite element method splits the computational domain into a number of elements forming the mesh that is used to solve the system. Each of these elements shares information with its neighbouring elements. To improve the computational efficiency, the mesh can be designed to have a greater resolution around points of interest, boundary layers or surface interfaces and coarser resolution elsewhere. Goal-based adaptive meshes, where the mesh is modified during the calculation to optimise criteria, can be used to improve the computational efficiency over time. The accuracy of the count rate measured by the detector is maximised in [19], and the accuracy of k-effective is maximised in [20]. These examples achieve a more accurate solution than a uniform mesh with an equivalent number of nodes. When the solution changes rapidly, and discontinuities occur at element boundaries, the discontinuous Galerkin method [21] can be used to help this.

Angle can be discretised into a number of intervals with a direction vector, called the dis-

crete ordinates method [22] however, ray effects occur if the number of intervals is not large enough [23]. Spherical harmonics [24] is a method that expands the solution by use of a number of functions defined on the surface of the sphere. Wavelets [25] are a transform method where the angular variable is represented with a spherical wavelet basis.

The most commonly used discretisation in energy is the multi-group approximation [14] which splits the range of neutron energies into a number of smaller groups. Each group has its flux defined as the integral quantity within that group. Cross-sections for a group are determined by averaging over the group and using the flux spectrum as a weighting function. Higher energy groups are described as fast, and lower energy groups are described as thermal.

The final discretisation is for the time, and standard linear multi-step methods can be used. Forward and backwards Euler methods are two such methods that could be used. Adaptive time steps, similar to adaptation in space, can be used where the time step is varied during the calculation in order to achieve results that are more accurate and quicker to calculate.

Neutron transport codes have been developed since the 1960s to model neutron transport. WIMS is one of the first, with a general-purpose reactor physics program and multi-group scheme for energy discretisation [26].

Neutron transport can be coupled with computational fluid dynamics (CFD) to produce a complete overview of the physics of a system, which has become significantly easier to achieve with the advances in computational power. FETCH [27] is one example of this. Within FETCH a neutronics code (EVENT) determines the heat input to the fluid, and a CFD code (FLUIDITY) models the heat transfer before returning the temperature to the neutronics code, which can then determine cross-sections by interpolation. A boiling water reactor (BWR) is modelled by Buchan *et al.* [28], and a pressurised water reactor (PWR) is modelled by Jewer *et al.* [29]. The additional computational power required is significant as both codes have their own degrees of freedom. Although modern advances in computational power allow this, further improvements are needed to apply this method to real applications. Smith *et al.* notes that reduced order modelling is one such advancement that could assist with this [30].

2.2 Diffusion Approximation

The multi-group steady-state diffusion equation for criticality can be written as:

$$-\nabla \cdot (D_g \nabla \phi_g) + \Sigma_g^a \phi_g + \sum_{\substack{g'=1 \\ g' \neq g}}^{N_g} \Sigma^s_{g \to g'} \phi_g = \lambda \, \chi_g \sum_{g'=1}^{N_g} \nu_{g'} \Sigma^f_{g'} \phi_{g'} + \sum_{\substack{g'=1 \\ g' \neq g}}^{N_g} \Sigma^s_{g' \to g} \phi_{g'}, \qquad \forall g \in \{1, 2, \ldots, N_g\}, \tag{2.3}$$

where ϕ is the scalar flux of the neutron population, Σ^a represents the absorption cross-section, Σ^f represents the fission cross-section, ν is the average number of neutrons produced per fission event, Σ^s represents the scatter cross-section, χ_g is the proportion of neutrons produced for each energy group per fission event, N_g is the number of energy groups used and the subscript g denotes the energy group. The diffusion coefficient, D_g, is given by:

$$D_g = \frac{1}{3(\Sigma_g^a + \Sigma_g^s)} \, . \tag{2.4}$$

The eigenvalue, λ, is defined as the reciprocal of k_{eff}, that is $\lambda = \frac{1}{k_{\text{eff}}}$, where:

$$k_{\text{eff}} = \frac{\text{number of neutrons in one generation}}{\text{number of neutrons in the preceding generation}}. \tag{2.5}$$

The boundary condition for reflection is:

$$D \frac{\partial \phi}{\partial n} = 0 \, , \tag{2.6}$$

and for a vacuum or bare surface is:

$$-D \frac{\partial \phi}{\partial n} = \frac{1}{2} \phi \, , \tag{2.7}$$

in which n is the outward-pointing normal to the boundary.

2.2.1 Discretisation

A control-volume discretisation of the diffusion equation in 2D with a regular mesh of $N_x \times N_y$ cells can be written as:

$$\begin{aligned}
&\frac{1}{2}\max\left\{D_{i-1,j,g}+D_{i,j,g},\ 0\right\}\frac{\phi_{i,j,g}-\phi_{i-1,j,g}}{\Delta x^2}\ +\ \frac{1}{2}\max\left\{D_{i,j,g}+D_{i+1,j,g},\ 0\right\}\frac{\phi_{i+1,j,g}-\phi_{i,j,g}}{\Delta x^2}\\
&+\frac{1}{2}\max\left\{D_{i,j-1,g}+D_{i,j,g},\ 0\right\}\frac{\phi_{i,j,g}-\phi_{i,j-1,g}}{\Delta y^2}\ +\ \frac{1}{2}\max\left\{D_{i,j,g}+D_{i,j+1,g},\ 0\right\}\frac{\phi_{i,j+1,g}-\phi_{i,j,g}}{\Delta y^2}\\
&+{\Sigma^a}_{i,j,g}\phi_{i,j,g}\ +\ \sum_{\substack{g'=1\\ g'\neq g}}^{N_g}{\Sigma^s}_{i,j,g\to i,j,g'}\phi_{i,j,g}=\chi_g\lambda\sum_{g'=1}^{N_g}\nu_{g'}{\Sigma^f}_{i,j,g'}\phi_{i,j,g'}+\sum_{g'=1}^{N_g}{\Sigma^s}_{i,j,g'\to i,j,g}\phi_{i,j,g'}, \qquad (2.8)\\
&\qquad \forall i\in\{2,3,..,N_x-1\}, \qquad \forall j\in\{2,3,..,N_y-1\},
\end{aligned}$$

in which Δx and Δy are the uniform cell widths in the x- and y-directions respectively, N_x and N_y are the numbers of cells in the x- and y-directions respectively, the subscripts i, j refer to the cells in the x- and y-directions respectively, and $\phi_{i,j}$ represents the scalar flux in cell i, j. In this expression, the first and last cells are omitted in order to apply the boundary conditions efficiently. To apply a reflective boundary condition (see Equation (2.6)), the diffusion coefficients in the outermost cells are set to a large negative number. To set a bare surface boundary condition (see Equation (2.7)), the diffusion coefficients in the outermost cells are again set to a large negative number, and the absorption term is modified as now described. To apply such a boundary condition to the left or right edges (where the normal to the boundary is aligned with the x-direction):

$$\Sigma^a_{i,j,g} \leftarrow \Sigma^a_{i,j,g} + \frac{1}{2\Delta x}\,, \qquad (2.9)$$

and to apply this to the top or bottom edges (where the normal to the boundary is aligned with the y-direction):

$$\Sigma^a_{i,j,g} \leftarrow \Sigma^a_{i,j,g} + \frac{1}{2\Delta y}\,. \qquad (2.10)$$

For cells that have both boundary conditions the following modification is made:

$$\Sigma^a_{i,j,g} \leftarrow \Sigma^a_{i,j,g} + \frac{1}{2\Delta x} + \frac{1}{2\Delta y}\,. \qquad (2.11)$$

The discretised form of Equation (2.3) can be written as:

$$\boldsymbol{A}\boldsymbol{\phi} = \lambda \boldsymbol{B}\boldsymbol{\phi}. \tag{2.12}$$

where matrix $\boldsymbol{A}$ contains the absorption, diffusion and scatter out of energy groups from the left-hand side of Equation (2.8), matrix $\boldsymbol{B}$ represents the fission terms and scatter into energy groups given in the right-hand side of Equation (2.8) and the vector $\boldsymbol{\phi}$ contains the values of the scalar flux for each cell in every energy group. The matrices are of size $(N_x-2)(N_y-2)(N_g)$ by $(N_x-2)(N_y-2)(N_g)$. Although a 2D discretisation is given here, the methods could be applied in 1D or 3D.

The right-hand side of Equation (2.12) can be determined by using a "best guess" for $\boldsymbol{\phi_g}$, effectively linearising the equation which can now be solved by the Jacobi method:

$$\phi_{i,j,g}^{(k+1)} = \frac{1}{a_{i,j,g}} \left(s_{i,j,g} - \sum_{i' \neq i} \sum_{j' \neq j} a_{i',j',g} \phi_{i',j',g}^{(k)} \right), \tag{2.13}$$

where:

$$s_{i,j,g} = \chi_g \lambda \sum_{g'=1}^{N_g} \nu_{g'} \Sigma_{i,j,g'}^{f} \phi_{i,j,g'}^{(k)} + \sum_{g'=1}^{g-1} \Sigma_{i,j,g' \to i,j,g}^{s} \phi_{i,j,g'}^{(k+1)} + \sum_{g'=g}^{N_g} \Sigma_{i,j,g' \to i,j,g}^{s} \phi_{i,j,g'}^{(k)} \ , \tag{2.14}$$

where k is the Jacobi iteration.

2.3 Power Method

The power method [31] is one method to determine the dominant eigenvalue and eigenvector of an eigenvalue problem. Equation (2.12) is an eigenvalue problem that can be solved using this method using an initial guess of ϕ and λ. The flux is updated using Algorithm 1 before being passed to Algorithm 2, the power method, where it is normalised and then both ϕ and λ are updated.This is repeated until either k_{eff} converges, with a tolerance of 10^{-8}, or the maximum number of iterations is reached, 100. When utilising Reduced-Order models Algorithm 1 is replaced with the ROMs.

Algorithm 1 Power method: inner iterations.

function INNER_ITERATIONS($\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{\phi}^{\text{guess}}$, λ^{guess})
  $\boldsymbol{s} = \lambda^{\text{guess}} \boldsymbol{B} \phi^{\text{guess}}$
  solve $\boldsymbol{A}\boldsymbol{\phi} = \boldsymbol{s}$
  return $\boldsymbol{\phi}$

Algorithm 2 Power method: outer iterations.

function OUTER_ITERATIONS($\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{\phi}^{\text{guess}}$, λ^{guess})
  $\boldsymbol{\phi}^{(0)} = \boldsymbol{\phi}^{\text{guess}}$, $\lambda^{(0)} = \lambda^{\text{guess}}$
  $i\_max = 200$
  $k\_tol = 10^{-8}$
  $i = 0$
  `not_converged = True`
  while `not_converged` **do**
    $\boldsymbol{\phi}^{(i+1)} =$ INNER_ITERATIONS$(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{\phi}^{(i)}, \lambda^{(i)})$! *to solve Equation* (2.12)
    $\boldsymbol{\phi}^{(i+1)} \leftarrow \dfrac{\boldsymbol{\phi}^{(i+1)}}{\boldsymbol{b}^T \boldsymbol{B} \boldsymbol{\phi}^{(i+1)}}$! *normalising the flux*
    $\lambda^{(i+1)} = \dfrac{\boldsymbol{b}^T \boldsymbol{A} \boldsymbol{\phi}^{(i+1)}}{\boldsymbol{b}^T \boldsymbol{B} \boldsymbol{\phi}^{(i+1)}}$
    if $(i = i\_max$ **or** $|k_{\text{eff}}^{(i+1)} - k_{\text{eff}}^{(i)}| < k\_tol)$ **then**
      `not_converged = False`
      $\boldsymbol{\phi} \leftarrow \boldsymbol{\phi}^{(i+1)}$
      $\lambda \leftarrow \lambda^{(i+1)}$
    else
      $i \leftarrow i + 1$
  return $\boldsymbol{\phi}$, λ

2.4 High-Fidelity Models

A high-fidelity model for neutron transport is one that can accurately resolve the Boltzmann transport equation 2.1. All of the discretisations outlined in section 2.1.2 must be done so the solution produced is as close to the real solution as possible, potentially requiring a very large number of degrees of freedom to model accurately. This high-fidelity model (HFM) can take a very long time to solve, sometimes taking days to run. The Westinghouse PWR, previously described with 1.7 trillion degrees of freedom [3, 4], is one example of a HFM that would take too long to run for use as a real-time analysis or optimisation tool.

The high-fidelity models utilised within this thesis serve as a comparison for the methodologies presented with the aim of recreating the given HFM solutions as accurately as possible but with significantly less computational cost. For all chapters, the high-fidelity model is only resolved in two dimensions in space and a steady-state solution is formed, in order to test methodologies with computationally inexpensive problems. In Chapters 3, 6, 7 and 8, the diffusion approximation outlined in section 2.2 is utilised for the high-fidelity model. In all chapters, the power method outlined in Section 2.3 is used to determine λ.

2.5 Multigrid methods

Multigrid methods are a way to improve the convergence speed of an iterative method [32, 33]. The convergence speed is improved by solving a hierarchy of discretisations, with each discretisation coarser than the previous one. The coarser discretisations are solved much quicker than, the finer ones, with a fine solution correction occurring every so often. These methods have been well established in fields such as fluid dynamics [34, 35].

As most neutron transport methods use iterative methods to solve them, multigrid methods have been an effective way to speed up the solutions, especially when the real problems have such a large number of degrees of freedom. Larsen provides a unified framework for transport acceleration methods using multigrid algorithms [36]. The multigrid method has been applied to model the sub-grid scale discretisation of the Boltzmann transport equation [37]. The multigrid method has also been utilised with an adaptive mesh in space and angle [38].

2.6 Reduced-Order Modelling

As computational power has advanced over the last few decades, so has computational modelling, which can be seen in the different modelling software available in nearly all areas of scientific research. It has played a key role, alongside theory and experiment, in understanding increasingly complex physical processes. As the complexity of these processes increases, so does that of the model. Computational power has managed to keep up with this increase in most areas, but some, such as neutron transport, have complex models beyond the typical computational power available. Reduced Order Modelling (ROM) is one such method that could assist with this.

Reduced-order modelling or model reduction [6] is a numerical technique that has made a significant impact across a broad range of fields, including aerodynamics [39], haemodynamics [40], fracture [41, 42], porous media [43, 44] and molecular dynamics [45]. ROM aims to significantly reduce the time taken for a solution to be generated by producing a low-dimensional approximation of this high-dimensional system, where the reduction in degrees of freedom significantly reduces the computational cost of producing a solution. The ROM solution should produce a result that is very similar to the HFM.

ROM achieves their computational cost reduction by splitting the computational costs into two phases: Offline and online. The offline phase is where the computationally expensive components are performed, such as generating the HFM or producing basis functions. The online phase should run in real-time or faster and is where new solutions are generated. These phases can be seen below:

- Offline Phase
    1. Produce High-Fidelity Model solutions in the form of snapshots
    2. Produce Basis Functions
    3. Produce Approximation to the discretised governing equations
- Online Phase
    1. Use pre-calculated matrices from the offline stage to approximate the discretised equations for given parameter values.

The efficiency of ROM is based on the two phases format, and this is because it assumes the matrices representing the discretised governing equations can be decomposed into a sum of parameters dependent weights and parameters independent matrices [46]. Snapshots from the online phase are used to modify the basis functions used in [47], demonstrating an alternative approach where the model produced by the offline phase is modified during the online phase. An approach for working with incomplete data, Gappy POD, was first introduced by Everson and Sirovich [48]. This method has been used to reconstruct human faces from incomplete data sets [49] and aerodynamic data [50].

The idea of ROM, and that the process of reducing complexity could be automated has been around since the development of computers [6]. Proper Orthogonal Decomposition (POD) was the first fundamental application of ROM in 1967. Introduced by Lumley [51] for application to fluid dynamics, the majority of advances in this area came from the fields of system and control. Truncated Balanced Realization came from Moore [52] in 1981 and Hankel-norm reduction from Glover [53] in 1984. Further advancements came from Sirovich [54] in 1987.

POD is one of the most popular forms of ROMs that exist, despite the many methods that exist [6]. It has appeared frequently in the last few decades [55–57]. POD can be applied to non-linear partial differential equations, which are common in fluid dynamics, and has since flourished within that field. POD has also been called principal component analysis [58] or Karhunen-Loeve decomposition [59, 60]. The main application of POD in these situations was to identify the main features and trends present within experimental data.

2.6.1 Non-linear systems

For non-linear systems, the non-linear operator can increase the computational complexity of the ROM to the same as the HFM. In order to address this problem, several methods have been developed, including the Quadratic Expansion Method (QEM), Discrete Empirical Interpolation (DEIM) [61, 62] and Residual DEIM.

QEM calculated a number of sub-matrices offline that are used in the online phase using the reduced variables [63]. DEIM samples a number of points in the non-linear operator during the offline phase, that can be interpolated during the online phase. At least as many sampling points as basis functions should be used for a well-posed system [64]. An extension of this is

the mixed DEIM-quadratic formulation of the non-linear operator called residual DEIM [64].

2.6.2 Non-intrusive ROM

All of the previous methods discussed are projection-based [65] which can cause problems if the source code of the HFM is complex (multiphysics applications), protected (licensed software) or legacy code. In these cases, it can be hard to make the necessary modifications to the source code. an alternative method has been developed and is referred to variously as 'non-intrusive reduced-order modelling' (NIROM), 'POD with interpolation', 'Galerkin-free', 'surrogate POD' (see [66] and references within), 'data-driven reduced-order modelling' [67–69] and 'system/model identification' [70, 71]. As in projection-based methods, these non-intrusive methods take a set of high-fidelity model solutions (snapshots) and find a reduced basis, often by using POD. However, to find solutions for unseen parameter values, the projection of the high-fidelity system onto the basis functions is replaced by interpolating between snapshots (or snapshots that have themselves been projected onto the basis functions).

A semi-intrusive method is presented in [72]. Here POD is used to generate data and ISAT algorithms are used to expand and interpolate the data based on discretised partial differential equations. While the offline phase is time-consuming and intrusive the online phase is quicker by three orders of magnitude and is non-intrusive.

In [50] the basis functions generated from POD are interpolated using linear interpolation of the initial conditions. This is effective for their scenario involving aerodynamics and this method was also used more recently in [73] where it was applied to 2D incompressible flow. It is also applied to finite element problems with [74] but additional error analysis is done using non-intrusive methods. This interpolation has also been done on multiple levels in [75] where it is applied to the snapshots and the reduced model.

An alternative approach to NIROM is supplied in [76]. Here an adaptive sampling strategy is employed on data used when creating the POD basis functions. It also highlights the non-interpolating nature of some POD models that exist.

2.6.3 ROM applied to neutron transport

ROM has found great success over the last decade when applied to several areas of reactor modelling. Although the focus of this research is "discretise-then-reduce" ROMs the alternative "reduce-then-discretise" ROMs have been used in reactor physics [77]. Wahi *et al.* uses a lumped parameter model of a BWR to model a point reactor, with the physics greatly simplified before discretisation [78].

Buchan *et al.* demonstrated POD with snapshots on two problems with complex angular geometry, a dog-leg duct problem and emission of radiation from an infinite array of pins [79] successfully resolving the direction of travel of neutrons and photons. The angular solutions form the snapshots and are sampled at locations in the spatial domain. Accuracy is retained by the ROM while also reducing the computational cost by two orders of magnitude.

A ROM based on the reduced basis method was demonstrated by Sartori *et al.* on a 2D model of the TRIGA Mark II reactor test case [80]. A two-dimensional parameter space, consisting of two control rods, is sampled by Voronoi tesselation. This is further advanced by the introduction of an error estimate where they sample the parameter space with a greedy algorithm [81]. They model a time-dependent 3D model of the same reactor previously modelled. They use 70 basis functions, achieving a speed up of three orders of magnitude while only incurring an error of 0.01% in the reactivity.

The reduced basis finite element method is demonstrated by Chunyu *et al.* on the IAEA PWR benchmark problem [82]. They use the fission, removal and scatting cross-sections as parameters to solve the multi-group neutron diffusion equation, successfully reducing the dimensionality by three orders of magnitude.

A detailed model of a Lead-cooled fast reactor was set up using the Monte Carlo code SERPENT was set up by Lorenzi *et al.* [83]. The average cross-sections are used to solve the neutron diffusion PDE. Of the two ROMs compared, the modal method and POD, POD is determined to be more effective. In addition to this, they develop a method that results in a greater computational cost reduction, called the Adjoint Proper Orthogonal Decomposition Method.

Ree *et al.* solves the neutron transport equation and generates snapshots using the discrete generalised multi-group [84]. POD is then used to construct basis functions and use only 6

degrees of freedom per coarse group to achieve a relative error of approximately 0.01% in the fission density.

In [85] produced a number of snapshots of a PWR assembly, with control rod height varying for each snapshot. A ROM based on POD and the method of snapshots was able to predict k-effective for different control rod heights and temperature profiles to an accuracy of 0.01%. Heaney *et al.* also developed a non-linear interpolation method for the same problem, where multiple sets of basis functions are constructed from snapshots of different control-rod heights [86]. This method achieves a slight computational speed up and improves the error by an order of magnitude. In both of these, the ROMs developed a set of basis functions for each energy group, rather than across all energy groups simultaneously, the monolithic approach. As demonstrated by German and Ragusa [87] this method is thought the be more robust than the monolithic approach.

ROM has also been researched in how it could be applied to coupled systems. In [88] ROM is applied to two separate physics systems, the neutronics and thermodynamics, which are then coupled. They demonstrate that a computational reduction can be achieved through this and that thought must be given to ROMs when they are coupled [77] also achieved a computational cost reduction of three orders of magnitude by using POD applied to a neutronics system that was coupled with heat transfer.

Coupling between the Navier-stokes equations and heat transfer equations was done by Georgaka *et al.* by modelling thermal mixing in a T-junction pipe which is found in reactor cooling systems. They use POD with snapshots to achieve a computational speed up of two orders of magnitude and a maximum error of 0.1% for 20 basis functions [89].

2.7 Machine Learning

Since knowledge of machine learning methods has increasingly become more common, and the computational power required to implement them has become easily accessible, the research into their possible uses and applications for physics-based problems has increased tremendously. The libraries available for implementing machine learning, such as TensorFlow [90] and PyTorch [91], have been written for optimal use with hardware such as Graphical Processing Units (GPUs) or even Tensor Processing Units (TPUs). This allows almost any user of these libraries to

implement their methods on this hardware, with no expertise required.

One application is to produce surrogate models of computationally expensive codes, which can produce solutions at a significantly reduced cost. Deep neural networks have become the main surrogate models used in fluid dynamics over the last decade [92, 93], either by themselves or incorporated within a reduced-order model framework [94]. Within fluid dynamics, neural networks have been utilised to approximate steady-state solutions [95] and time-series problems [96].

In the field of nuclear energy, using surrogate models is becoming more popular and has been applied to inverse problems for accident analysis [97], NN for power control of PWRs [98], and Physics-Informed Neural Networks (PINNS) have been developed for point kinetics simulations [99]. In [100], they have been used as ROMs and combined PCA with DNNs for transient analysis, in [101] DNNs were used for neutron cross-section generation, and in [102] a digital twin for neutron diffusion calculations was proposed. PINNS have also been developed across subdomains of problems and used for solving non-smooth heterogeneous neutron diffusion problems [103].

The above methods utilising machine learning are data-driven, requiring training of a network on available data. Often this may result in a network modelling the known physics of a system. An alternative is to utilise these libraries to solve scientific processes on GPUs or TPUs, capitalising on the frameworks already being optimised for this purpose. Previous work doing this includes applications in performing Fourier Transforms [104, 105]; Monte Carlo simulations for finance [106]; many-body quantum physics [107]; and density functional theory [108]. This has also been applied to the Navier-Stokes equation, using the TensorFlow library to implement the discretisations within neural network layers on GPUs [109] and a TPUs [110]. One method [111] implemented the discretisations on convolutional layers for application to the Navier-Stokes equations. These convolutional layers were used as a physics-loss function, which was used to correct the training of the convolutional neural network.

Although machine learning techniques are becoming more prevalent for predicting the behaviour of the HFM (third part of offline stage and online stage in NIROM-type methods), it is much less common to find machine learning algorithms being used in the dimensionality reduction stage (second part of the offline stage in both projection-based or NIROM methods). One

type of neural network that is ideal for dimensionality reduction is the autoencoder. A type of unsupervised (or self-supervised) feed-forward neural network, the autoencoder learns the identity map with a network architecture that includes a bottleneck. The central bottleneck layer forces the autoencoder to learn a reduced or low-dimensional representation of the data. The neurons in this layer determine the so-called latent space and are referred to as latent variables. These types of networks are common in image classification and identification [112, 113], and have been used with great success to fill in gaps in images [114] and to remove noise from data [115].

2.7.1 Autoencoders

The autoencoder can be split into two parts: The encoder (Reducing the number of variables) and the decoder (Increasing the number of variables). Both parts have a number of layers each with an input and output dimension. For each layer, other than the first one, the input dimension is the output dimension of the previous layer. The input dimension for the first encoder layer will be the number of nodes ($\mathcal{N}$) and the output dimension for the last encoder layer will be the reduced number of nodes ($\mathcal{P}$). Similarly, the input of the first decoder layer will be $\mathcal{P}$ and the output of the last decoder layer will be our $\mathcal{N}$. The number of layers and the dimensional change between these layers can vary depending on the complexity of the problem, although the decoder should resemble the inverse of the encoder.

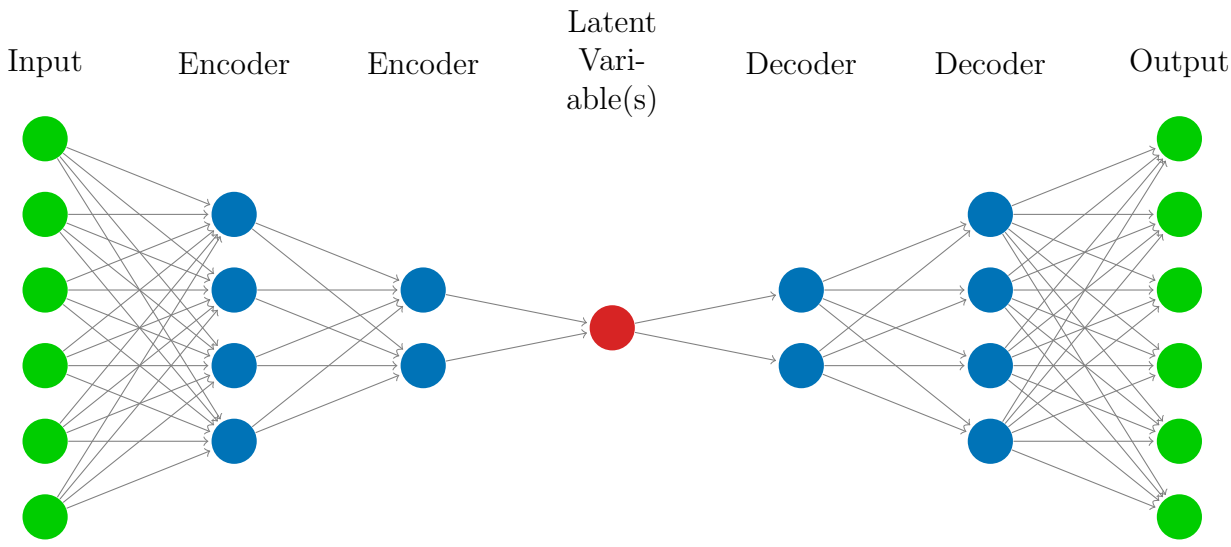


Figure 2.1: An autoencoder comprising of three encoder layers and three decoder layers.

Once the layers have been set up the model is compiled with an optimiser function and a loss function. These dictate how the training is performed and impact the performance of the model. For offline training purposes, assume the discretised forward model has been solved giving results v_i^n where n is the time level and i is the node number, therefore $v^n = (v_1^n, v_2^n, \ldots, v_{\mathcal{N}}^n)^T$ where $\mathcal{N}$ is the total number of nodes. If the forward model variables are passed through the encoder it generates the compressed variables $\alpha^n = (\alpha_1^n, \alpha_2^n, ..., \alpha_{\mathcal{P}}^n)^T$ in which $\mathcal{P}$ is the number of reduced variables (latent variables). If the reduced order variables α^n are passed through the decoder then the forward model variables $u^n = (u_1^n, u_2^n, \ldots, u_{\mathcal{N}}^n)^T$ are produced. If the training process has been successful, then $v^n \approx u^n$. This is illustrated in figure 2.1 for an example system with 3 encoder layers and 3 decoder layers.

Autoencoders have primarily been used to reduce the data of pictures [112] but recently have started being used in fluid dynamics for the reducing data [116], and show great potential for this. Due to being inherently non-linear, the autoencoder is more able to represent complex functions and their evolution through time when compared to the linear SVD. While the SVD is able to find a succinct representation of the space spanned by the snapshots, the autoencoder has the potential to find a more efficient representation. For example, it can represent translation with just one latent variable, whereas an SVD would require many basis functions, each representing one snapshot of the translation. The SVD approach is susceptible to large Gibbs oscillations as it struggles to represent abruptly changing fields. However, autoencoders are likely to require longer time in the offline phase due to the large number of iterations needed to train them. It is also difficult to know the optimum number of variables to reduce with an autoencoder and how many layers to use.

2.7.2 Autoencoders used for ROM

Commonly used in POD is the SVD which decomposes the snapshots into POD basis functions and singular values. To carry out this compression, an autoencoder could be used to replace the SVD. Several authors [117–119] have highlighted the connection between autoencoders and SVD (or PCA), namely that an autoencoder with one hidden layer and *linear* activation functions can produce basis functions than span the same space as basis functions derived from an SVD.

This connection between the autoencoder and the SVD paves the way for an autoencoder to be seen as a tool to perform *nonlinear* dimensionality reduction (by choosing nonlinear activation functions) and therefore to be used as an alternative to the SVD/PCA [112, 120]. Amongst those early to recognise the potential of the autoencoder for dimensionality reduction within a model reduction framework were Milano et al. [121], who reconstructed the near wall field from computational solutions for pressure and shear stress at the wall. The suitability of autoencoders for dimensionality reduction has been established, and a number of authors have since explored their use in reduced-order models, mostly non-intrusive reduced-order models. This is because the strength of the autoencoder is that it is a nonlinear embedding, but this also means that the reduced-order system will be nonlinear and is therefore more complex to solve than for a POD-based ROM. For non-intrusive models, this nonlinearity does not present any additional challenge and no change in the algorithm is required, hence the greater uptake of autoencoders in NIROM-type methods. For example, Gonzalez et al. [116] and Wiewel et al. [122] both use a convolutional autoencoder to reduce the dimension of their problems and Long Short-Term Memory networks to learn the dynamics. The former demonstrates their method on interacting vortices in a 2D box and 2D lid-driven cavity flow. The interacting-vortices problem aims to demonstrate the location invariance properties of convolutional networks. The long-term statistics of the flow, as embodied by the turbulent kinetic energy, are captured much better by the autoencoder-based method than by the projection-based POD-Galerkin model. Coming from the field of computer graphics, Wiewel et al. [122] solve 3D problems with millions of spatial degrees of freedom. With their reduced-order model, they are able to produce realistic-looking simulations of complex dynamical systems such as sloshing waves, colliding bodies of fluid and smoke convection. Their reduced model runs approximately two orders of magnitude faster than their high-fidelity solver.

As previously mentioned, the substitution of an autoencoder for an SVD is less straightforward for projection-based reduced-order models than it is for non-intrusive reduced-order models. There are a couple of simple examples of the incorporation of autoencoders in a projection-based reduced-order model [123, 124], however, the most comprehensive to date is a paper by Lee et al. [125] They propose a framework for projecting dynamical systems onto nonlinear manifolds using deep convolutional autoencoders and derive *a posteriori* error bounds. Focusing on advection-dominated benchmark problems known to be challenging for POD-based ROMs due to a slow decay of singular values [126], they model the convection, diffusion and chemical

reactions of a flame as it evolves in time over a range of values for two parameters which occur in the nonlinear reaction source term. The error in the autoencoder-based method is over 40 times less than that of the standard POD-based projection method, showing that, although their method requires a higher offline computational time (for the training of the autoencoder), they can produce much more accurate results than projection-based methods for some nonlinear problems.

2.8 Domain Decomposition

Domain decomposition is an iterative method developed by H. Schwarz to solve problems with a complicated geometry [127]. This was achieved by splitting a global domain into smaller sub-domains where the geometry in these sub-domains is less complicated than the global domain and has an overlap between sub-domains. Iteration is then performed to solve sub-domains until the solution converges within these sub-domains. The development of modern computers gave this method traction once again, where it was theorised that sub-domains could be solved in parallel and without overlapping [7].

A fast non-overlapping Schwarz domain decomposition was used to solve the neutron diffusion equation applied to a PWR test case by Jamelot and Ciarlet Jr. [128]. The domain was split up into many sub-domains, and different numbers of cores were experimented with, the results showed that utilising parallel processing allowed a significant improvement in the computational time required to solve the problem with no loss in accuracy.

Domain decomposition methods do not change the governing equations of a system, but instead, turn a single domain problem with boundary conditions into multiple sub-domain problems with different boundary conditions. This means that the internal solver used within a sub-domain can be replaced with a ROM. Baiges et al. [129] applied POD in conjunction with domain decomposition to solve the Navier–Stokes equation and reported a computational cost that was two orders of magnitude less than that of the HFM. They have also been combined with NIROM for application to fluid flow [130] and turbulent flow [131].

Domain decomposition methods combined with machine learning have only recently started being investigated, primarily in a non-intrusive framework. Arcucci et al [132] presents the Domain Decomposition Reduced Order Data Assimilation mode, which combines NIROM with

data assimilation. This is applied to a fluid dynamics problem. Heaney et al. [133] developed a domain-decomposition NIROM using autoencoders as a form of dimensionality reduction. This method was applied to multi-phase flow in pipes.

An application to neutron transport was provided by Cherezov et al. [134]. Here, a non-overlapping domain decomposition method was combined with POD and applied to a full reactor core, which was decomposed into sub-domains containing fuel assemblies. Three by three and 2×2 clusters of these sub-domains were solved. These solutions were then used as the snapshots from which to construct the basis functions for different types of fuel assemblies. They showed that combining these methods can produce an accurate solution of the global system, with a significant speed up, by solving four different layouts or arrangements of the reactor core, in each case using the same basis functions produced from the clusters of sub-domains.

2.9 Summary

The Boltzmann transport (equation (2.1)) is a computationally demanding problem requiring discretisation in seven dimensions, three in space, two in angle, one in energy and one in time. This is further exacerbated by the coupling of physics models, which significantly increases the degrees of freedom required, as each physics model requires their own. This coupling is required to accurately model an entire reactor, a requirement for real-time and safety analysis. Due to the lack of computational power typically available, nuclear modelling cannot be done in a reasonable amount of time. To take advantage of the benefits of computational modelling, methods to reduce computational costs or enhance the accessibility of hardware must be implemented.

Reduced-Order modelling (ROM) has seen great success in almost all fields, nuclear modelling included. These methods are typically applied to the global domain, requiring a large amount of the behaviour of a system to be modelled at once. A reactor typically has repeated areas, in the form of fuel rods or fuel assemblies, that will have similar behaviours. A ROM that can take advantage of this through repeated smaller ROMs may significantly reduce the computational cost in both the online and offline phases. The modelling of reactors could greatly benefit from a way to implement this.

Machine learning has seen some incredible growth and success over the last decade. These methods have been used successfully in physics-based areas, as surrogate models and for ROM methods. However, these have mainly been applied outside the nuclear area, fluid dynamics being one of the most popular. Nuclear modelling could benefit greatly from these developments.

Chapter 3

Diffusion Neural Network Solver

The content within this chapter is based on:

T. R. F. Phillips, C. E. Heaney, B. Chen, A. G. Buchan and C. C. Pain. Solving the discretised neutron diffusion equations using neural networks. *International Journal for Numerical Methods in Engineering* [8]

Chapter 2 contains methods for how the high-fidelity snapshots of this chapter were generated.

3.1 Introduction

As new hardware develops, so does the need for software to be adapted for use with it. This can be more difficult within computational modelling, as existing code is typically written for serial execution on Central Processing Units (CPUs). Therefore, adapting this code to utilise the parallel capabilities of Graphical Processing Units (GPUs), or even Tensor Processing Units (TPUs), can be difficult without specific expertise that many researchers will lack. With the significant increase in interest in machine learning comes a similar increase in familiarity with AI libraries, such as TensorFlow [90] and PyTorch [91]. Hardware developments, such as the TPUs of Google [135], or the CS-2 of Cerebras [136], could significantly benefit the computational modelling community, yet which can not be utilised effectively with current codes. The CS-2 [136] has about 1 million cores on a single chip with vastly increased computational speed. Yet, it requires much less energy than GPUs or CPUs, making it ideal hardware for researchers

to run computationally demanding problems in an energy-efficient manner. If the potential of combining the new AI computers and AI software with traditional numerical methods can be harnessed, one can expect a revolution in computational physics across disciplines. The approach presented here implements conventional discretisations using AI software to simplify code development and exploit the substantial developments already made in AI technology, such as code abstraction and platform agnosticism. This simplification could significantly increase the number of researchers able to become more involved in developing current models (each code is often limited to a handful of expert developers), speeding up its implementation and parallel scalability.

Incorporating known physics within neural networks is not new, with physics-informed machine learning [137] being a significant development. This paved the way for the use of deep neural networks (DNN) to form solutions to both forward and inverse problems of systems described by PDEs. Early works include using data from passive dyes, smoke or vapour to extract solutions to the Navier-Stokes equations [138, 139]. Others include the use of Physics-Informed Neural Networks (PINNs) for analysing subsurface transport [140], and for other fluid problems where the mix of computational (LES-CFD) and experimental data (particle image velocimetry) was used to identify missing quantities in turbulent flows [141]. These methods still utilise training the neural network, incorporating the physics in the loss functions, resulting in a network that needs to be re-trained for new systems.

An alternative is to implement the standard numerical scheme directly into the neural network, thereby avoiding the training that is usually required. This has been applied to the Navier-Stokes equation, implementing the advection term using convolutional layers [109]. Using the TensorFlow library, they achieve a significant speed up using GPUs over the original Fortran-based code running on CPUs. A similar approach has been used with TPUs, implementing a finite-difference discretisation of the Navier-Stokes equations [110].

The approach described in this chapter is a new and alternative way of harnessing AI technologies for forming solutions for governing PDEs. Here, it is proposed to perform the computations that form the solution from standard discretisation techniques through a single neural network, utilising convolutional layers for the diffusion term. These are implemented to replicate a multigrid solver, incorporating a similar architecture to the U-Net [142]. The results here demonstrate that the solutions are identical to the discretised model but allow the use of

GPUs, without the requirement of writing parallel code. Using a GPU with neural network code written in Python results in a computation time that is one-quarter that of using a CPU with a solver written in Fortran.

Additionally, performing all operations through the neural network allows optimal code efficiency on all architectures whilst enabling the latest developments in methods such as sensitivities [143], uncertainty quantification [144] and data assimilation [145], thanks to the neural network architecture. Therefore, the research proposed here has the potential to cause a shift in how numerical models are developed and used. The beneficiaries will be wide-ranging, including researchers or organisations requiring PDE models and associated AI-Physics models. The ability to rely on high-level libraries that seamlessly integrate AI tools with PDE models will reduce development time with increased performance portability and parallel scalability.

The chapter is organised as follows: Section 3.2 contains the methodology, section 3.3 contains an overview of the results, and finally, a summary of the chapter is given in section 3.4.

3.2 Methodology

The first part of this section introduces the governing equations and their discretisation. It is explained how the discretisation can be written in the form of convolutional operations with pre-defined weights. The Jacobi method and its neural network equivalent is explained next. Convergence can be improved by using a multigrid method, and this can be implemented using a convolutional autoencoder, in particular, the U-Net. Finally, an overview of the multi-group solver is done, including how the eigenvalue is determined.

3.2.1 Discretisation

A control-volume discretisation of the diffusion equation in 2D with a regular mesh of $N_x \times N_y$ cells can be written as:

$$\begin{aligned}
&\frac{-(D_{i,j,g}+D_{i-1,j,g})}{2\Delta x^2}\phi_{i-1,j,g} - \frac{(D_{i,j,g}+D_{i+1,j,g})}{2\Delta x^2}\phi_{i+1,j,g} - \frac{(D_{i,j,g}+D_{i,j-1,g})}{2\Delta y^2}\phi_{i,j-1,g} \\
&-\frac{(D_{i,j,g}+D_{i,j+1,g})}{2\Delta y^2}\phi_{i,j+1,g} + \left(\frac{(D_{i+1,j,g}+2D_{i,j,g}+D_{i-1,j,g})}{2\Delta x^2} + \frac{(D_{i,j-1,g}+2D_{i,j,g}+D_{i,j+1,g})}{2\Delta y^2}\right)\phi_{i,j,g} \\
&+\Sigma^a{}_{i,j,g}\phi_{i,j,g} \;+\; \sum_{\substack{g'=1 \\ g'\neq g}}^{N_g} \Sigma^s_{i,j,g\to i,j,g'}\phi_{i,j,g} = \sum_{g'=1}^{N_g} \Sigma^s_{i,j,g'\to i,j,g}\phi_{i,j,g'} + \lambda\chi_g \sum_{g'=1}^{N_g} \nu_{g'}\Sigma^f_{i,j,g'}\phi_{i,j,g'}, \qquad (3.1) \\
&\forall i \in \{2,3,\ldots,N_x-1\}, \qquad \forall j \in \{2,3,\ldots,N_y-1\}, \qquad \forall g \in \{1,2,\ldots,N_g\},
\end{aligned}$$

in which Δx and Δy are the uniform cell widths in the x and y directions respectively, N_x and N_y are the numbers of cells in the x and y directions respectively, the subscripts i and j refer to the cells in the x and y directions respectively, N_g is the number of energy groups, the subscript g refers to the energy group and $\phi_{i,j,g}$ represents the scalar flux in energy group g in cell i,j.

The boundary conditions are applied to the first and last cells in both the x and y directions, so Equation (3.1) is not solved for these cells. Reflective boundary conditions for the left edge ($i = 1$) would be enforced by the following constraints:

$$\phi_{1,j,g} = 0, \quad D_{2,j,g} = -D_{1,j,g} \qquad (3.2)$$

and for the right edge ($i = N_x$):

$$\phi_{N_x,j,g} = 0, \quad D_{N_x,j,g} = -D_{N_x-1,j,g}\,. \qquad (3.3)$$

Equivalent constraints can be applied to the top and bottom edges. For bare surface boundary conditions (Equation (2.7)), where the normal to the boundary is aligned with the x-direction, the absorption term is modified as follows:

$$\Sigma^a_{i,j,g} \leftarrow \Sigma^a_{i,j,g} + \frac{1}{2\Delta x}\,. \qquad (3.4)$$

To apply this to where the normal to the boundary is aligned with the y direction:

$$\Sigma^a_{i,j,g} \leftarrow \Sigma^a_{i,j,g} + \frac{1}{2\Delta y}\,. \tag{3.5}$$

For cells that have both boundary conditions the following modification is made:

$$\Sigma^a_{i,j,g} \leftarrow \Sigma^a_{i,j,g} + \frac{1}{2\Delta x} + \frac{1}{2\Delta y}\,. \tag{3.6}$$

Equation (3.1) and its associated boundary conditions are often written as:

$$\boldsymbol{A}\boldsymbol{\phi} = \lambda \boldsymbol{B}\boldsymbol{\phi}. \tag{3.7}$$

where the matrix $\boldsymbol{A}$ contains the absorption, diffusion and scattering out of energy groups from the left-hand side of Equation (3.1), matrix $\boldsymbol{B}$ represents the fission terms and scattering into energy groups given in the right-hand side of Equation (3.1) and the vector $\boldsymbol{\phi}$ contains the values of the scalar flux for each cell in every energy group. In the following, we keep with the notation used thus far, which stores the unknown scalar flux in a 2D array. The diffusion terms in Equation (3.1) could be considered as components of a fourth-order tensor. Although this way of formulating the problem may seem less familiar, the motivation will become clear in the following section, when we compare discretisation stencils to convolutional operators. Bearing this in mind, we rewrite the system in Equation (3.1) as

$$\sum_{u=-1}^{1}\sum_{v=-1}^{1} a^{u,v}_{i,j,g}\,\phi_{i+u,j+v,g} = s_{i,j,g}\,, \quad \forall i \in \{2,3,\ldots,N_x{-}1\},\ \forall j \in \{2,3,\ldots,N_y{-}1\},\ \forall g \in \{1,2,\ldots,N_g\}, \tag{3.8}$$

where:

$$
a_{i,j,g}^{u,v} = \begin{cases} -\dfrac{D_{i,j,g}+D_{i+u,j+v,g}}{2\Delta x^2} & \text{for } |u|=1,\ v=0 \\ -\dfrac{D_{i,j,g}+D_{i+u,j+v,g}}{2\Delta y^2} & \text{for } u=0,\ |v|=1 \\ \dfrac{D_{i+1,j,g}+2D_{i,j,g}+D_{i-1,j,g}}{2\Delta x^2}+\dfrac{D_{i,j+1,g}+2D_{i,j,g}+D_{i,j-1,g}}{2\Delta y^2}+\Sigma^a{}_{i,j,g}+\displaystyle\sum_{\substack{g'=1\\ g'\neq g}}^{N_g}\Sigma^s_{i,j,g\to i,j,g'} & \text{for } u=0=v \\ 0 & \text{for } |u|=1=|v| \end{cases} \tag{3.9}
$$

$$
s_{i,j,g} = \lambda\chi_g \sum_{g'=1}^{N_g} \nu_{g'}\Sigma^f_{i,j,g'}\phi^{(k)}_{i,j,g'} + \sum_{g'=1}^{g-1}\Sigma^s_{i,j,g'\to i,j,g}\phi^{(k+1)}_{i,j,g'} + \sum_{g'=g}^{N_g}\Sigma^s_{i,j,g'\to i,j,g}\phi^{(k)}_{i,j,g'}\ . \tag{3.10}
$$

The right-hand side of Equation (3.8) can be determined by using a "best guess" for $\phi_{i,j,g}$. This effectively linearises Equation (3.8) which can now be solved by the Jacobi method:

$$
\phi^{(k+1)}_{i,j,g} = \frac{1}{a^{0,0}_{i,j,g}}\left(s_{i,j,g} - \sum_{u=-l}^{l}\sum_{v=-l}^{l} a^{u,v}_{i,j,g}\,\phi^{(k)}_{i+u,j+v,g} + a^{0,0}_{i,j,g}\phi^{(k)}_{i,j,g}\right), \tag{3.11}
$$

where $2l+1$ is the width of the stencil ($l=1$ for linear elements), k is the Jacobi iteration and $\{\{a^{u,v}_{i,j,g}\}^{l}_{u=-l}\}^{l}_{v=-l}$ represents the stencil to calculate the scalar flux in cell i, j. The diagonal terms in the usual matrix-vector form of Equation (3.11) are now denoted by $a^{0,0}_{i,j,g}$ and the non-diagonal terms are subtracted from the source term.

3.2.2 Neural network as a Jacobi Solver

This section describes how one iteration of a Jacobi method can be implemented as a convolutional layer of a neural network using pre-determined weights. A convolutional layer has a filter or kernel, which is a small grid (smaller than the input data and typically of dimension 3×3, 5×5 or 7×7) whose cells have values known as weights associated with them. The filter is applied to part of the input by multiplying the input value by the weight in the overlapping cells. The products are summed to produce the output. The filter is then applied to a neighbouring part of the input and another output is created. This process is repeated until the filter has passed over all the input data. The action of a 2D convolutional layer on a 2D

input can be written as follows

$$x_{i,j}^{(k+1)} = \sum_{u=-l}^{l} \sum_{v=-l}^{l} w^{u,v}\, x_{i+u,j+v}^{(k)}, \tag{3.12}$$

where the input and output are 2D grids with components $x_{i,j}^{(k)}$ and $x_{i,j}^{(k+1)}$ respectively. The weights of the filter are represented by $w^{u,v}$ and the size of the filter is $(2l+1) \times (2l+1)$ — in this case, $l = 1$. A filter acting on one piece of input data can be seen in Figure 3.1.

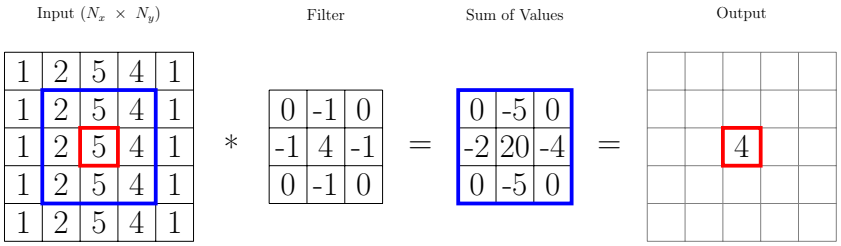


Figure 3.1: A convolutional filter which applies the discretised diffusion operator (a five-point control volume stencil) in 2D to 9 cells. The filter is first applied to all the cells in the blue block on the left; the result of which can be seen in the blue block following the equals sign. The 9 values are then summed to give the value in the red block on the right which is the output value.

Given the previous definition of a filter, we can define a function $\boldsymbol{f}$ which represents that filter passing over the entire input as follows

$$\boldsymbol{f}(\boldsymbol{\Phi}; \boldsymbol{w})\,|_{i,j} = -\frac{\phi_{i-1,j,g}}{2\Delta x^2} + \frac{\phi_{i,j,g}}{\Delta x^2} - \frac{\phi_{i+1,j,g}}{2\Delta x^2} - \frac{\phi_{i,j-1,g}}{2\Delta y^2} + \frac{\phi_{i,j,g}}{\Delta y^2} - \frac{\phi_{i,j+1,g}}{2\Delta y^2} \quad \forall i, j\,, \tag{3.13}$$

for a filter with weights $\boldsymbol{w}$

$$\boldsymbol{w} = -\frac{1}{2}\begin{bmatrix} 0 & \frac{1}{\Delta y^2} & 0 \\ \frac{1}{\Delta x^2} & \frac{-2}{\Delta x^2} + \frac{-2}{\Delta y^2} & \frac{1}{\Delta x^2} \\ 0 & \frac{1}{\Delta y^2} & 0 \end{bmatrix}. \tag{3.14}$$

This function calculates the second derivative of the input $\boldsymbol{\Phi}_g$ according to the lowest order control volume discretisation. For two scalars ϕ_g and D_g, the following is true

$$\begin{aligned} \nabla^2(D_g\phi_g) &= \nabla \cdot \nabla(D_g\phi_g) && (3.15)\\ &= \nabla \cdot (\phi_g \nabla D_g + D_g \nabla \phi_g) && (3.16)\\ &= \phi_g \nabla^2 D_g + D_g \nabla^2 \phi_g + 2\nabla D_g \cdot \nabla \phi_g\,, && (3.17) \end{aligned}$$

which leads to the following identity

$$2\nabla D_g \cdot \nabla\phi_g = \nabla^2(D_g\phi_g) - \phi_g\nabla^2 D_g - D_g\nabla^2\phi_g \,. \tag{3.18}$$

Expanding out the diffusion term in Equation (2.3) and then substituting in the expression from Equation (3.18) results in

$$\begin{aligned}
\nabla \cdot (D_g\nabla\phi_g) &= D_g\nabla^2\phi_g + \nabla\phi_g \cdot \nabla D_g && (3.19)\\
&= D_g\nabla^2\phi_g + \frac{1}{2}\left(\nabla^2(D_g\phi_g) - \phi_g\nabla^2 D_g - D_g\nabla^2\phi_g\right) && (3.20)\\
&= \frac{1}{2}\left(\nabla^2(D_g\phi_g) - \phi_g\nabla^2 D_g + D_g\nabla^2\phi_g\right) \,. && (3.21)
\end{aligned}$$

Equation (3.21) is used in Equation (3.22). We can therefore write the diffusion operator in terms of three convolutional layers:

$$\begin{aligned}
-\nabla \cdot (D_g\nabla\phi_g) &= -\frac{1}{2}\left(\nabla^2(D_g\phi_g) + D_g\nabla^2\phi_g - \phi_g\nabla^2 D_g\right) && \textit{analytical form} \quad (3.22)\\
&\sim \boldsymbol{f}(\boldsymbol{D}_g \odot \boldsymbol{\Phi}_g; \boldsymbol{w}) + \boldsymbol{D}_g \odot \boldsymbol{f}(\boldsymbol{\Phi}_g; \boldsymbol{w}) - \boldsymbol{\Phi}_g \odot f(\boldsymbol{D}_g; \boldsymbol{w}) && \textit{discretised form}\\
& && (3.23)\\
&= \boldsymbol{f}^{\text{Diff}}(\boldsymbol{\Phi}_g, \boldsymbol{D}_g; \boldsymbol{w}) \,, && (3.24)
\end{aligned}$$

where $\boldsymbol{\Phi}_g$ is a $N_x \times N_y$ matrix containing all $\phi_{i,j,g}$ components, $\odot$ denotes the Hadamard product which performs entrywise multiplication, $\boldsymbol{D}_g$ is a $N_x \times N_y$ matrix containing all $D_{i,j,g}$ components, the equation above serves as a definition of the diffusion convolution $\boldsymbol{f}^{\text{Diff}}$, and $\boldsymbol{f}$ represents the application of the convolutional layer with weights $\boldsymbol{w}$. The terms in Equation (3.23) are shown to be the same as in Equation (3.22). The discretised diffusion equation can now be written for energy group g as:

$$\boldsymbol{f}^{\text{Diff}}(\boldsymbol{\Phi}_g, \boldsymbol{D}_g; \boldsymbol{w}) + \left(\boldsymbol{\Sigma}^{\boldsymbol{a}}{}_g \;+\; \sum_{\substack{g'=1\\ g'\neq g}}^{N_g} \boldsymbol{\Sigma}^{\boldsymbol{s}}_{\boldsymbol{g\to g'}}\right) \odot \boldsymbol{\Phi}_g = \boldsymbol{s}_g, \quad \forall g \in \{1, 2, \ldots, N_g\}, \tag{3.25}$$

in which the source $\boldsymbol{s}_g$ for energy group g also contains coupling terms between the energy groups other than g. Equation (3.25) can be solved with the Jacobi method as before. However, when implementing this, instead of using Equation (3.25), we rewrite this to use one fewer

convolutional operation for efficiency. Equation (3.11) can be solved using a neural network with pre-determined filters. The term $\sum_{u=-l}^{l}\sum_{v=-l}^{l} a_{i,j,g}^{u,v}\,\phi_{i+u,j+v,g}^{(k)} - a_{i,j,g}^{0,0}\,\phi_{i,j,g}^{(k)}$ can be determined using a convolutional filter containing just the off-diagonal terms:

$$\boldsymbol{D}_g \odot \boldsymbol{f}(\boldsymbol{\Phi}_g^{(k)}; \boldsymbol{w_D})\,|_{i,j,g} + \boldsymbol{f}(\boldsymbol{D}_g \odot \boldsymbol{\Phi}_g^{(k)}; \boldsymbol{w_D})\,|_{i,j,g} = \sum_{u=-l}^{l}\sum_{v=-l}^{l} a_{i,j,g}^{u,v}\,\phi_{i+u,j+v,g}^{(k)} - a_{i,j,g}^{0,0}\,\phi_{i,j,g}^{(k)}, \quad (3.26)$$

where $\boldsymbol{f}$ is a convolutional pass with weights filter $\boldsymbol{w_D}$:

$$\boldsymbol{w_D} = -\frac{1}{2}\begin{bmatrix} 0 & \frac{1}{\Delta y^2} & 0 \\ \frac{1}{\Delta x^2} & 0 & \frac{1}{\Delta x^2} \\ 0 & \frac{1}{\Delta y^2} & 0 \end{bmatrix}, \quad (3.27)$$

which is the same as filter $\boldsymbol{w}$ but with the central term set to zero, to prevent additional unnecessary operations when performing a Jacobi iteration. Equation (3.11) is therefore equivalent to:

$$\boldsymbol{\Phi}_g^{(k+1)} = (\boldsymbol{A}_g^{0,0})^{\odot -1} \odot (\boldsymbol{s}_g - \boldsymbol{D}_g \odot \boldsymbol{f}(\boldsymbol{\Phi}_g^{(k)}; \boldsymbol{w_D}) + \boldsymbol{f}(\boldsymbol{D}_g \odot \boldsymbol{\Phi}_g^{(k)}; \boldsymbol{w_D})), \quad (3.28)$$

where $(\boldsymbol{A}_g^{0,0})^{\odot -1}$ is the Hadamard inverse [146] which is an $N_x \times N_y$ array whose $i^{\text{th}}, j^{\text{th}}$ component is $\frac{1}{a_{i,j,g}^{0,0}}$ for energy group g.

Figure 3.2 details the architecture for a neural network that performs the same operation as Equation (3.28). This can be written as:

$$\boldsymbol{\Phi}_g^{(k+1)} = \mathrm{J}\left(\boldsymbol{\Phi}_g^{(k)}, (\boldsymbol{A}_g^{0,0})^{\odot -1}, \boldsymbol{s}_g, \boldsymbol{D}_g\right), \quad (3.29)$$

where $\mathrm{J}(\,\cdot\,)$ is a function that calculates the result of one Jacobi iteration and $\boldsymbol{s_g}$ is a 2D matrix containing all $s_{i,j,g}$ components. Figure 3.2 shows the architecture of the neural network with a single Jacobi iteration that solves the neutron transport problem. Green boxes contain the inputs, blue boxes are convolutional layers, orange boxes are mathematical functions as layers and the grey box is the output of the network. The second line in each box is the dimension of the output.

We can also use more accurate stencils for the diffusion operator. For example, the recently developed Convolution Finite Element Method (ConvFEM) approach, see [9], for quadratic elements results in the 5×5 filter:

$$\boldsymbol{w} = \frac{10^{-2}}{9}\begin{bmatrix} -5 & 50 & -15 & 50 & -5 \\ 50 & -320 & -660 & -320 & 50 \\ -15 & -660 & 3600 & -660 & -15 \\ 50 & -320 & -660 & -320 & 50 \\ -5 & 50 & -15 & 50 & -5 \end{bmatrix}. \tag{3.30}$$

However, any Finite Element, Finite Difference or Control Volume discretisation stencil could equally be used here. Using a higher-order filter requires additional halo nodes, so N_x and N_y are larger. For the quadratic filters this means that equation (3.1) is solved for $\forall i \in \{3, 4, \ldots, N_x - 2\}$ and $\forall j \in \{3, 4, \ldots, N_y - 2\}$. The boundary conditions change when using higher-order filters. Assuming that the material parameters are homogeneous around the entire boundary for at least two cells deep then boundary conditions can be enforced easily with the same approach as equations (3.2) and (3.3). If D_{left}, D_{right}, D_{top} and D_{bottom} are the diffusion coefficients for the left, right, top and bottom sides respectively, then this is true if:

$$D_{left} = D_{right} = D_{top} = D_{bottom}, \tag{3.31}$$

and for the left side:

$$D_{left} = D_{i,j,g} \quad \forall i \in \{3, 4\} \quad \forall j \in \{3, 4, \ldots, N_y - 2\}, \tag{3.32}$$

with corresponding constraints for the right, top and bottom sides. If equation (3.31) holds and (3.32) holds for all sides then the boundary conditions for the left side, $i = 1$, can be implemented with:

$$\phi_{1,j,g} = 0, \quad \phi_{2,j,g} = 0, \quad D_{left} = -D_{1,j,g} = -D_{2,j,g}, \tag{3.33}$$

and similar conditions for the other sides. Boundary conditions (3.4), (3.5) and (3.6) are all implemented the same way. Equation (3.32) may not hold if two cells/node next to the boundary do not have the same value and thus this approach might not be used in this situation or one may use some sort of average. An alternative that works for all filter sizes is simply to set the halo values of the diffusion coefficient and the fluxes to be zero and then no addition to the

absorption cross sections for the boundary condition are required. This effectively implements the $2\Delta x$ extrapolation boundary condition obtained using Equations (3.4), (3.5) and (3.6).

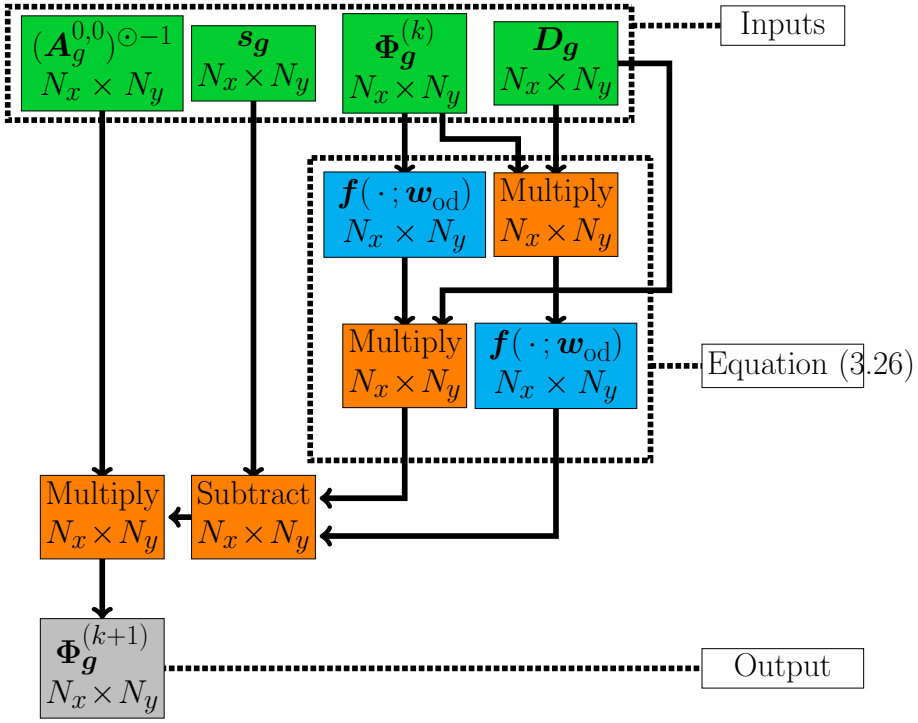


Figure 3.2: Schematic of the neural network used for a Jacobi iteration in which J($\cdot$) represents a single Jacobi iteration written in Equation (3.28). This network performs a single Jacobi iteration on the flux of a single energy group. The inputs are the flux ($\boldsymbol{\Phi}_g^{(k)}$), representative source ($\boldsymbol{s}_g$), diffusion coefficients ($\boldsymbol{D}_g$) and the strictly diagonal coefficients ($\boldsymbol{A}_{\boldsymbol{g}}^{\boldsymbol{0,0}})^{\odot -1}$) (green boxes). A number of layer operations are performed, mathematical operations are shown in orange and convolutional operations are in cyan. The output is the flux of the next Jacobi iteration ($\boldsymbol{\Phi}_g^{(k+1)}$). Arrows originate from which layer the data originated and the end of an arrow indicates which layer takes that data as input. The dimensions of the layers are given on the second line of each box.

3.2.3 Multigrid

In Figure 3.3 we show how the commonly applied U-Net [142] ANN architecture has been repurposed to form a saw tooth multigrid method and is used in this work. Figure 3.4 shows a single multigrid iteration, using the U-Net, with two restrictions, note that the subscript indicating energy group is no longer shown, the bold subscript now indicates the coarseness of the mesh, with 1 being the finest mesh and becoming more coarse as the subscript increases. The residual ($\boldsymbol{r^k}$) is calculated using:

$$\boldsymbol{r}_{\mathbf{1}}^{(k)} = \lambda \boldsymbol{B}_{\mathbf{1}} \boldsymbol{\Phi}_{\mathbf{1}}^{(k)} - \boldsymbol{A}_{\mathbf{1}} \boldsymbol{\Phi}_{\mathbf{1}}^{(k)}, \tag{3.34}$$

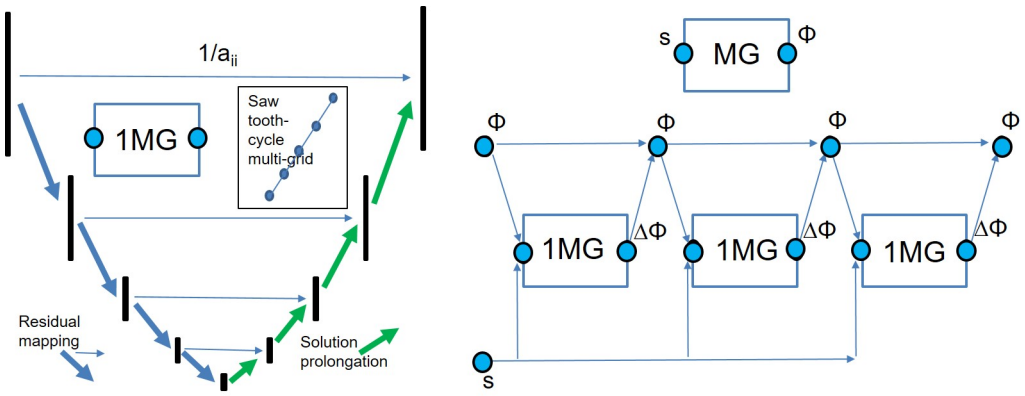


Figure 3.3: Schematics showing the U-Net ANN architecture (left) that is used to form a single multigrid saw tooth cycle. On the right, we can see how multiple cycles are brought together to form the overall solution method.

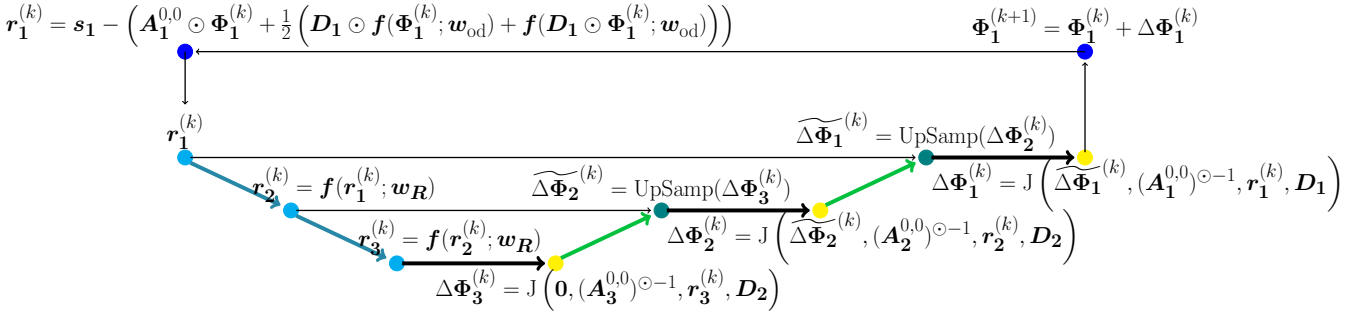


Figure 3.4: Multigrid iteration, with the subscript indicating the resolution and superscript representing the multigrid iteration and J($\cdot$) representing a Jacobi iteration. The residual is calculated and restricted twice, indicated by the cyan nodes. These residuals are used with Jacobi smoothing, indicated by yellow nodes. After smoothing, prolongation is performed using UpSampling layers, indicated by teal nodes. After the finest level is reached, the flux is updated and the process repeats.

$\boldsymbol{r}_1^{(k)}$ is restricted twice to $(\boldsymbol{r}_2^{(k)})$ and $(\boldsymbol{r}_3^{(k)})$. A Jacobi iteration is performed on the lowest level (bold subscript 3) to determine $\Delta\boldsymbol{\Phi}_3^{(k)}$, starting with an array of zeros. This is prolongated to estimate $\widetilde{\Delta\boldsymbol{\Phi}_2}^{(k)}$ which is then smoothed to $\Delta\boldsymbol{\Phi}_2^{(k)}$ with another Jacobi iteration using the residual of the next highest level. This repeats until the finest level is reached (bold subscript 1), where the flux is updated $(k+1)$ and the process is repeated for a number of multigrid iterations.

The restriction may be performed with the convolution:

$$\boldsymbol{r}_2^{(k)} = \boldsymbol{f}(\boldsymbol{r}_1^{(k)}; \boldsymbol{w}_R), \tag{3.35}$$

with filter weights:

$$\boldsymbol{w_R} = \begin{bmatrix} 0.25 & 0.25 \\ 0.25 & 0.25 \end{bmatrix}. \tag{3.36}$$

Upsampling layers may perform the role of prolongating the solution up from a level. Upsampling layers repeat values within an array, increasing the dimensions of the data [147], resulting in an approximation for the data at a finer mesh:

$$\widetilde{\boldsymbol{\Phi}_{\mathbf{1}}}^{(k)} = \text{UpSamp}(\boldsymbol{\Phi}_{\mathbf{2}}^{(k)}). \tag{3.37}$$

The upsampling operations simply copy the value from the course grid node or cell to the four nodes/cells that the nodal values are prolonged from.

Figure 3.5 shows how Figure 3.4 would look like a single network. Green boxes contain the inputs, blue boxes are convolutional layers, orange boxes are mathematical functions as layers, yellow boxes are sub-networks, teal boxes are upsampling layers and the grey box is the output of the network. The second line in each box is the dimension of the output. This can be written as:

$$\boldsymbol{\Phi}_{\mathbf{1}}^{(k+1)} = \text{MG}\left(\boldsymbol{\Phi}_{\mathbf{1}}^{(k)}, \boldsymbol{s_1}, (\boldsymbol{A}_{\mathbf{1}}^{0,0})^{\odot -1}, (\boldsymbol{A}_{\mathbf{1}}^{0,0})^{\odot -1}, (\boldsymbol{A}_{\mathbf{2}}^{0,0})^{\odot -1}, (\boldsymbol{A}_{\mathbf{3}}^{0,0})^{\odot -1}, \boldsymbol{D_1}, \boldsymbol{D_2}, \boldsymbol{D_3}\right), \tag{3.38}$$

and for a single energy group g:

$$\boldsymbol{\Phi}_{\mathbf{1}g}^{(k+1)} = \text{MG}_g\left(\boldsymbol{\Phi}_{\mathbf{1}g}^{(k)}, \boldsymbol{s_{1}}_g, (\boldsymbol{A}_{\mathbf{1}g}^{0,0})^{\odot -1}, (\boldsymbol{A}_{\mathbf{1}g}^{0,0})^{\odot -1}, (\boldsymbol{A}_{\mathbf{2}g}^{0,0})^{\odot -1}, (\boldsymbol{A}_{\mathbf{3}g}^{0,0})^{\odot -1}, \boldsymbol{D_1}_g, \boldsymbol{D_2}_g, \boldsymbol{D_3}_g\right), \tag{3.39}$$

where MG($\cdot$) is a function that calculates the result of one sawtooth multigrid iteration and many of these iterations are strung together to form the final solution, see Figure 3.3. MG$_g$($\cdot$) is the multigrid iteration applied to energy group g, indicated by the subscript.

It should be noted that the diffusion coefficients and other material properties are mapped to a coarser grid using a harmonic average of these coefficients before the discretisation on the coarser grids are formed. The same discretisation is used on each multigrid level but with different cell or node sizes Δx, Δy.

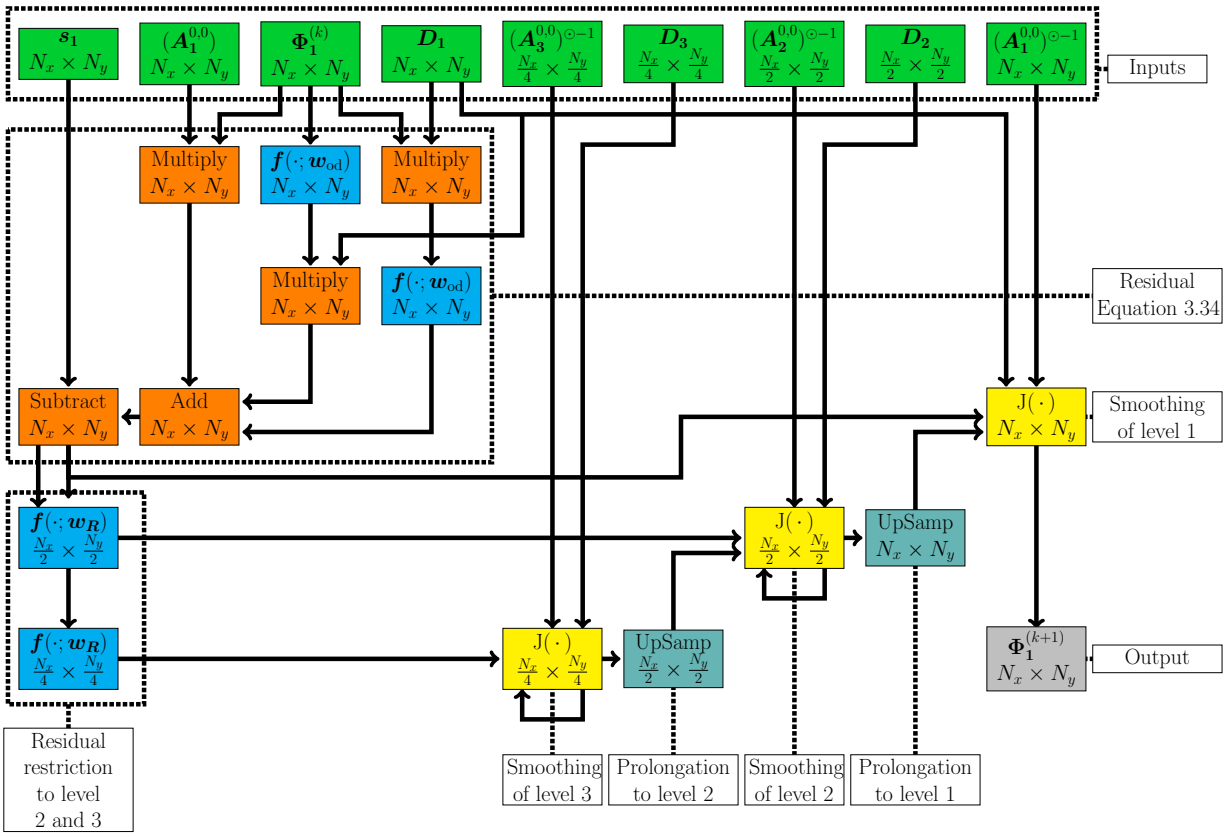


Figure 3.5: Multigrid network, MG(·), representing a single multigrid iteration. This network performs a single multigrid iteration on the flux of a single energy group. Takes the flux ($\mathbf{\Phi_1^{(k)}}$), representative source ($\boldsymbol{s_1}$), diffusion coefficients ($\boldsymbol{D_1}$) and the strictly diagonal coefficients ($\boldsymbol{A_1^{0,0}}$), along with the coarser resolution coefficients, as inputs (green boxes). A number of layer operations are performed, mathematical operations in orange, convolutional passes in cyan, sub-model operations in yellow and upsampling in teal. The sub-models can be iterated on multiple times. Finally it outputs the flux of the next multigrid iteration flux ($\mathbf{\Phi_1^{(k+1)}}$). Arrow origins show which layer the data originated and the end of the arrow shows which layer takes that data as input. Dimensions of layers are given on the second line of each box.

3.2.4 Multi-group network

The multigrid function and network, given by Equation (3.38) and Figure 3.5 respectively, show how a single multigrid iteration may be applied to a single energy group g. Multi-group problems must have balanced scattering terms, achieved through iterating until the terms balance. Equation (3.10) shows how a block Gauss-Seidel approach is used when constructing $s_{i,j,g}$. The scattering term, Σ^s, is constructed using the most recent flux information, achieved by resolving each energy group sequentially.

Figure 3.6 shows the network equivalent to how energy groups can be resolved as a block Gauss-Seidel approach. Green boxes represent inputs, Yellow boxes represent sub-networks and grey

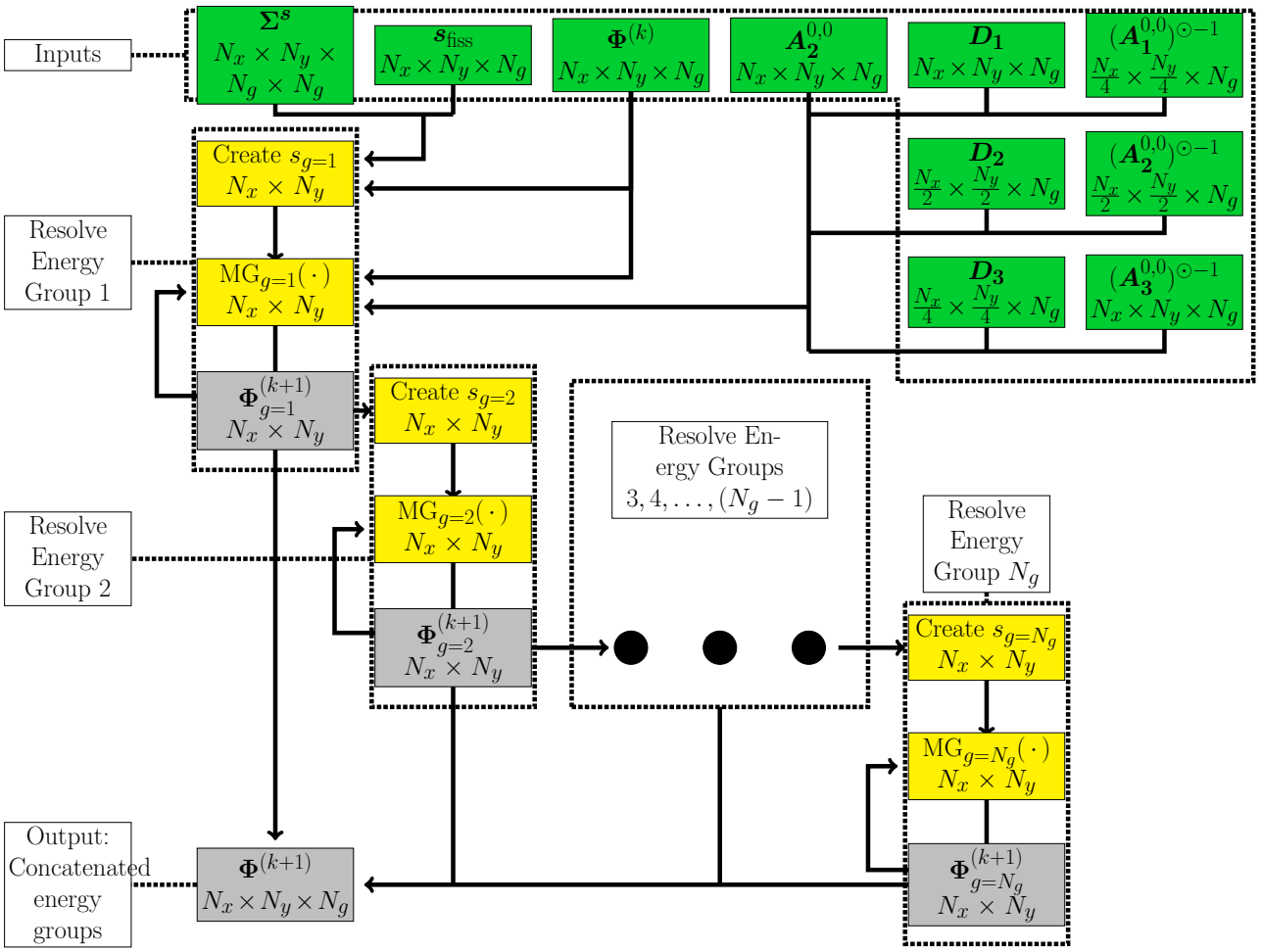


Figure 3.6: Multi-group network representing a single multi-group iteration. This network performs a single multi-group iteration on the flux of all energy groups. Takes the flux ($\boldsymbol{\Phi}_{\mathbf{1}}^{(k)}$), scattering cross-sections Σ_s, source fission term ($\boldsymbol{s}_{\text{fiss}}$), diffusion coefficients ($\boldsymbol{D_1}$) and the strictly diagonal coefficients ($\boldsymbol{A}_{\mathbf{1}}^{0,0}$), along with the coarser resolution coefficients, as inputs (green boxes). Each energy group is updated sequentially, first through updating the source term for a specific energy group and then performing a number of multigrid sub-model iterations. The updated flux for an energy group is then passed onto subsequent energy groups. Left out of the figure for clarity, the inputs (green boxes) are all connected to subsequent sub-models (yellow boxes). Finally, it outputs the flux of the next multi-group iteration flux ($\boldsymbol{\Phi}_{\mathbf{1}}^{(k+1)}$). Arrow origins show which layer the data originated and the end of the arrow shows which layer takes that data as input. Dimensions of layers are given on the second line of each box.

boxes represent outputs. For clarity, the green outputs are only shown as being linked to the first energy group but would be linked to all subsequent energy groups. $\boldsymbol{s}_g$ is a vector containing Σ^s, $\boldsymbol{s}_{\text{fiss}}$ and $\boldsymbol{\Phi}^{(k)}$ and is formed using Equation (3.10). $\boldsymbol{s}_g$ is then used in the MG sub-model to resolve for $\boldsymbol{\Phi}_g$, repeating for a number of multigrid iterations until:

$$\boldsymbol{\Phi}_g^{(k+1)} \approx \boldsymbol{\Phi}_g^{(k)}. \tag{3.40}$$

$\boldsymbol{\Phi}_g$ is then used in $\boldsymbol{s}_{g'}$ where $g' > g$. Once all energy groups have been resolved they can be concatenated to form $\boldsymbol{\Phi}^{(k+1)}$. This is repeated until:

$$\boldsymbol{\Phi}^{(k+1)} \approx \boldsymbol{\Phi}^{(k)}. \tag{3.41}$$

An alternative to the Gauss-Seidel approach is to use the Jacobi approach to resolve all energy groups simultaneously, which is achievable using the multigrid network alone. This is performed by passing all N_g energy groups to the MG network at the same time, only updating s outside of this. The source term in Equation (3.10) instead changes to:

$$s_{i,j,g} = \lambda \chi_g \sum_{g'=1}^{N_g} \nu_{g'} \Sigma^f_{i,j,g'} \phi^{(k)}_{i,j,g'} + \sum_{g'=1}^{N_g} \Sigma^s_{i,j,g' \to i,j,g} \phi^{(k)}_{i,j,g'} \tag{3.42}$$

.

Equation (3.7) is an eigenvalue problem so λ needs to be determined. An approximation, usually $\lambda = 1$, is used and the fission term is passed to the multi-group network where

$$\boldsymbol{s}_{\text{fiss},g} = \lambda \chi_g \sum_{g'=1}^{N_g} \nu_{g'} \boldsymbol{\Sigma}^{\boldsymbol{f}}_{g'} \boldsymbol{\Phi}^{(k)}_{g'}, \tag{3.43}$$

for each energy group g and $\boldsymbol{s}_{\text{fiss}}$ is an array containing all g of $\boldsymbol{s}_{\text{fiss},g}$. The power method [31] is the method chosen here to determine the dominant eigenvalue for this problem. The implementation of the power method used here is the same as [148] and functions outside of the multi-group network.

3.3 Results

All networks were implemented in python using Keras [147] with the TensorFlow backend [90]. The following section contains solutions produced by a single neural network acting as the multi-group solver, with pre-determined weights. The KAIST benchmark is used [149] with a simplified fuel rod geometry, the same fuel assembly geometry and a modified reactor core geometry.

3.3.1 Fuel Assembly

The multi-group iteration network is now used to produce solutions to a 2D fuel assembly. This fuel assembly is based on the KAIST benchmark [149] and uses their cross-sections.

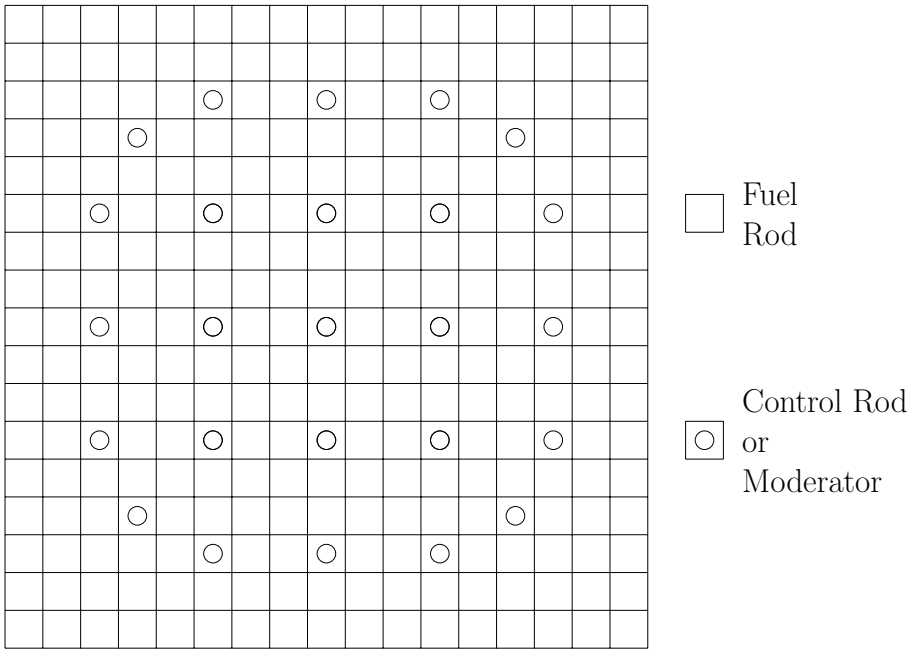


Figure 3.7: Geometry of UOX fuel assembly.

Figure 3.7 contains the geometry for the UOX fuel assembly. This consists of a 17×17 lattice containing either UOX fuels rods, guide-tubes or control rods. Depending on the simulation, each square contains either 20×20 cells/nodes or 10×10 cells. For the 20×20 cells this forms a total of $115,600$ cells and 1364 cells along boundaries to enforce the boundary conditions. The energy was discretised into seven groups, meaning the fuel assembly has $818,720$ degrees of freedom. Each side of the fuel assembly is of length 21.42 cm meaning each cell is 0.062 cm $\times$ 0.062 cm. Each side of the fuel assembly also has vacuum boundary conditions applied to it. The material parameters for the nodes for the two cell configurations are given in Figure 3.8.

For solutions of the fuel assembly, two configurations are used to produce the solutions. Figure 3.7 contains spaces that can either be guide-tubes or control rods. In the first configuration, all of these spaces are guide-tubes, representing a system where the control rods are fully withdrawn. In the second configuration, all of these spaces are control rods, representing a system where the control rods are fully inserted. For all solutions in this section, 2 Jacobi iterations, 100 multigrid iterations and 100 multi-group iterations were performed.

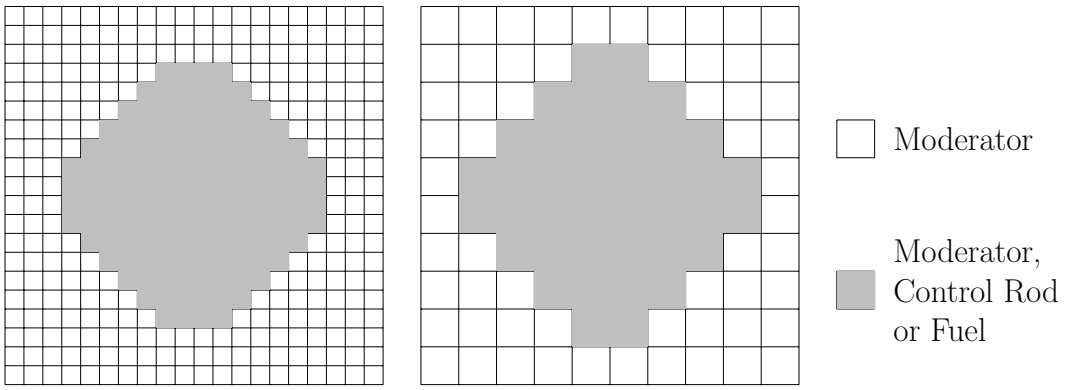


Figure 3.8: Geometry of pins for 20×20 or 10×10 cells.

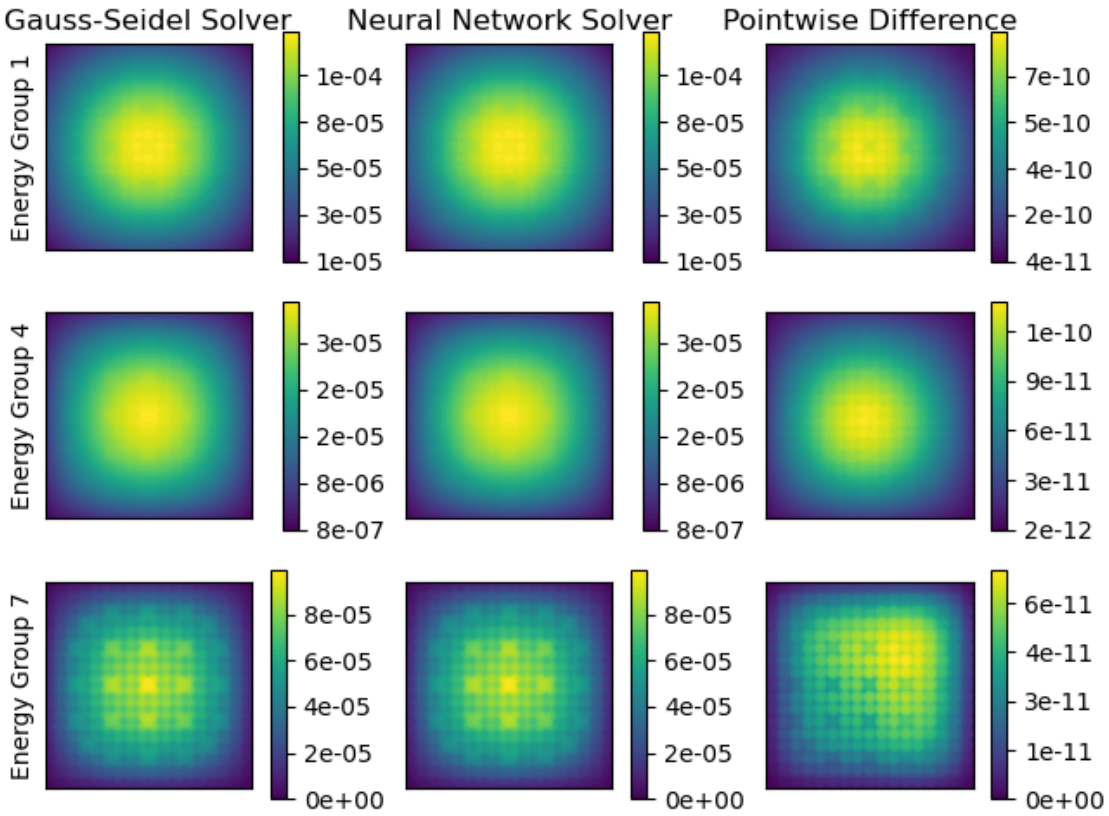


Figure 3.9: Scalar flux (neutrons $\mathrm{cm}^{-2}\,\mathrm{s}^{-1}$) across the fuel assembly for three energy groups for a fuel assembly with control rods fully withdrawn, generated using the multi-group network with linear filters for diffusion.

Figure 3.9 contains the flux profiles for three energy groups for a fuel assembly with no control rods inserted. The positions of the guide-tubes can be observed in between the fuel rods. It can be observed that the pointwise difference between the Gauss-Seidel solver and the neural network solver here is very low.

Figure 3.10 contains the flux profiles for three energy groups for a fuel assembly with no control rods inserted. The positions of the control rods can be observed in between the fuel rods,

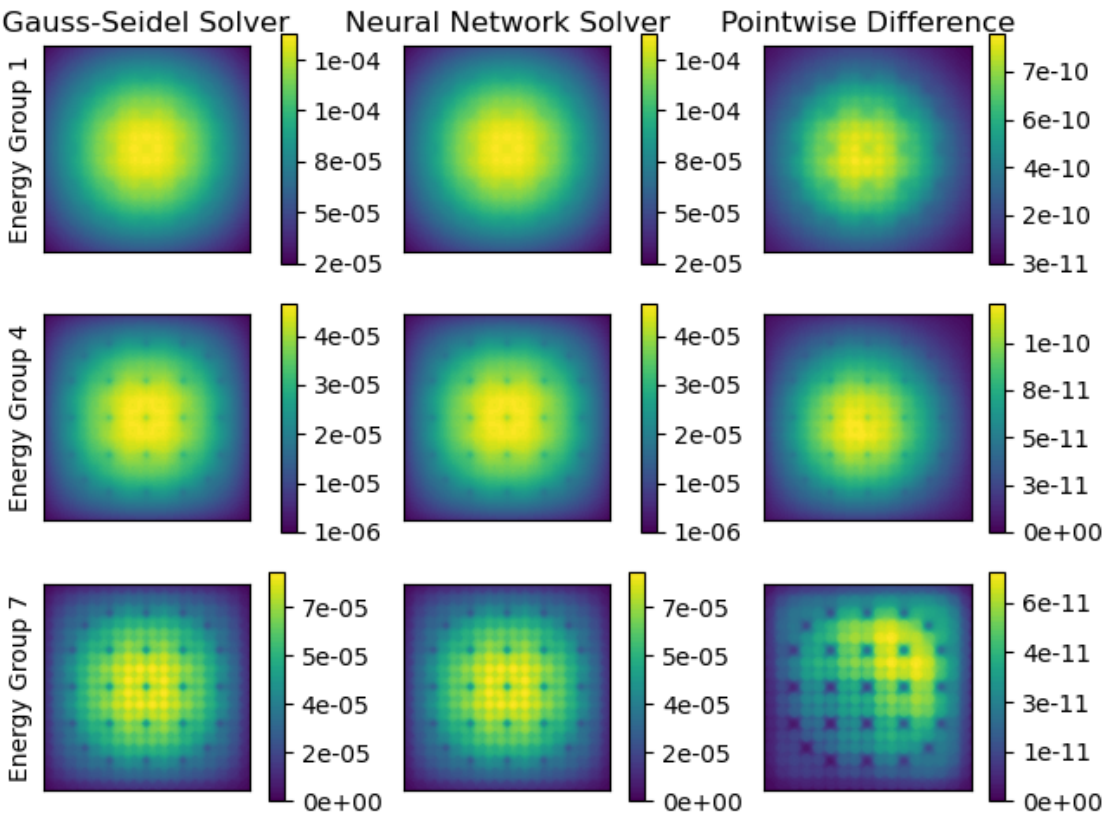
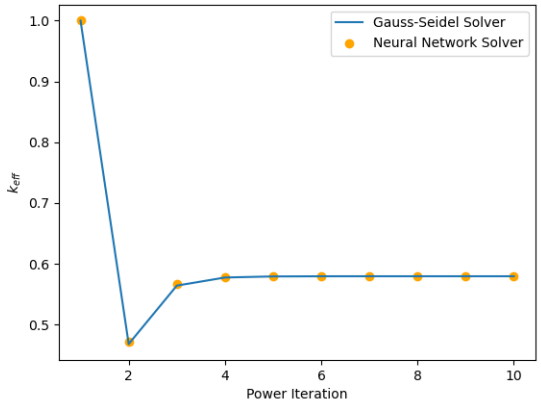


Figure 3.10: Scalar flux (neutrons $\mathrm{cm}^{-2}\,\mathrm{s}^{-1}$) across the fuel assembly for three energy groups for a fuel assembly with control rods fully inserted, generated using the multi-group network with linear filters for diffusion.

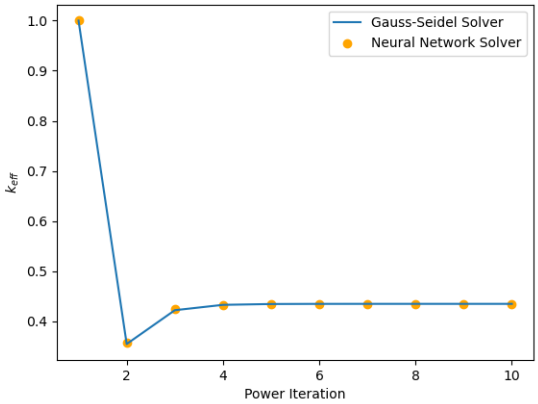
where the flux sharply decreases. It can be observed that the pointwise difference between the Gauss-Seidel solver and the neural network solver here is very low. Figure 3.11 contains the k_{eff} convergences for a fuel assembly with control rods fully withdrawn and inserted. It can be observed that k_{eff} is lower when control rods are inserted, as expected. The convergence for the Gauss-Seidel solver and the neural network solver here are identical, as expected when the solvers converge to the same solution.

3.3.2 Fuel Assembly - ConvFEM Filters

Figures 3.12 and 3.13 contain the scalar flux solution for a fuel assembly with control rods fully withdrawn and fully inserted respectively. Both solutions were generated using the Quadratic ConvFEM filter for the discretised Laplacian as defined by Equation (3.30).

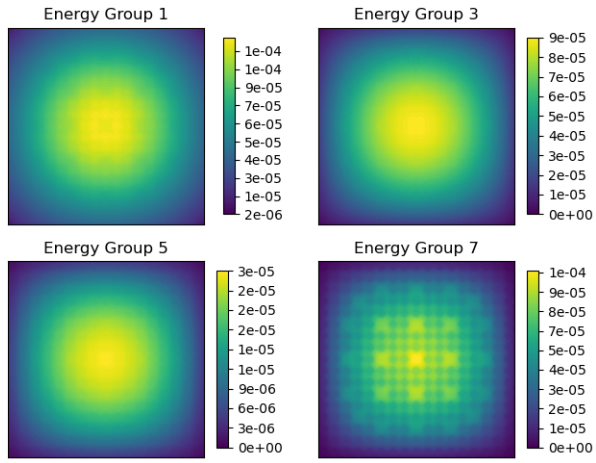


(a) k_{eff} vs power iteration for fuel assembly with control rods fully withdrawn, converging to 0.5796 for the Gauss-Seidel solver and 0.5798 for the Neural Network Solver.

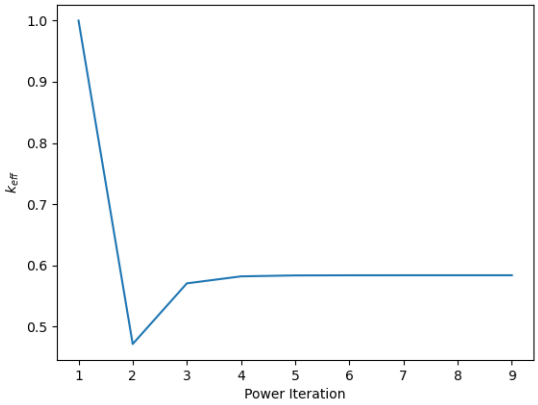


(b) k_{eff} vs power iteration for fuel assembly with control rods fully inserted, converging to 0.4346 for the Gauss-Seidel solver and 0.4347 for the Neural Network Solver.

Figure 3.11: Fuel Assembly k_{eff} vs power iteration using Multi-group network, generated using the multi-group network with linear filters for diffusion.



(a) Scalar flux (neutrons $\text{cm}^{-2}\,\text{s}^{-1}$) across the fuel assembly for four energy groups for all control rods withdrawn.

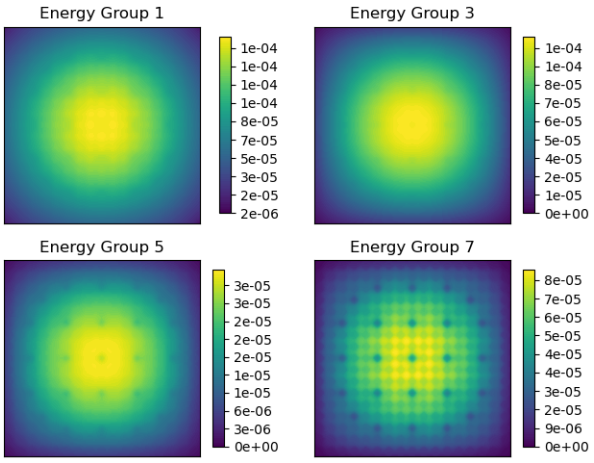


(b) k_{eff} vs power iteration for a fuel assembly solution for all control rods withdrawn, converging to 0.5890 compared to the 0.5798 that the linear filter generated.

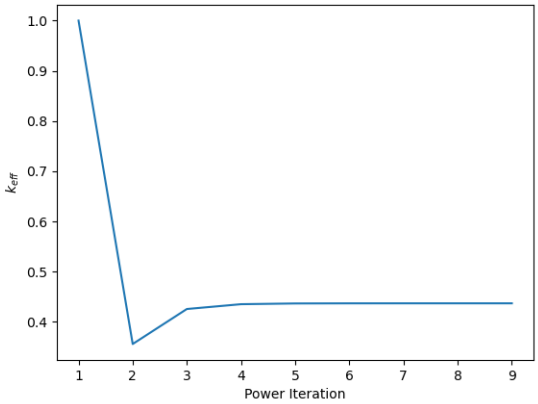
Figure 3.12: Fuel assembly flux and k_{eff} convergence for a fuel assembly solution with control rods fully withdrawn, generated using the multi-group network with Quadratic ConvFEM filters for diffusion.

3.3.3 Fuel Assembly - Time comparisons

Table 3.1 shows the time comparisons for 100 Jacobi iterations performed for the fuel assembly test case. The 100 iterations were performed 400 times and we calculate the minimum, maximum and average of these times. The neural network implementation used the multi-group



(a) Scalar flux (neutrons $\mathrm{cm}^{-2}\,\mathrm{s}^{-1}$) across the fuel assembly for four energy groups for all control rods inserted.



(b) k_{eff} vs power iteration for a fuel assembly solution for all control rods inserted, converging to 0.4409 compared to the 0.4347 that the linear filter generated.

Figure 3.13: Fuel assembly flux and k_{eff} convergence for a fuel assembly solution with control rods fully inserted, generated using the multi-group network with Quadratic ConvFEM filters for diffusion.

network (see Figure 3.6), but replaced the multigrid network (see Figure 3.5) with the Jacobi network (see Figure 3.2) so that it is directly comparable with the Fortran implementation. The neural network implementation was run on a single AMD EPYC 7742 CPU[1] and a single 24 GB NVIDIA RTX 6000 GPU. The serial Fortran implementation was run on a single CPU of the same type as above. From Table 3.1, it can be observed that the neural network implementation on the GPU is the fastest, resulting in an average time that is approximately 4 times faster than the Fortran implementation on the CPU, which is 5 times faster than the neural network implementation run on a single CPU. Part of the reason as to why the neural network implementation on the CPU is slowest, is that the convolutional filter requires 9 operations to be performed when only 4 are required for the finite volume discretisation (and therefore only 4 operations are performed in the Fortran code, see Equation (3.27)). Iterations run on the CPU showed a higher variability in their timings than on the GPU, which can be seen in the standard deviations shown in Table 3.1.

[1]In order to run on a single CPU, the number of CPUs requested had to be set, both in the instructions to the HPC and in the TensorFlow code.

implementation	hardware	computational times (s)			
		min	max	average	standard deviation
neural network	AMD EPYC 7742 CPU	21.2420	23.0204	22.3475	0.5084
neural network	NVIDIA RTX 6000 GPU	1.1720	1.3999	1.1847	0.0309
Fortran	AMD EPYC 7742 CPU	4.2615	4.8998	4.5892	0.2748

Table 3.1: Time comparisons for 100 Jacobi iterations performed on the fuel assembly test case using a CPU and a GPU for the neural network solver and a CPU for the serial Fortran code. The 100 iterations were performed 400 times so the minimum, maximum and average times, and standard deviations are shown.

3.3.4 Reactor Core

The multi-group network is now used to produce solutions for a reactor core. This reactor core is a simplified version of the KAIST benchmark [149] and uses the same cross-sections, with the reflector material being replaced with the moderator material. Figure 3.14 contains the

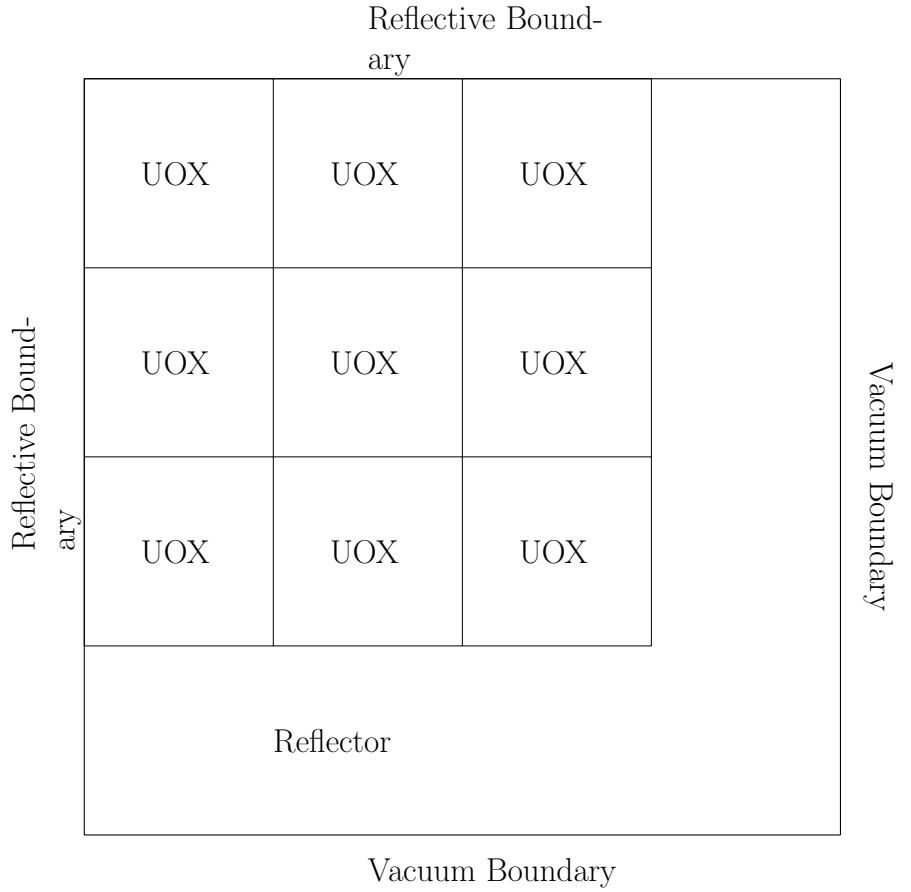


Figure 3.14: Geometry of Reactor Core for a simplified version of the KAIST benchmark [149].

geometry of a quarter of the reactor core. This contains a 3×3 grid of fuel assemblies. Each of these are a UOX fuel assembly, with the same information shown in Section 3.3.1 but with

10×10 nodes/cell in each square containing a pin. The upper and left sides have reflective boundary conditions applied and the right and bottom sides have vacuum boundary conditions applied, meaning the whole system represents the bottom right quarter of a reactor. The width of the reflector is also 21.42 cm so each side of the domain (shown in Figure 3.14 is 85.68 cm. The geometry or grid is uniform in the reflector or whole domain, meaning the number of cells along the width of the reflector is the same as the number of cells along the width of a fuel assembly. The reflector, therefore, contains $202,300$ cells/nodes, all nine fuel assemblies contain a total of $260,100$ cells and a total of 2724 cells are used for boundary conditions. The energy was again discretised into seven groups resulting in $3,236,800$ degrees of freedom. For all solutions in this section, 5 Jacobi iterations, 100 multigrid iterations and 100 multi-group iterations were performed.

Inserted	Withdrawn	Withdrawn
Withdrawn	Inserted	Withdrawn
Withdrawn	Inserted	Inserted

(a) Reactor configuration one.

Withdrawn	Withdrawn	Withdrawn
Withdrawn	Inserted	Withdrawn
Inserted	Inserted	Withdrawn

(b) Reactor configuration two.

Figure 3.15: Reactor core configurations where withdrawn means control rods are fully withdrawn and inserted means control rods are fully inserted.

For the configurations of the reactor core, each fuel assembly can either have control rods fully inserted or fully withdrawn. Figure 3.15 shows the two reactor configurations presented here. Each fuel assembly was randomly selected to have control rods fully inserted or fully withdrawn. Configuration one has five fuel assemblies with fully withdrawn control rods and four fuel assemblies with control rods fully inserted. Configuration two has six fuel assemblies with fully withdrawn control rods and three fuel assemblies with fully inserted control rods.

Figure 3.16 contains the flux profiles for four energy groups for reactor configuration one. It can be observed that flux is higher in regions where control rods are fully withdrawn, with a notable drop in the upper left corner where they are inserted.

Figure 3.17 contains the flux profiles for four energy groups for reactor configuration two. Again,

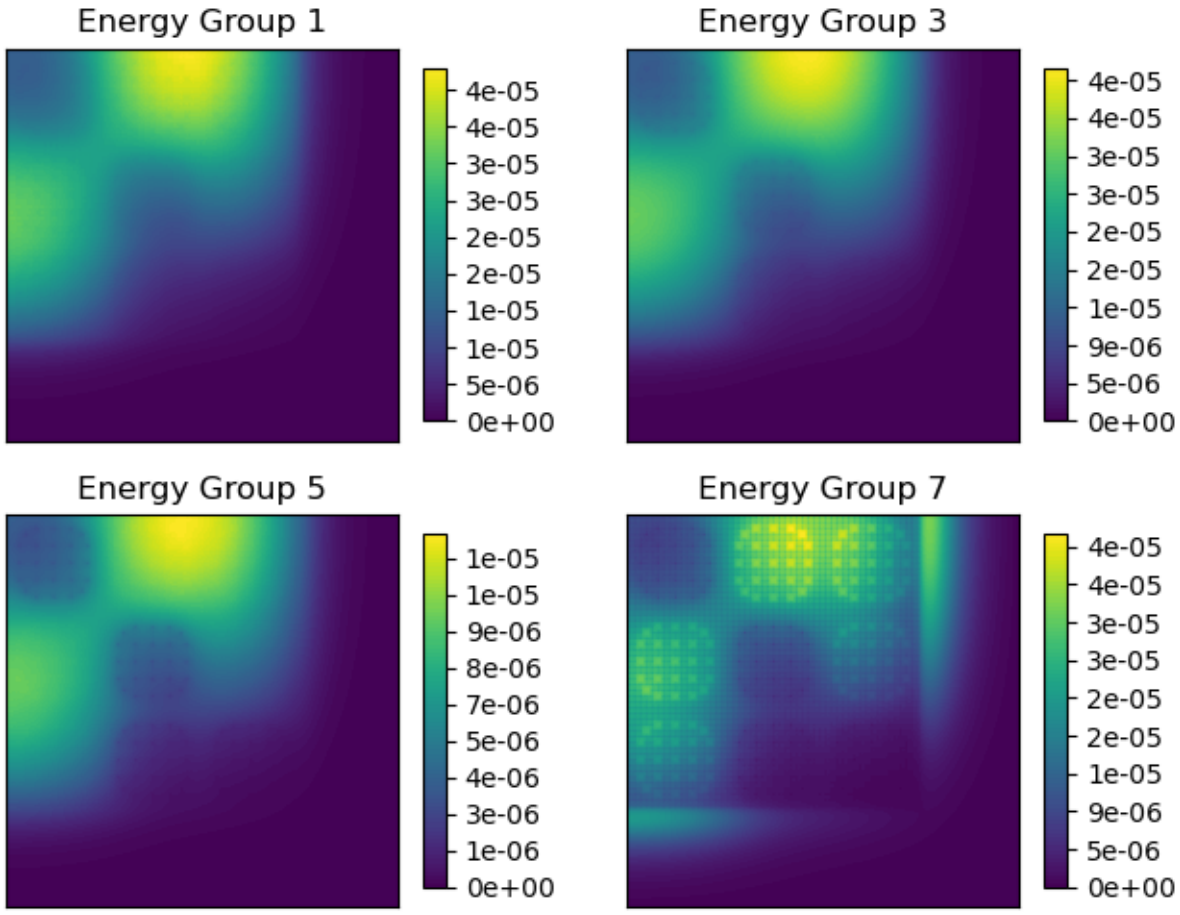


Figure 3.16: Scalar flux (neutrons $\mathrm{cm}^{-2}\,\mathrm{s}^{-1}$) across the fuel assembly for four energy groups for reactor configuration one, generated using the multi-group network with linear filters for diffusion.

the flux shows a drop where control rods are inserted, with the highest amount occurring in the upper left corner by the reflective boundaries.

Figure 3.18 shows the k_{eff} convergence for both reactor configurations. It can be observed that configuration two has a slightly higher k_{eff} than configuration two, which is expected based on the configuration.

3.3.5 Reactor Core - Time comparisons

Table 3.2 shows the time comparisons for 100 Jacobi iterations performed on the reactor core test case. The neural network implementation was tested on the CPU and GPU (details of hardware given in see Section 3.3.3 and in Table 3.2), and the serial Fortran implementation was tested on the CPU. The 100 Jacobi iterations were performed 100 times. The reactor core test case has about 4 times the number of degrees of freedom as for the fuel assembly test

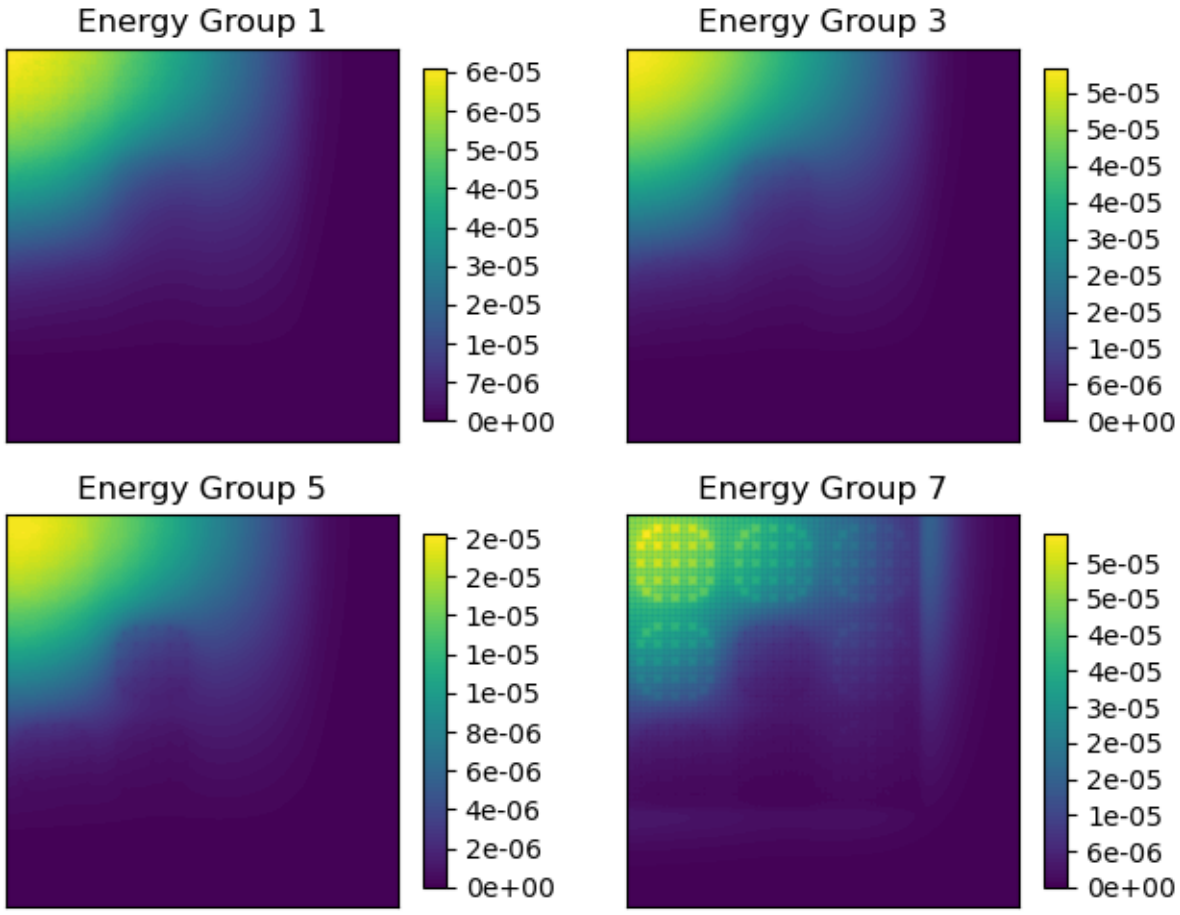
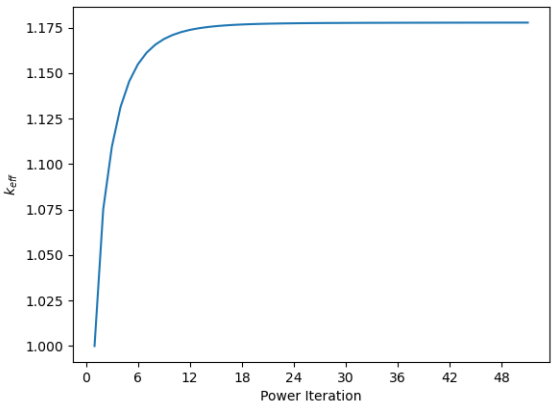
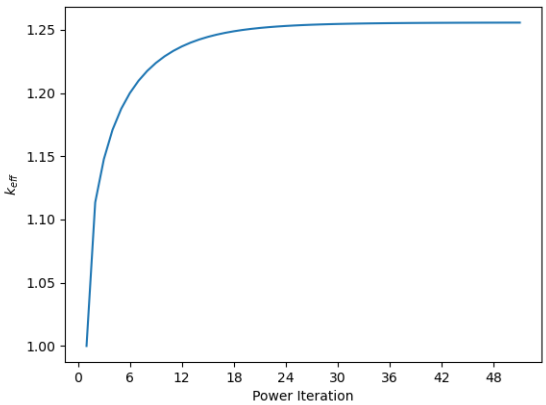


Figure 3.17: Scalar flux (neutrons $\mathrm{cm}^{-2}\,\mathrm{s}^{-1}$) across the fuel assembly for four energy groups for reactor configuration two, generated using the multi-group network with linear filters for diffusion.



(a) k_{eff} vs power iteration for reactor configuration one, converging to 1.1777.



(b) k_{eff} vs power iteration for reactor configuration two, converging to 1.2557.

Figure 3.18: Reactor core k_{eff} vs power iteration using the multi-group network for both configurations and linear filters for diffusion.

case, which is reflected in the timings being 4 times larger for the reactor core than for the assembly. Again, it can be observed that the neural network implementation on the GPU is the fastest, being about 4 times faster than the Fortran implementation on the CPU, which is 5 times faster than the neural network implementation run on a single CPU. The standard deviations for the implementations run on the CPU are higher than those for the GPU, as was also observed for the assembly test case. Although, there is actually very little variation in the multiple realisations of the Fortran code (run on the CPU) and the neural network run on the GPU.

		computational times (s)			
implementation	hardware	min	max	average	standard deviation
neural network	AMD EPYC 7742 CPU	88.8473	100.6071	94.5058	5.0606
neural network	NVIDIA RTX 6000 GPU	4.7403	4.8235	4.7901	0.0167
Fortran	AMD EPYC 7742 CPU	16.5012	16.8404	16.6222	0.0897

Table 3.2: Time comparisons for 100 Jacobi iterations performed on the reactor core test case using a CPU and a GPU for the neural network solver and a CPU for the serial Fortran code. The 100 iterations were performed 100 times so the minimum, maximum average times and standard deviations are shown.

3.4 Summary

This chapter presents a new approach that uses the tools within Artificial Intelligence (AI) software to replicate the processes of solving partial differential equations that are discretised through standard numerical method schemes. Two test cases are used to demonstrate the approach, a fuel assembly and a reactor core. The fuel assembly solution is compared with a standard Gauss-Seidel solver.

The approach here defines the weights of convolutional layers to reproduce the operations of iterative solvers, forming a neural network that acts as a Jacobi iteration. This concept is then extended to a standard sawtooth multigrid solver, forming a neural network that acts as a multigrid iteration, incorporating the Jacobi neural network internally. This multigrid network is used with another network that acts as a multi-group solver. The benefit of using such an approach is that it allows one to exploit the power of AI software (e.g. PyTorch and TensorFlow) and their built-in AI technologies. For example, their executions are already optimised for different computer architectures, whether it be CPUs, GPUs, HPC or next-

generation AI chips.

The fuel assembly test case demonstrates that the approach produces the same solution (within the tolerance of the solver) as that obtained using a standard Gauss-Seidel approach, producing the same k_{eff} convergence. This test case is also used to demonstrate how higher-order filters may be substituted into the convolutional layers for a more accurate stencil for the diffusion operator. The approach is also extended to a more computationally demanding problem, a reactor core. In both test cases, the neural network solver on a GPU was quicker than the Fortran code on a CPU, resulting in one-quarter of the time. This provides a significant speed up and the ability to utilise the GPU to perform parallel operations without requiring knowledge of how to write code to do so. This method could be utilised as a replacement for traditional solvers, allowing methods that utilise neural networks, such as data assimilation and uncertainty quantification, to be performed with ease.

Future work would also involve putting the power eigenvalue iteration into the neural network. This would enable the neural network to calculate sensitivities of the eigenvalue to material properties, say automatically using the backpropagation algorithm of the neural network. It is extended to transport theory in chapter 4. An important next step would be to optimise the code and methods further (e.g. taking into account the multigrid bottleneck) so that large problems can be run on GPUs or new AI computers.

Chapter 4

Neutron Transport Neural Network Solver

The content within this chapter is based on the unpublished manuscript:

T. R. F. Phillips, C. E. Heaney, B. Chen, A.G. Buchan, and C. C. Pain. Solving the discretised Boltzmann transport equations using neural networks: Applications in neutron transport. [9]

Chapter 2 contains methods for how the high-fidelity snapshots of this chapter were generated. Chapter 3 is a precursor to this chapter.

4.1 Introduction

The previous chapter 3 implemented a neural network solver, using AI libraries, to solve the multi-group diffusion equation for nuclear reactor applications and eigenvalue problems. This method is extended from the 2D space problem to a 4D space-angle problem in this chapter, to produce a framework that can be used to solve the full Boltzmann transport problem. A structured grid is placed across the space and a series of coarsened grids is produced (as is usual practice, see [150, 151]) by coarsening the grid by a factor of two in each of the 4 dimensions using simple prolongation and restriction operators.

To converge the solution the sawtooth multigrid cycle is used here, partly because, it often has superior performance to the classical V-cycle and also because it straightforwardly maps

onto the architecture of the U-net neural network, see [142, 152] for U-net applications and architecture. Others have used recurrent U-net architectures[153], like the recurrent network presented here. A single sawtooth cycle, see [154], (also a single forward pass through the U-net) requires the formation of the discrete equation residuals on the finest grid and the restriction of this residual onto the series of coarsened grids. Once the coarsest grid level is reached (often one cell/variable) then a Jacobi relaxation can be applied to the coarsened equations with the residual from the finest grid as a source. The result of this is then prolonged onto the next finest grid and the process is repeated right up to the finest grid level, completing one cycle of the multigrid method. This cycle, or pass through the U-net, is repeated to converge the multigrid solution. More elaborate and better converging multigrid cycles exist such as W- and F- cycles or extensions of the basic sawtooth cycle, see [155]. However, when running on a GPU, say, with extreme parallelisation one may notice that performing a Jacobi relaxation of the coarsest grid can be just as expensive as on some of the finer grids because there is very limited parallelisation that can be extracted on the coarsest grids. Thus a compromise is necessary between spending more time on the coarser grids (such as with the W-cycle) and less such as with the V or saw-tooth cycles and should take into account the architecture that one wants to run the model on.

In addition, in the multigrid method presented here, on the finest grid, the residual is passed to an upwind discretisation method on the same grid and as well as on all coarsened grids. This method is used because it is simple, fast and helps stabilise the multigrid solution. This is partly done because of the difficulties in solving non-symmetric positive definite matrices using multigrid methods, which is the case when solving the Boltzmann transport equation in first-order form. However, despite this, major progress has been made on the use of multigrid methods for these equations, see [156, 157]. This is important as the most commonly used approach sweeps, in a serial fashion, through the grid or mesh updating the radiation field and is thus not well suited to run on the highly parallel GPUs or AI computers.

One of the most commonly used discretisation methods for the Boltzmann transport equation solution is the Discontinuous Galerkin (DG) method, see original applications to neutron transport in [158] and more recently for fluids [159]. This is often applied in linear FEM and lumped mass form to Boltzmann transport as used in the Attila code, see [160, 161]. However, implementing DG using this approach requires a number of neural network channels - one for

each node of the finite element (4 for linear quadrilateral elements) - and thus a more complex neural network is needed. This increase in complexity of DG further increases for higher-order elements as even more channels are required. The complexity increase associated with DG is also shared with continuous elements. For example, for quadratic continuous elements, one can see that the stencil is different for the centre and corner and mid-side nodes for quadrilateral elements in 2D and similarly for 3D elements. Thus new FEM is developed here called the Convolutional FEM or ConvFEM, for short, which has the same stencil everywhere and thus can result in much simpler neural network implementations. For the linear elements, the ConvFEM approach is identical to the classical bi-linear FEM approach. Other schemes such as that used in the [162–164] that can use high order polynomials and upwind biases for stability could also be used. This upwind bias means that one should ideally use non-centred filters to implement them as efficiently as possible.

However, the solution of the Bubnov-Galerkin discretisations of the Boltzmann transport equations is infamously unstable, for first-order transport equations, which is also the case for similar diamond differencing schemes. Our solution to this is to add a non-linear Petrov-Galerkin term [165–167], which is simply expressed as a diffusion term that we add to the overall solution method. This not only increases the diagonal dominance of the method but also suppresses Gibbs oscillations, see [163] for control volume oscillation suppressing schemes, producing more realistic-looking results that are less prone to produce negative angular fluxes (unphysical solutions). Other work on the use of Petrov-Galerkin dissipation for the Boltzmann transport equations includes [168, 169]. Here, we develop a slight departure from the original, see [165], and the above non-linear Petrov-Galerkin methods. This results in anisotropic dissipation.

The chapter is organised as follows: Section 4.2 contains the methodology, section 4.3 contains an overview of the results and finally, a summary of the chapter is given in section 4.4.

4.2 Methodology

The first part of this section outlines the governing equations and their discretisation. The neural network solver is then explained, along with the 4D multigrid method and how it can be implemented as a neural network.

4.2.1 Boltzmann Transport Equation

The simplified multi-group steady-state Boltzmann equation for neutron transport can be written as:

$$\Omega \cdot \nabla \psi_g + \Sigma_g^a \psi_g + \sum_{\substack{g'=1 \\ g' \neq g}}^{N_g} \Sigma_{g \to g'}^s \phi_g = \lambda \, \chi_g \sum_{g'=1}^{N_g} \nu_{g'} \Sigma_{g'}^f \phi_{g'} + \sum_{\substack{g'=1 \\ g' \neq g}}^{N_g} \Sigma_{g' \to g}^s \phi_{g'}, \quad \forall g \in \{1, 2, \ldots, N_g\}, \tag{4.1}$$

where ψ_g and ϕ_g are the angular and scalar fluxes of the neutron population, $\Omega = (\mu, \nu)^T$ is the direction of particle travel, Σ_g^a represents the absorption cross-section, Σ_g^f represents the fission cross-section, ν_g is the average number of neutrons produced per fission event, Σ_g^s represents the scatter cross-section, χ_g is the proportion of neutrons produced for each energy group per fission event, N_g is the number of energy groups used and the subscript g denotes the energy group. The diffusion coefficient, $\Omega = (\mu, \nu)^T$, is the direction of neutron transport. The eigenvalue, λ, is defined as the reciprocal of k_{eff}, that is $\lambda = \frac{1}{k_{\text{eff}}}$, where:

$$k_{\text{eff}} = \frac{\text{number of neutrons in one generation}}{\text{number of neutrons in the preceding generation}}. \tag{4.2}$$

The boundary condition for reflection is:

$$\frac{\partial \psi_g}{\partial n} = 0\,, \tag{4.3}$$

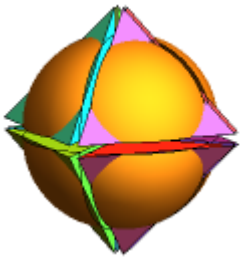
and for a vacuum or bare surface boundary conditions for incoming, to the domain, directions $(\mu_n, \nu_n)^T$:

$$\psi_g(\mu_n, \nu_n) = \psi_{n,g} = 0. \tag{4.4}$$

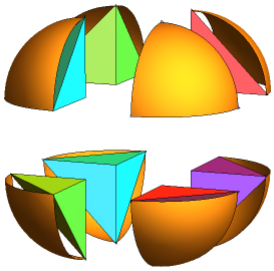
4.2.2 Discrete Ordinates

Here, the set of discrete ordinates that can be used to discretise in angle of particle travel are defined. The angle is discretised by placing an octahedron on a unit sphere as can be seen in Figure 4.1. A rectangular grid is then placed across each face of the octahedron, with a collapsed edge/node, seen in Figure 4.1c. Each octant is then placed together to form the final

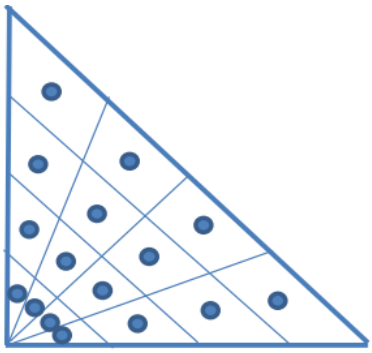
grid across the unit sphere, shown in Figure 4.1e.



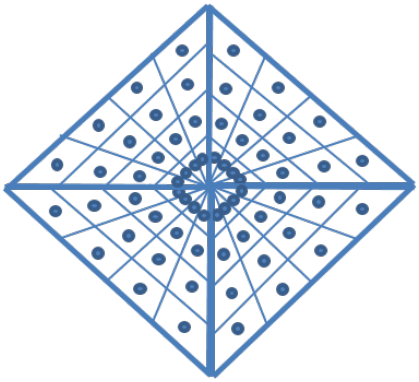
(a) Octahedron in a sphere, closed



(b) Octahedron in a sphere, open



(c) 4×4 grid on a triangle face.



(d) Top view of 4 octants placed over half unit sphere.

(e) Final grid across the unit sphere.

Figure 4.1: Diagrams showing how the unit sphere is split into 8 parts - forming an octahedron - and then how a regular $N_a \times N_a$ grid is placed over each of those parts or faces. Top shows the octahedron in a sphere, the left hand showing closed and right hand showing open [170] Bottom shows how a regular grid is placed over the octahedron. Diagrams showing the grid for a single face and the corresponding unit sphere.

Although a 4×4 grid is shown, if the grid is kept square (i.e. $N_a \times N_a$), and N_a is a power of 2, then this is a convenient way for the multigrid approach to be utilised through angle as well as space. Each cell in each face represents a discrete ordinate or SN direction. If $N_f = 8$ faces are used this results in $N_a \times N_a \times N_f$ discrete ordinate or SN directions.

Although any number of faces could be used, 8 faces is an ideal number as the directions are similar for every cell within each face as each face occupies an octant. If μ_n, ν_n and ξ_n represent the angles for a direction n from the x-axis, y-axis and z-axis respectively of the unit sphere. Each face, therefore, contains directions where all $N_a \times N_a$ directions have the same sign such

that:

$$\mu_n > 0 \; \forall \; n \quad \text{or} \quad \mu_n < 0 \; \forall \; n, \tag{4.5}$$

$$\nu_n > 0 \; \forall \; n \quad \text{or} \quad \nu_n < 0 \; \forall \; n, \tag{4.6}$$

$$\xi_n > 0 \; \forall \; n \quad \text{or} \quad \xi_n < 0 \; \forall \; n, \tag{4.7}$$

where each face contains a different combination of the three angles. Each direction (which is the mean direction over the patches shown after projecting onto the unit sphere) also has a corresponding weight or area of the patch associated with it (p_n). On the unit sphere, assuming coordinates (μ, ν, ξ), the mean directions are given, for patch n, by:

$$\mu_n = \frac{\int_{S_n} \mu dS}{\int_{S_n} dS}, \qquad \nu_n = \frac{\int_{S_n} \nu dS}{\int_{S_n} dS}, \qquad \xi_n = \frac{\int_{S_n} \xi dS}{\int_{S_n} dS}, \tag{4.8}$$

and $p_n = \int_{S_n} dS$ and in which S_n is the surface patch on the unity sphere, see Figure 4.1e. For a 2D system $\xi_n = 0$ for all N. Some of the inaccuracies in the surface area calculation are accounted for by normalising the weights p_n such that:

$$\sum_{n=1}^{N} p_n = 4\pi. \tag{4.9}$$

After integrating over each path of the unit sphere n and then dividing through by the area of the patch p_n, for a given direction n and energy group g, Equation (4.1) becomes :

$$\left[\mu_n \frac{\partial}{\partial x} + \nu_n \frac{\partial}{\partial y} + \tilde{\sigma}_{Tn,g}\right] \tilde{\psi}_{n,g} = \tilde{q}_{n,g}, \quad \forall n \in \{1, 2, ..., N\}, \quad \forall g \in \{1, 2, ..., N_g\}, \tag{4.10}$$

in which μ_n is the angle relative to the x-axis, ν_n is the angle relative to the y-axis, $\tilde{\sigma}_T n, g$ is the total cross section for direction n and energy group g, $\tilde{\psi}_{n,g}$ is the angular flux for direction n and energy group g and $\tilde{q}_{n,g}$ is the source of neutrons. The discrete ordinate directions $(\mu_n, \nu_n)^T$ is formed from the average direction associated with each patch n on the unit sphere. The angular flux (ψ) and emission of neutrons (q) are determined from the scalar flux (ϕ) and representative source term (s, Equation (4.1)) respectively:

$$\tilde{\psi}_{n,g} = p_n \tilde{\phi}_g / p_n = \tilde{\phi}_g \; , \quad \tilde{q}_{n,g} = p_n s_g / p_n = s_g, \tag{4.11}$$

and we have divided through by p_n because we have divided the entire equation by p_n and for an isotropic source for group g of s_g.

For a multi-group problem, assuming isotropic scattering, the total cross-section of neutrons $\tilde{\sigma}_{Tn,g}$ at a node represented by indexes i, j, and for direction n and energy group g, that is $\sigma_{Ti,j,n,g} = (\tilde{\sigma}_{Tn,g})_{i,j}$, is defined using:

$$\sigma_{Ti,j,n,g} = \Sigma^a{}_{i,j,g} \; + \sum_{\substack{g'=1 \\ g' \neq g}}^{N_g} \Sigma^s_{i,j,g \to i,j,g'}, \quad \forall n \in \{1, 2, ..., N_a\}. \tag{4.12}$$

After the angular flux in all N_d directions has been determined, the scalar flux can be calculated using:

$$\phi_g = \sum_{n=1}^{N_d} p_n \psi_{n,g}, \tag{4.13}$$

in which the quadrature weights are equal to the area p_n of each patch n on the unit sphere associated with direction n, see Figure 4.1. The code for the quadrature sets including the directions and weights (patch areas) can be found in the GitHub repository [171].

4.2.3 Discretisation in Cartesian space

An upwind control-volume discretisation of the Boltzmann transport equation in 2D with a regular mesh of $N_x \times N_y$ nodes can be written as:

$$\begin{aligned}
&\mathcal{H}(\mu_n)\mu_d \frac{(\psi_{i,j,n,g} - \psi_{i-1,j,n,g})}{\Delta x} + \mathcal{H}(-\mu_n)\mu_n \frac{(\psi_{i-1,j,n,g} - \psi_{di,j,n,g})}{\Delta x} \\
&+\mathcal{H}(\nu_n)\nu_n \frac{(\psi_{di,j,g} - \psi_{di,j-1,g})}{\Delta y} + \mathcal{H}(-\nu_n)\nu_n \frac{(\psi_{i,j-1,n,g} - \psi_{i,j,n,g})}{\Delta y} \\
&+\Sigma^a{}_{i,j,n,g}\phi_{i,j,n,g} \; + \sum_{\substack{g'=1 \\ g' \neq g}}^{N_g} \Sigma^s_{i,j,n,g \to i,j,g'}\phi_{i,j,n,g} = \sum_{g'=1}^{N_g} \Sigma^s_{i,j,g' \to i,j,n,g}\phi_{i,j,g'} + \lambda\chi_g \sum_{g'=1}^{N_g} \nu_{fis_{g'}} \Sigma^f_{i,j,g'}\phi_{i,j,n,g} \\
&\qquad \forall i \in \{l, 3, .., N_x - l\}, \qquad \forall j \in \{l, 3, .., N_y - l\}, \qquad \forall g \in \{1, 2, .., N_g\}, \qquad \forall n \in \{1, 2, .., N_d\},
\end{aligned} \tag{4.14}$$

in which subscript n represents direction, $\mathcal{H}(\nu_n) = 1$ if $\nu_n > 0$ else $\mathcal{H}(\nu_n) = 0$, Δx and Δy are the uniform cell or node widths in the x and y directions respectively, N_x and N_y are the numbers of nodes in the x and y directions respectively, the subscripts i and j refer to the cells

in the x and y directions respectively, N_g is the number of energy groups, the subscript g refers to the energy group and $\phi_{i,j,n,g}$ represents the scalar flux in energy group g in cell or node i, j.

For the incoming zero angular flux directions, we set the halo values, around the domain, to zero (bare surface boundary condition, see Equation (4.4)). For outgoing flux (no boundary condition needed) we fill the halo nodes with the values of the solution just inside the domain and next to the boundary. This approach enables us to have the same stencil everywhere and results in a highly efficient implementation. For the upwind scheme, this adjustment to the halo values does not change the solution (because of the full upwind bias of the discretisation) but it does affect the solution from the Petrov-Galerkin method (below).

The Petrov-Galerkin method, see [172], applied to Equation (4.10) can be expressed, in filter form, as:

$$\boldsymbol{r}_{n,g} = \boldsymbol{f}(\boldsymbol{\Psi}_{n,g}; \mu_n \boldsymbol{w_x}) + \boldsymbol{f}(\boldsymbol{\Psi}_{n,g}; \nu_n \boldsymbol{w_y}) + \boldsymbol{f}^{\text{k-Diff}}(\boldsymbol{\Psi}_{n,g}; \boldsymbol{k}_{\boldsymbol{x}_{n,g}}, \boldsymbol{w_{Diffxx}}) + \boldsymbol{f}^{\text{k-Diff}}(\boldsymbol{\Psi}_{n,g}; \boldsymbol{k}_{\boldsymbol{y}_{n,g}}, \boldsymbol{w_{Diffyy}}) - \boldsymbol{f}(\boldsymbol{q}_{n,g}; \boldsymbol{w_{ml}}) = \boldsymbol{0}, \quad (4.15)$$

in which $\boldsymbol{\Psi}_{n,g}$ is a $N_x \times N_y$ matrix containing all $\psi_{i,j,n,g}$ components for angle n and group g, $\boldsymbol{f}$ represents the filter operations for advection (first two terms of equation (4.10)) and is a matrix of size $N_x \times N_y$, for example at node i, j its component values are:

$$\boldsymbol{f}(\boldsymbol{\Psi}_{n,g}; \mu_n \boldsymbol{w_x})|_{i,j} = \sum_{u=-l}^{l} \sum_{v=-l}^{l} \mu_n w_{xu,v} \Psi_{i,j,n,g}, \quad (4.16)$$

$$\forall n \in \{1, 2, ..., N_d\},\ \forall g \in \{1, 2, ..., N_g\},\ \forall i \in \{1 + l, 2, ..., N_x - l\},\ \forall j \in \{1 + l, 2, ..., N_y - l\}, \quad (4.17)$$

and $\boldsymbol{f}^{\text{k-Diff}}$, in equation (4.15), are the diffusion filter operations for the numerically stabilising diffusion used in the Petrov-Galerkin method - noticed that this is applied anisotropically in the x- and y-directions from the non-linear stabilisation within the Petrov-Galerkin method (see next section). $\boldsymbol{q}_{n,g}$ in Equation (4.15) contains the scatter and removal operator as well as the fission source from the eigenvalue problem r.h.s. of Equation (4.22). We often collect the scalars μ_n, ν_n, $\forall n$ into the vectors $\boldsymbol{\mu} = (\mu_1\ \mu_2\ ...\ \mu_{N_d})^T$, $\boldsymbol{\nu} = (\nu_1\ \nu_2\ ...\ \mu_{N_d})^T$. We use a lumped approximation for this term and thus $\boldsymbol{w_{ml}}$ is a 1×1 filter containing the mass $\Delta x \Delta y$ associated

with all the nodes. The other filters $\boldsymbol{w_x}$, $\boldsymbol{w_y}$, $\boldsymbol{w_{Diffxx}}$, $\boldsymbol{w_{Diffyy}}$ are matrices of size $(2l+1) \times (2l+1)$ and are the result of discretising the operators $\frac{\partial}{\partial x}$, $\frac{\partial}{\partial y}$, $\frac{\partial^2}{\partial x^2}$, $\frac{\partial^2}{\partial y^2}$ respectively, see Appendix for definitions for various ConvFEM discretisations of these operators. For computational speed, it can be more efficient to use $\boldsymbol{f}(\boldsymbol{\Psi}_{n,g}; \mu_n \boldsymbol{w_x} + \nu_n \boldsymbol{w_y})$ rather than $\boldsymbol{f}(\boldsymbol{\Psi}_{n,g}; \mu_n \boldsymbol{w_x}) + \boldsymbol{f}(\boldsymbol{\Psi}_{n,g}; \nu_n \boldsymbol{w_y})$ in equation (4.15). In the above $\boldsymbol{r}_{n,g}$ is the discrete equation residual and the source $\boldsymbol{q}_{n,g}$ effectively gathers all the terms without spatial gradients in Equation (4.14).

The diffusion calculation is formed for directions x- and y- using the identities:

$$-\frac{\partial}{\partial x}\left(k_{xn,g}\frac{\partial \tilde{\psi}_{n,g}}{\partial x}\right) = -\frac{1}{2}\left(\frac{\partial^2}{\partial x^2}(k_{xn,g}\tilde{\psi}_{n,g}) + k_{xn,g}\frac{\partial^2}{\partial x^2}\tilde{\psi}_{n,g} - \tilde{\psi}_{n,g}\frac{\partial^2}{\partial x^2}k_{xn,g}\right) \quad \textit{analytical form} \tag{4.18}$$

$$\begin{aligned} &\sim \boldsymbol{f}(\boldsymbol{k_x}_{n,g} \odot \boldsymbol{\Psi}_{n,g}; \boldsymbol{w_{Diffxx}}) + \boldsymbol{k_x}_{n,g} \odot \boldsymbol{f}(\boldsymbol{\Psi}_{n,g}; \boldsymbol{w_{Diffxx}}) \\ &\quad - \boldsymbol{\Psi}_{n,g} \odot \boldsymbol{f}(\boldsymbol{k_x}_{n,g}; \boldsymbol{w_{Diffxx}}) \\ &\quad = \boldsymbol{f}^{\text{k-Diff}}(\boldsymbol{\Psi}_{n,g}; \boldsymbol{k_x}_{n,g}, \boldsymbol{w_{Diffxx}}) \,, \qquad \textit{discretised form} \end{aligned} \tag{4.19}$$

and

$$-\frac{\partial}{\partial y}\left(k_{y_{n,g}}\frac{\partial \tilde{\psi}_{n,g}}{\partial y}\right) = -\frac{1}{2}\left(\frac{\partial^2}{\partial x^2}(k_{y_{n,g}}\tilde{\psi}_{n,g}) + k_{y_{n,g}}\frac{\partial^2}{\partial y^2}\tilde{\psi}_{n,g} - \tilde{\psi}_{n,g}\frac{\partial^2}{\partial y^2}k_{y_{n,g}}\right) \quad \textit{analytical form} \tag{4.20}$$

$$\begin{aligned} &\sim \boldsymbol{f}(\boldsymbol{k_y}_{n,g} \odot \boldsymbol{\Psi}_{n,g}; \boldsymbol{w_{Diffyy}}) + \boldsymbol{k_y}_{n,g} \odot \boldsymbol{f}(\boldsymbol{\Psi}_{n,g}; \boldsymbol{w_{Diffyy}}) \\ &\quad - \boldsymbol{\Psi}_{n,g} \odot \boldsymbol{f}(\boldsymbol{k_y}_{n,g}; \boldsymbol{w_{Diffyy}}) \\ &\quad = \boldsymbol{f}^{\text{k-Diff}}(\boldsymbol{\Psi}_{n,g}; \boldsymbol{k_y}_{n,g}, \boldsymbol{w_{Diffyy}}) \,, \qquad \textit{discretised form} \end{aligned} \tag{4.21}$$

where $\odot$ denotes the Hadamard product which performs entrywise multiplication, $\boldsymbol{k_x}_{n,g}$, $\boldsymbol{k_y}_{n,g}$ are $N_x \times N_y$ matrices containing all diffusion values in the x- and y-directions $k_{xi,j,n,g}$, $k_{y_{i,j,n,g}}$ and $\boldsymbol{f}^{\text{k-Diff}}$ represents the application of the convolutional layer with weights associated with the discretised Laplacians in the x-direction $\frac{\partial^2}{\partial x^2} \sim \boldsymbol{w_{Diffxx}}$ and y-direction $\frac{\partial^2}{\partial y^2} \sim \boldsymbol{w_{Diffyy}}$. The equations (4.19) and (4.21) act as a definition of $\boldsymbol{f}^{\text{k-Diff}}$ and show how it is used to form second order derivatives.

The discretised form of Equation (4.1) can therefore be written, in matrix from, as:

$$\boldsymbol{A}\boldsymbol{\phi} = \lambda \boldsymbol{B}\boldsymbol{\phi}. \tag{4.22}$$

in which $\boldsymbol{\phi}$ is the entire solution vector and thus the matrix $\boldsymbol{A}$ contains the absorption, diffusion and scattering out of energy groups from the left-hand side of Equation (4.14), matrix $\boldsymbol{B}$ represents the fission terms in the right-hand side of Equation (4.14) and the vector $\boldsymbol{\phi}$ contains the values of the scalar flux for each cell in every energy group. The matrices are of size $N_g(N_x - l)(N_y - l)$ by $N_g(N_x - l)(N_y - l)$ in which we have not counted the halo nodes. Although a 2D discretisation is given here, the methods could be applied in 1D or 3D.

4.2.4 Neural network filters

This section describes the neural network filter and shows how they can implemented as a convolutional layer of a neural network using pre-determined weights. All networks were implemented in python using Keras [147] with the TensorFlow backend [90]. A convolutional layer has a filter or kernel, which is a small grid (smaller than the input data and typically of dimension 3×3, 5×5 or 7×7) whose cells have values known as weights associated with them. The filter is applied to part of the input by multiplying the input value by the weight in the overlapping cells. The products are summed to produce the output. The filter is then applied to a neighbouring part of the input and another output is created. This process is repeated until the filter has passed over all the input data. The action of a 2D convolutional layer on a 2D input can be written as follows as shown in Equation (4.16). In compact form this is generally:

$$\boldsymbol{a}_{n,g}^{k+1} = \boldsymbol{f}(\boldsymbol{a}_{n,g}^{k}; \boldsymbol{w}). \tag{4.23}$$

The weights of the filter are represented by $w_{u,v}$ and are contained in the filter in matrix form $\boldsymbol{w}$ and, in this case, the size of the filter is $(2l + 1) \times (2l + 1)$. A filter acting on one piece of input data, with $l = 1$, can be seen in Figure 4.2.

For the upwind approximation of advection (see Equation (4.14)), the filter weights have dif-

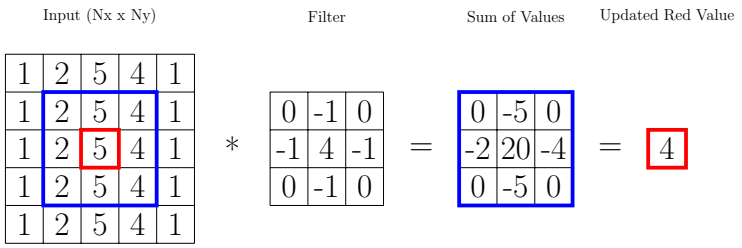


Figure 4.2: A convolutional filter which applies the discretised diffusion operator (a five-point finite-difference stencil) in 2D to 9 cells or grid points. The filter is first applied to all the cells in the blue block on the left; the result of which can be seen in the blue block after the equals sign. The 9 values are then summed to give the value in the red block on the right which is the output value.

ferent values, based on the the signs of μ_n and ν_n. For the x-direction:

$$\boldsymbol{w_x} = \begin{bmatrix} 0 & 0 & 0 \\ \frac{-1}{\Delta x} & \frac{1}{\Delta x} & 0 \\ 0 & 0 & 0 \end{bmatrix} \text{ if } \mu_n > 0, \tag{4.24}$$

or

$$\boldsymbol{w_x} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & \frac{-1}{\Delta x} & \frac{1}{\Delta x} \\ 0 & 0 & 0 \end{bmatrix} \text{ if } \mu_n < 0, \tag{4.25}$$

and for the y-direction:

$$\boldsymbol{w_y} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & \frac{1}{\Delta y} & 0 \\ 0 & \frac{-1}{\Delta y} & 0 \end{bmatrix} \text{ if } \nu_n > 0, \tag{4.26}$$

or

$$\boldsymbol{w_y} = \begin{bmatrix} 0 & \frac{1}{\Delta y} & 0 \\ 0 & \frac{-1}{\Delta y} & 0 \\ 0 & 0 & 0 \end{bmatrix} \text{ if } \nu_n < 0. \tag{4.27}$$

As previously established, Section 4.2.2, every direction in each face has the same sign of μ_n and ν_n. This means that each face requires one set of filters. A 2D filter is sufficient for a

2D problem with isotropic scattering. If $\boldsymbol{\Psi}_{\boldsymbol{x},n,g} = \boldsymbol{f}(\boldsymbol{\Psi}_{n,g}; \boldsymbol{w}_{\boldsymbol{x}})$ and $\boldsymbol{\Psi}_{\boldsymbol{y},n,g} = \boldsymbol{f}(\boldsymbol{\Psi}_{n,g}; \boldsymbol{w}_{\boldsymbol{y}})$ is the convolutional discretising the advection in the x- and y-directions, then there are N_f of these filters associated with the discrete ordinate faces shown in Figures 4.1. Each filter is then applied to all $N_a \times N_a$ directions for a single face. Using the Jacobi method the iteration for direction n and energy group g can be represented as:

$$\boldsymbol{\Psi}_{n,g}^{(k+1)} = \boldsymbol{d}_{n,g}^{\odot -1} \odot (\boldsymbol{d}_{n,g} \odot \boldsymbol{\Psi}_{n,g}^{(k)} - \boldsymbol{r}_{n,g}^{(k)}), \tag{4.28}$$

in which the vector $\boldsymbol{d}_{n,g}$ contains the diagonal of the matrix or filter upwind discretised system multiplied (see Equation (4.14)) by a factor β ($\beta = 3$ is used here) to introduce relaxation into the scheme on the finest grid when the Petrov-Galerkin discretisation is applied. The inverse Hadamard product [146], used in the above, is defined $\boldsymbol{d}_{n,g}^{\odot -1}|_{i,j} = \frac{1}{d_{i,j,n,g}}$. The superscript k refers to the iteration level and the residual $\boldsymbol{r}_{n,g}^{(k)}$ is formed from equation (4.15) with the most recent value of the angular flux $\boldsymbol{\Psi}_{n,g}^{(k)}$. When the upwind scheme is applied on the finest grid and on all coarsened grid levels $\beta = 1$ is used. Figure 4.3 details the architecture of the neural network that performs the Jacobi iteration given by Equation (4.28).

4.2.5 Space-Angle Multigrid

Here we define the 4D multigrid method used here. 4D is generated from 2D in Cartesian space and 2D in angle. The schematic shown in Figure 4.4 shows how it works and it has the following parts: (1) We assume in space and in angle (similar in multi-dimensions) that we have a different filter for each discrete ordinate direction but the same across space. (2) In angle we simply add the directions, or patches on the unit sphere, together to form coarser discrete ordinate equations. (3) In Cartesian space we discretise on the coarser grids to form the coarse grid equations on each multigrid level. It should be noted that the material properties are mapped to a coarser grid using a harmonic average before the discretisation on the coarser grids is formed. The same discretisation is used on each multigrid level but with different cell or node sizes Δx, Δy. The multigrid restriction and prolongation operations associated with this multi-grid method are shown in the schematic 4.5.

We show the multigrid cycle used to form the iterative space-angle multigrid solution method in Figure 4.6. This shows a single sawtooth cycle multigrid iteration with two restrictions.

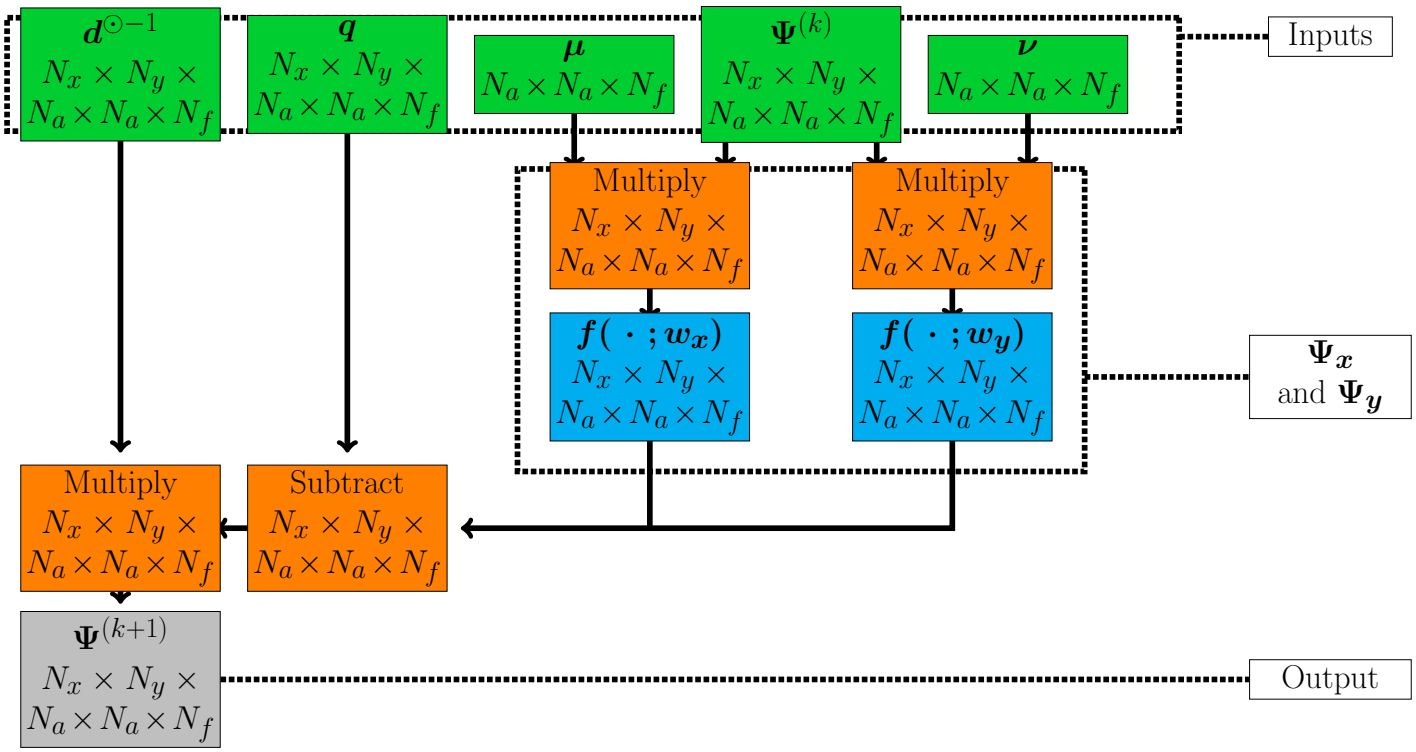


Figure 4.3: This is a schematic of the Neural network, J, representing a single Jacobi iteration. This network performs a single Jacobi iteration on the flux of a single energy group. Takes the angular flux ($\boldsymbol{\Psi}^{(k)}$), source ($\boldsymbol{q}$), angular velocities ($\boldsymbol{\mu}$,$\boldsymbol{\nu}$) and the strictly diagonal coefficients ($\boldsymbol{d}^{\odot -1}$) as inputs (green boxes). A number of layer operations are performed, mathematical operations in orange and convolutional passes in cyan. Finally, it outputs the flux of the next Jacobi iterations ($\boldsymbol{\Psi}^{(k+1)}$). Arrow origins show which layer the data originated and the end of the arrow shows which layer takes that data as input. Dimensions of layers are given on the second line of each box.

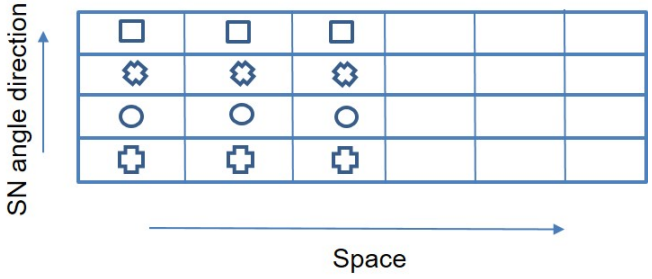


Figure 4.4: Space and Angle multigrid schematic showing how the multigrid method works. The different shapes show that the filters are different for each angle direction or ordinate (SN).

First, the residual from the Petrov-Galerkin method, Equation (4.15) is sent down through all the multigrid levels by restriction. Then the upwind scheme is used with the Jacobi iterations on each of the grids starting from the coarsest space-time grid and working up to the finest grid. On the finest grid level, a Jacobi iteration is applied using the upwind scheme and then the Jacobi iteration is applied to the Petrov-Galerkin method but with an enhanced diagonal

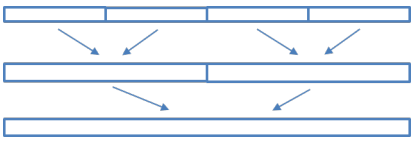


Figure 4.5: Restriction and prolongation (reverse arrow operation) operations in 1D. In 4D instead of restricting two variables/nodes or cells, 2^4 variables are restricted in the 4D space and angle multigrid method.

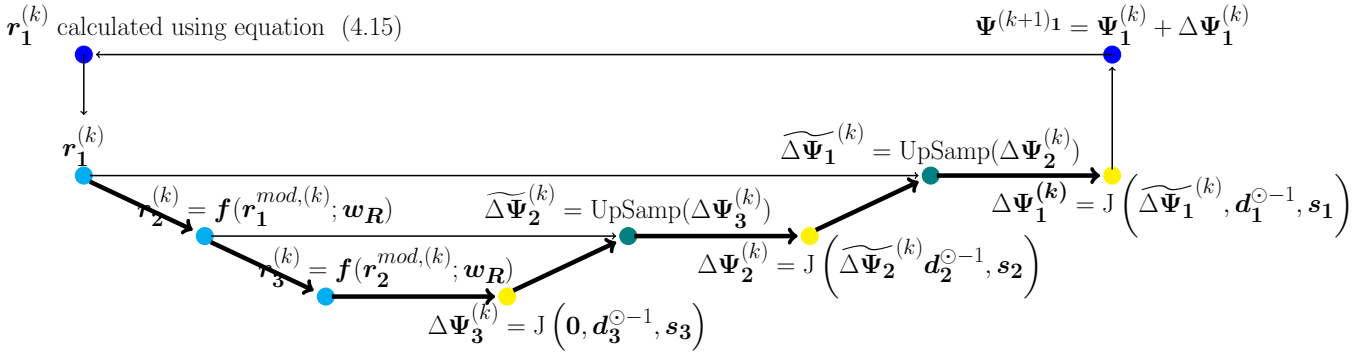


Figure 4.6: A sawtooth cycle multigrid iteration, with the subscript indicating the resolution and superscript representing the multigrid iteration and J($\cdot$) representing a Jacobi iteration. The residual is calculated and restricted twice. These residuals are used, along with Jacobi iterations, to determine the change in flux. After the finest level is reached, the flux is updated and the process repeats.

described by Equation (4.28). That is, as shown in Figure 4.6, a Jacobi iteration is performed on the lowest level (subscript 3) to determine $\Delta\boldsymbol{\Psi}_3^{(k)}$. This is prolongated to estimate $\widetilde{\Delta\boldsymbol{\Psi}_2}^{(k)}$ from which Jacobi smoothing is performed to obtain $\Delta\boldsymbol{\Psi}_2^{(k)}$. This repeats until the finest level is reached (subscript 1), where the flux is updated $(k+1)$ and the process is repeated for a number of multigrid iterations.

The residual calculation using convolutional layers is given by Equation (4.15) using the Petrov-Galerkin method. The restriction is done in two stages. In the first stage the residual is modified by multiplying the flux in each angle by its weight, p_n, and $\Delta x \Delta y$:

$$\boldsymbol{r}_{\mathbf{1},n}^{mod} = p_n \boldsymbol{r}_{\mathbf{1},n} \Delta x \Delta y, \tag{4.29}$$

in which k represents the iteration level (it increments after a multi-grid cycle such as shown in Figure 4.6) and in the second stage a convolutional operation is performed:

$$\boldsymbol{r}_{\mathbf{2}}^{(k)} = \boldsymbol{f}(\boldsymbol{r}_{\mathbf{1}}^{mod,(k)}; \boldsymbol{w_R}), \tag{4.30}$$

with filter weights:

$$\boldsymbol{w_R} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}. \tag{4.31}$$

An upsampling convolutional layer is applied from one solution level to the next finest level using the convolutional operation UpSamp($\cdot$), see Figure 4.6. This operation simply copies the coarser grid solution to the finner grid solutions. Thus, upsampling layers repeat values within an array, increasing the dimensions of the data [147], resulting in an approximation for the data at a finer mesh:

$$\widetilde{\boldsymbol{\Psi}_1}^{(k)} = \text{UpSamp}(\boldsymbol{\Psi}_2^{(k)}) \tag{4.32}$$

Although not shown in Figure 4.7 the Jacobi networks, given by the yellow boxes with a J($\cdot$), also have a set of μ_n and ν_n as inputs, which is dependent upon the restriction level in angle.

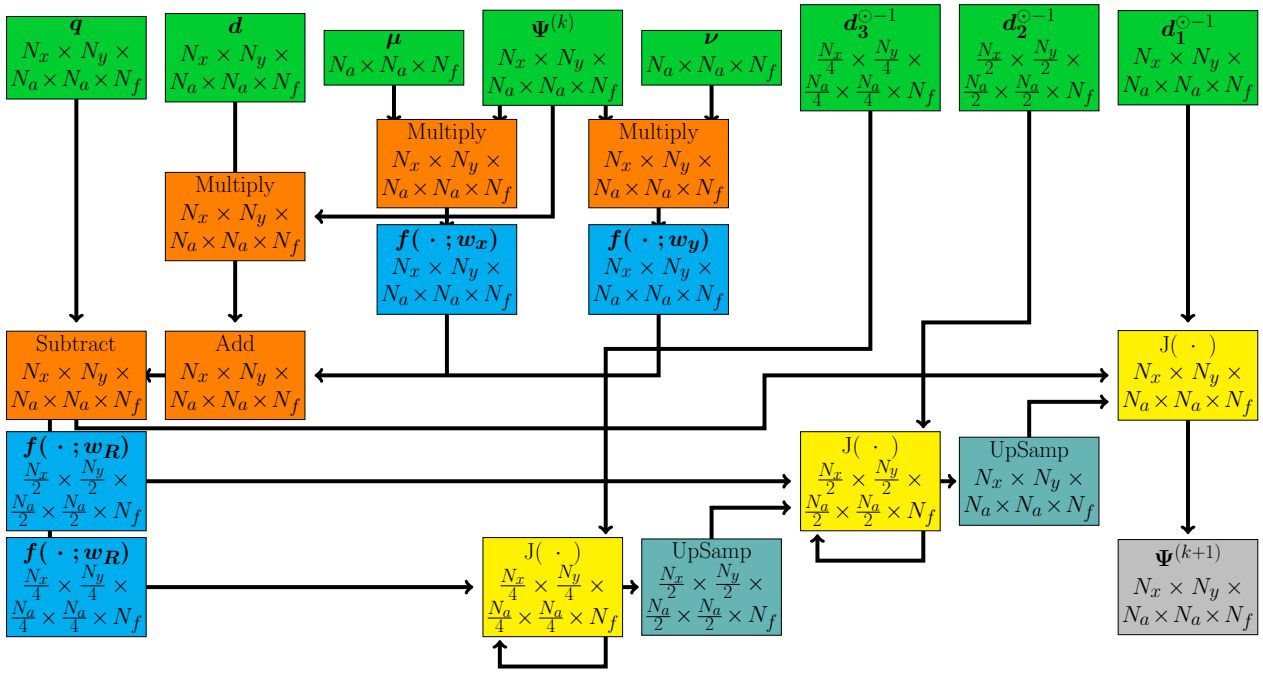


Figure 4.7: Multigrid network, MG, representing a single multigrid iteration. This network performs a single multigrid iteration on the flux of a single energy group. It takes the angular flux ($\boldsymbol{\Psi}^{(k)}$), source ($\boldsymbol{q}$), angular velocities ($\boldsymbol{\mu}$,$\boldsymbol{\nu}$) and the strictly diagonal coefficients ($\boldsymbol{d_1}$), along with the coarser resolution coefficients, as inputs (green boxes). A number of layer operations are performed, mathematical operations in orange, convolutional operations in cyan, sub-model operations in yellow and upsampling in teal. The sub-models can be iterated on multiple times. Finally, it outputs the flux of the next multigrid iteration flux ($\boldsymbol{\Psi}^{(k+1)}$). Arrow origins show which layer the data originated and the end of the arrow shows which layer takes that data as input. Dimensions of layers are given on the second line of each box.

Figure 4.7 shows how the multigrid iteration would look as a single network. Green boxes

contain the inputs, blue boxes are convolutional layers, orange boxes are mathematical functions as layers, yellow boxes are sub-networks, teal boxes are upsampling layers and the grey box is the output of the network. The second line in each box is the dimension of the output. This can be written as:

$$\boldsymbol{\Psi}_{\mathbf{1}}^{(k+1)} = \mathrm{MG}\left(\boldsymbol{\Psi}_{\mathbf{1}}^{(k)}, \boldsymbol{q}_{\mathbf{1}}, \boldsymbol{a}_{\mathbf{1}}, \boldsymbol{\mu}, \boldsymbol{\nu}, \boldsymbol{d}, \boldsymbol{d}_{\mathbf{1}}^{\odot -1}, \boldsymbol{d}_{\mathbf{2}}^{\odot -1}, \boldsymbol{d}_{\mathbf{3}}^{\odot -1}\right) \tag{4.33}$$

where MG is a function that calculates the result of one sawtooth multigrid iteration.

4.2.6 Optimised Anisotropic Non-Linear Petrov-Galerkin Dissipation

If $\boldsymbol{\Psi}_{\boldsymbol{x},n,g} = \boldsymbol{f}(\boldsymbol{\Psi}_{n,g}; \boldsymbol{w}_{\boldsymbol{x}})$ and $\boldsymbol{\Psi}_{\boldsymbol{y},n,g} = \boldsymbol{f}(\boldsymbol{\Psi}_{n,g}; \boldsymbol{w}_{\boldsymbol{y}})$ then a diffusion coefficient, based on the Petrov-Galerkin method in [173] and also presented in [172], can be given by:

$$\boldsymbol{k}_{\boldsymbol{abs}\,n,g} = \frac{\alpha_{kabs}|\boldsymbol{R}_{n,g}|h}{\epsilon_k + \frac{1}{\mathcal{D}}(|\boldsymbol{\Psi}_{\boldsymbol{x},n,g}| + |\boldsymbol{\Psi}_{\boldsymbol{y},n,g}|)} \,, \tag{4.34}$$

in which the dimensionality of the system here is $\mathcal{D} = 2$. A second diffusion coefficient, based on [165] and also described in [166, 172] all be it with the use of the 1-norm rather than 2-norms, can be given by:

$$\boldsymbol{k}_{\boldsymbol{square}\,n,g} = \frac{\alpha_{ksquare}{\boldsymbol{R}_{n,g}}^2 h}{\epsilon_k + \frac{1}{\mathcal{D}}(|\boldsymbol{q}_{n,g} \odot \boldsymbol{\Psi}_{\boldsymbol{x}n,g}| + |\boldsymbol{q}_{n,g} \odot \boldsymbol{\Psi}_{\boldsymbol{y}n,g}|)(\boldsymbol{\Psi}_{\boldsymbol{P}\boldsymbol{x}n,g}^2 + {\boldsymbol{\Psi}_{\boldsymbol{y}n,g}}^2)} \,, \tag{4.35}$$

where $\boldsymbol{q}_{n,g}$ is:

$$\boldsymbol{q}_{n,g} = \frac{\mu_n \boldsymbol{\Psi}_{\boldsymbol{x}n,g} + \nu_n \boldsymbol{\Psi}_{\boldsymbol{y}n,g}}{\epsilon_k + {\boldsymbol{\Psi}_{\boldsymbol{x}n,g}}^2 + {\boldsymbol{\Psi}_{\boldsymbol{y}n,g}}^2} \,. \tag{4.36}$$

For both diffusion coefficients, $h = \frac{1}{2}(\Delta x + \Delta y)$ and ϵ_k is a small value chosen so that the resulting diffusion does not become too large — it is set to $\epsilon_k = 0.001$ in this work. The $\mathcal{O}(1)$ scaling coefficients α_{kabs}, $\alpha_{ksquare}$ are defined here as:

$$\alpha_{kabs} = \frac{2^p}{16}, \qquad \alpha_{ksquare} = \frac{2^p}{2}, \tag{4.37}$$

where p is the polynomial order of the finite element expansion: $p = 1$ for linear 3×3 filters, $p = 2$ for quadratic 5×5 filters and $p = 3$ for cubic 7×7 filters. The scalars, Equations (4.37),

increase with increasing p because the residual $\boldsymbol{R}_{n,g}$ gets smaller with increasing p. If $p > 1$ then $\boldsymbol{R}_{n,g}$ can instead be determined by subtracting the lower order expansion from the higher order expansion:

$$\boldsymbol{R}_{n,g} = \left[\mu_n \boldsymbol{f}(\boldsymbol{\Psi}_{n,g}; \boldsymbol{w_x}) + \nu_n \boldsymbol{f}(\boldsymbol{\Psi}_{n,g}; \boldsymbol{w_y})\right] - \left[\mu_n \boldsymbol{f}(\boldsymbol{\Psi}_{n,g}; \boldsymbol{w_x^{Low}}) + \nu_n \boldsymbol{f}(\boldsymbol{\Psi}_{n,g}; \boldsymbol{w_y^{Low}})\right] \quad (4.38)$$

in which $\boldsymbol{w_x^{Low}}$, $\boldsymbol{w_y^{Low}}$ are lower order (by one) filters than those used in the rest of the simulation e.g. if the filter size in 3D is $5 \times 5 \times 5$ then the lower order size might be $3 \times 3 \times 3$. This $\boldsymbol{R}_{n,g}$ effectively forms the residual of the discrete system of equations and results in a particularly impressive performing method. To obtain high rates of convergence use:

$$\boldsymbol{R}_{n,g} = \left[\mu_n \boldsymbol{f}(\boldsymbol{\Psi}_{n,g}; \boldsymbol{w_x^{High}}) + \nu_n \boldsymbol{f}(\boldsymbol{\Psi}_{n,g}; \boldsymbol{w_y^{High}})\right] - \left[\mu_n \boldsymbol{f}(\boldsymbol{\Psi}_{n,g}; \boldsymbol{w_x}) + \nu_n \boldsymbol{f}(\boldsymbol{\Psi}_{n,g}; \boldsymbol{w_y})\right] \quad (4.39)$$

in which $\boldsymbol{w_x^{High}}$, $\boldsymbol{w_y^{High}}$ are higher order (by one) filters. Equation (4.39) can be derived by placing the current solution $\boldsymbol{\Psi}_{n,g}$ into the higher order discretised equations and then subtract the current discretised equations (which have a zero residual) from the high order discretised equations (which have a non-zero residual). The result is Equation (4.39). A mixed mass approach to residual approximation can be similarly derived to obtain the residual approximation:

$$\boldsymbol{R}_{n,g} = \beta_r \boldsymbol{f}(\ \mu_n \boldsymbol{f}(\boldsymbol{\Psi}_{n,g}; \boldsymbol{w_x}) + \nu_n \boldsymbol{f}(\boldsymbol{\Psi}_{n,g}; \boldsymbol{w_y});\ {m_L}^{-1}\boldsymbol{m} - \boldsymbol{I}), \quad (4.40)$$

where $m_L = \Delta x \Delta y$ is the lumped mass term and thus $m_L^{-1} = \frac{1}{\Delta x \Delta y}$, $\boldsymbol{m}$ is the mass filter associated with the finite element matrices, see appendix. β_r is a coefficient where 3 has been found effective. A maximum diffusion coefficient is considered to be the diffusion whose magnitude is the same as that of advection and is thus:

$$\boldsymbol{k_{max}}_{n,g} = \sqrt{(\Delta x \mu_n)^2 + (\Delta y \nu_n)^2}\ , \quad (4.41)$$

and the final diffusion coefficient, combining the above conservative values of diffusion, is given by:

$$\boldsymbol{k}_{n,g} = \boldsymbol{min}\{\boldsymbol{k_{max}}_{n,g}, \boldsymbol{k_{abs}}_{n,g}, \boldsymbol{k_{square}}_{n,g}\}. \quad (4.42)$$

Anisotropic Residual based Diffusion for Non-Linear Petrov-Galerkin Dissipation

A major advanatage of representing the discrete equation residual from just the gradients is that we can form residual contributions from different directions and in this way form anisotropic diffusion coefficients that stabilize the method. This is what we do here and is the approach used in the applications. Now the residual in each direction in tern can also be estimated from the mixed mass approach as:

$$\boldsymbol{R_{x}}_{n,g} = \alpha_r \mu_n \boldsymbol{f}(\ \boldsymbol{f}(\boldsymbol{\Psi}_{n,g}; \boldsymbol{w_x});\ {m_L}^{-1}\boldsymbol{m} - \boldsymbol{I}), \tag{4.43}$$

$$\boldsymbol{R_{y}}_{n,g} = \alpha_r \nu_n \boldsymbol{f}(\ \boldsymbol{f}(\boldsymbol{\Psi}_{n,g}; \boldsymbol{w_y});\ {m_L}^{-1}\boldsymbol{m} - \boldsymbol{I}). \tag{4.44}$$

This provides an opportunity to apply anisotropically the Petrov-Galerkin diffusion – that is to apply different diffusion in different directions. In its simplest form this can be achieved through for example (using the previous approach but for each coordinate in turn):

$$\boldsymbol{k_{xabs}}_{n,g} = \frac{\alpha_{kabs}|\boldsymbol{R_x}_{n,g}|\Delta x}{(\epsilon_k + |\boldsymbol{f}(\boldsymbol{\Psi}_{n,g}; \boldsymbol{w_x})|)}, \quad \boldsymbol{k_{yabs}}_{n,g} = \frac{\alpha_{kabs}|\boldsymbol{R_y}_{n,g}|\Delta y}{(\epsilon_k + |\boldsymbol{f}(\boldsymbol{\Psi}_{n,g}; \boldsymbol{w_y})|)}, \tag{4.45}$$

and the second set of diffusion coefficients are:

$$\boldsymbol{k_{x\ square}}_{n,g} = \frac{\alpha_{ksquare}{\boldsymbol{R_x}_{n,g}}^2\Delta x}{\epsilon_k + |\mu_n|{\boldsymbol{\Psi_x}_{n,g}}^2}. \quad \boldsymbol{k_{y\ square}}_{n,g} = \frac{\alpha_{ksquare}{\boldsymbol{R_y}_{n,g}}^2\Delta y}{\epsilon_k + |\nu_n|{\boldsymbol{\Psi_y}_{n,g}}^2}, \tag{4.46}$$

and the diffusion coefficients in the x- and y-directions respectively are:

$$\boldsymbol{k_x}_{n,g} = \boldsymbol{min}\{\boldsymbol{k_{x\ max}}_{n,g}, \boldsymbol{k_{x\ abs}}_{n,g}, \boldsymbol{k_{x\ square}}_{n,g}\}, \quad \boldsymbol{k_y}_{n,g} = \boldsymbol{min}\{\boldsymbol{k_{y\ max}}_{n,g}, \boldsymbol{k_{y\ abs}}_{n,g}, \boldsymbol{k_{y\ square}}_{n,g}\}, \tag{4.47}$$

in which $\boldsymbol{k_{x\ max}}_{n,g} = \Delta x|\mu_n|$ and $\boldsymbol{k_{y\ max}}_{n,g} = \Delta y|\nu_n|$. This form, Equation (4.47), of anisotropic diffusion is the approach followed here because on structured grids aligned with the diffusion directions this is particularly effective with minimal discretisation error. However, Equation (4.47) can be generalised further by avoiding assuming that the diffusion is aligned with the coordinate system. Initially, for simplicity, we will assume the system is 2D. This then enables us to define another residual from:

$$\boldsymbol{R_{xy}}_{n,g} = \alpha_r \boldsymbol{f}(\ \mu_n \boldsymbol{f}(\boldsymbol{\Psi}_{n,g}; \boldsymbol{w_x}) + \nu_n \boldsymbol{f}(\boldsymbol{\Psi}_{n,g}; \boldsymbol{w_y});\ {m_L}^{-1}\boldsymbol{m} - \boldsymbol{I}), \tag{4.48}$$

with an associated diffusion coefficient of

$$\boldsymbol{k_{xyabs}}_{n,g} = \frac{\alpha_{kabs}|\boldsymbol{R_{xy}}_{n,g}|}{(\epsilon_k + \frac{|\boldsymbol{f}(\boldsymbol{\Psi}_{n,g};\boldsymbol{w_x})|}{\mathcal{D}\Delta x} + \frac{|\boldsymbol{f}(\boldsymbol{\Psi}_{n,g};\boldsymbol{w_y})|}{\mathcal{D}\Delta y})}. \tag{4.49}$$

Now we have three diffusion coefficients so one might find the largest of these three coefficients and rotate the coordinate system so that it is aligned with this diffusion direction, either $(\mu_n, 0)^T$, $(0, \nu_n)^T$ then project the other two directions so they are orthogonal to this direction and choose the largest of the remaining diffusion coefficients (after the projection). This becomes the orthogonal diffusion then rotate the system back to the original system to form the final diffusion tensor. A similar approach can be applied in 3D but then 6 residuals are formed e.g.

$$\boldsymbol{R_{xyz}}_{n,g} = \alpha_r \boldsymbol{f}(\ \mu_n \boldsymbol{f}(\boldsymbol{\Psi}_{n,g};\boldsymbol{w_x}) + \nu_n \boldsymbol{f}(\boldsymbol{\Psi}_{n,g};\boldsymbol{w_y}) + \xi_n \boldsymbol{f}(\boldsymbol{\Psi}_{n,g};\boldsymbol{w_z});\ {m_L}^{-1}\boldsymbol{m} - \boldsymbol{I}), \tag{4.50}$$

as well as $\boldsymbol{R_{xy}}_{n,g}$ as defined above in Equation (4.48) and where ξ_n is the 3^{rd} direction on the unit sphere in the discrete ordinate angular discretisation.

4.2.7 Neural Network Filters for the Convolutional Finite Element Method (ConvFEM)

The basic idea is to add discretisation stencils together in order to form an average stencil that looks the same at every node of the FEM mesh. One can see the stencils are different by looking at the 1D quadratic element. For this element, the node at the centre of the element has a direct link (through the FEM stencil) only to the nodes of that element (3 of them including the node itself) while the other nodes have direct links to all the nodes of the 2 elements they belong too and are thus linked directly to 5 nodes including the current node. This difference in stencils can also be seen from node to node in Figure 4.8a. The basis functions for quadratic 1D element discretisations are shown in figure 4.8a. To form the new ConvFEM discretisation, these different discretisations are simply added together to form another discretisation which looks the same everywhere. If there is a lumped basis function then it looks like an average of the two basis functions – see Figure 4.8b.

For a 2D quadratic element (Figure 4.9a) the node in the centre of the element is connected

directly to only the nodes of the element (8 of them) and the edge nodes of the element are directly connected only to the 2 elements that it belongs to (15-1=14 of them) and the corner nodes are connected to all of the nodes belonging to the four elements that it belongs to (25-1=24 of them). At the bottom we have highlighted (green box of Figure 4.9a) the 4 nodes around which the stencils need to be averaged for 2D and for 3D quadratic elements there are similarly 8 nodes around which averaging is necessary. For cubic elements, one needs to average, in 2D, over the 9 nodes shown in Figure 4.9b with the green box around these 9 nodes.

For example, when adding the two equations together for 1D quadratic elements the mass discretisation becomes:

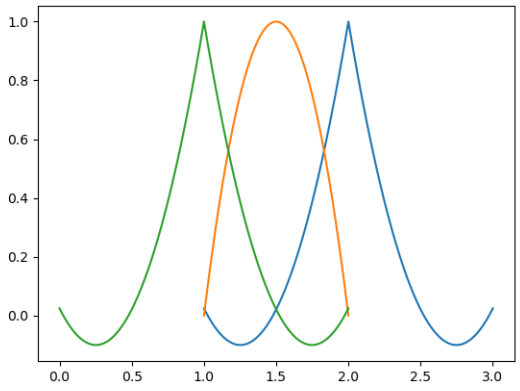
$$\int_E \frac{1}{2}(N_i \sum_j N_j C_j + S_i \sum_j S_j C_j) dV \tag{4.51}$$

in which N_i and S_i are the basis functions of the two discretisations in 1D and for diffusion discretisation we thus have:

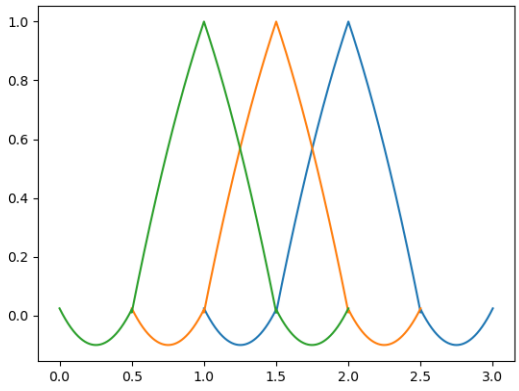
$$\int_E \frac{1}{2} \left(\frac{\partial N_i}{\partial x} \sum_j \frac{\partial N_j}{\partial x} C_j + \frac{\partial S_i}{\partial x} \sum_j \frac{\partial S_j}{\partial x} C_j \right) dV. \tag{4.52}$$

The finite element method (FEM) filter weights used in section 4.2.6 are constructed from the finite element basis functions. The basis functions for a number of nodes, dependent on the order, are summed together and averaged. The 1D quadratic basis functions are shown in Figure 4.8a. By taking the basis functions for the two neighbouring nodes, and averaging them, the overall basis functions for the same element are shown in Figure 4.8b. See the appendix for all the 2D filters needed for the ConvFEM discretisation used here.

The required elements needing to be averaged for quadratic and cubic elements are shown in Figure 4.9.

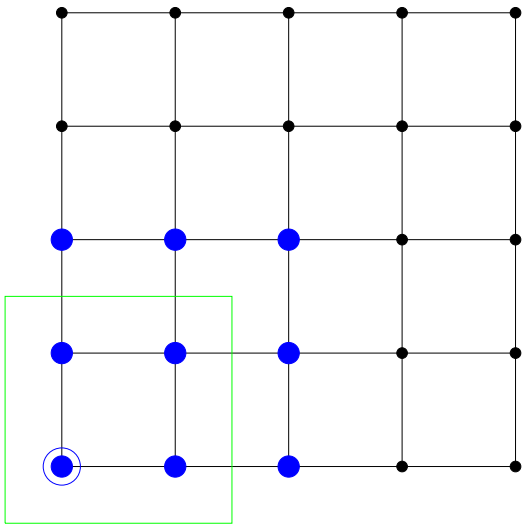


(a) Quadratic basis functions for a single element

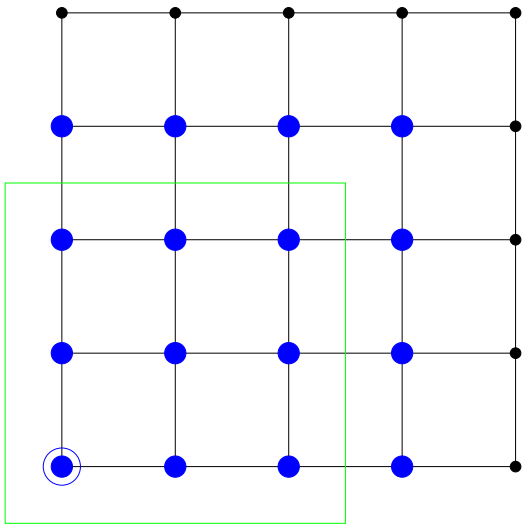


(b) Averaged basis functions for three elements

Figure 4.8: FEM Basis functions for quadratic 1D elements.



(a) Quadratic element indicating in the green box the nodal equations that are to be averaged.



(b) Cubic element indicating in the green box the nodal equations that are to be averaged.

Figure 4.9: Nodes within green boxes will have their associated discrete equations averaged. The blue nodes indicate the nodes of the 2D quadratic and cubic elements.

4.3 Results

4.3.1 Straight Duct problem

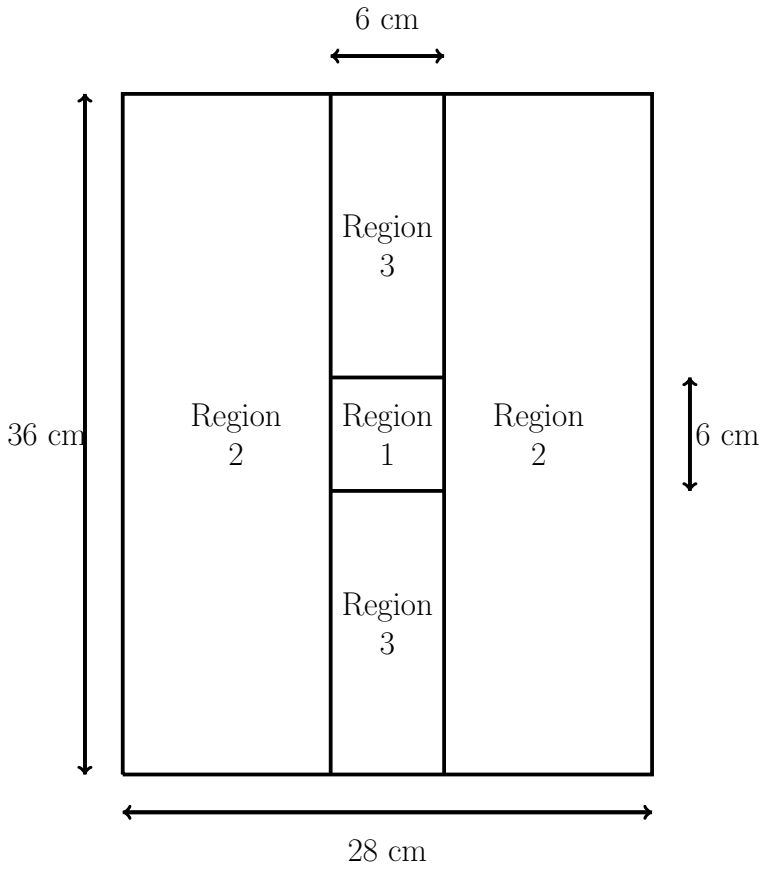


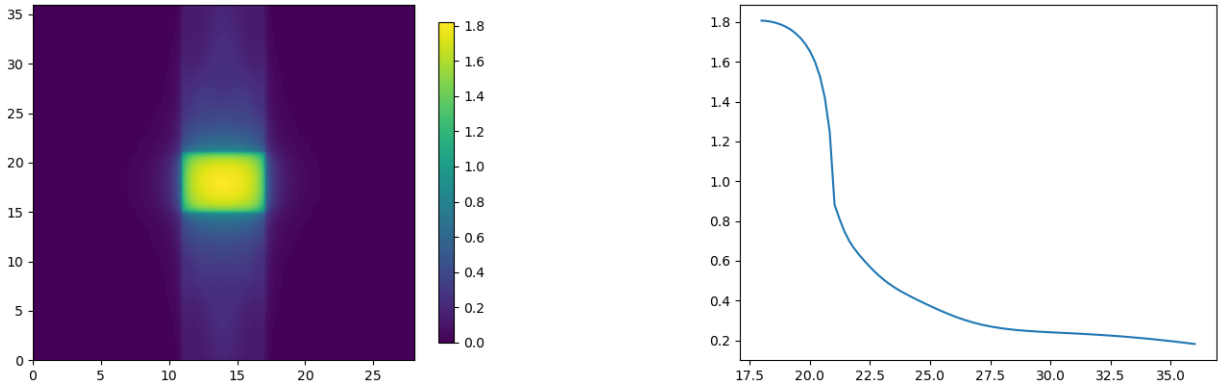
Figure 4.10: Diagram of single duct problem.

	Source (neutrons $\mathrm{cm}^{-2}\,\mathrm{s}^{-1}$)	Σ^a (cm^{-1})	Σ^s (cm^{-1}).
Region 1	1	0.5	0
Region 2	0	0.5	0
Region 3	0	0	0

Table 4.1: Source and cross sections for the three regions of the straight duct problem.

Figure 4.10 shows the diagram of the single duct problem. This domain is a 36cm × 28cm and contains a 6cm × 6cm source in the centre. Two regions of absorbing material are either side of the source and two ducts of width 6cm run along the y-axis either side of the source. The source and cross sections for the three regions are listed in table 4.1. This problem is mono-energetic, so no scattering occurs, and has a source, so no power iteration is required. Four different spatial resolutions are used: 45 × 35, 90 × 70, 180 × 140 and 360 × 280. These

four spatial resolutions have spacing between the nodes or cells of $\Delta x = 0.8$cm, $\Delta x = 0.4$cm, $\Delta x = 0.2$cm and $\Delta x = 0.1$cm respectively all with $\Delta y = \Delta x$. All dimensions of length are in cm. Simulations with $\Delta x = 0.8, 0.4, 0.2$ and 0.1 are resolved with $N_f = 8$ and $N_a = 4$ resulting in 128 total angular directions. For the finer resolution in angle simulations use $N_f = 8$ and $N_a = 8$ resulting in 512 total angular directions and $\Delta x = 0.1$. The finer resolution solutions were generated using 500 multigrid iterations and all other results in this section were generated using 100 multigrid iterations.



(a) Scalar flux (neutrons $\mathrm{cm}^{-2}\,\mathrm{s}^{-1}$) across the straight duct problem.

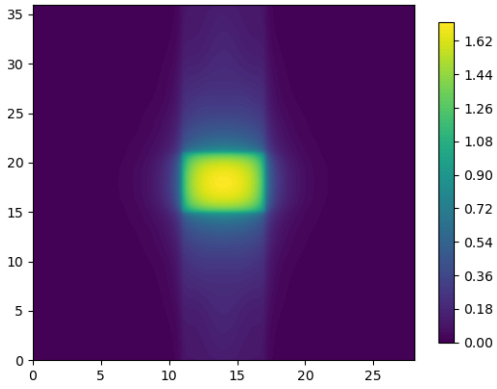
(b) Scalar flux (neutrons $\mathrm{cm}^{-2}\,\mathrm{s}^{-1}$) vs y (cm) at $x = 14$cm.

Figure 4.11: Solution for the straight duct problem generated using a neural network with a pure upwind solver with a spatial resolution of $\Delta x = 0.2$cm.

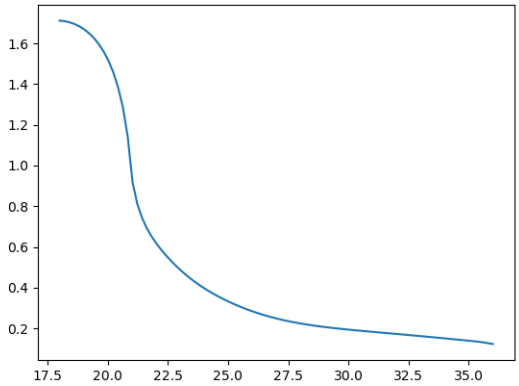
Figure 4.11 shows the solution for the single duct problem with the upwind method. Notice that there is a sharper change in flux at the boundary between source and tunnel, at $y = 21$cm. The centre of this problem is at $x = 14$cm and $y = 18$cm, which appears at the left-hand side of figure 4.11.

Figure 4.12 shows the solution for the single duct problem with Petrov-Galerkin residual added and using quadratic ConvFEM filters. Unlike Figure 4.11 the boundary interface at $y = 21$cm appears smoother. Figure 4.13 shows the Quadratic and Quintic ConvFEM filter solutions for $\Delta x = 0.1$ using 128 and 512 angles. It can be observed that the solution for 128 angles has a slight curve around the tail end, whereas the one with 512 angles does not have this. The solutions for $\Delta x = 0.1$ using 512 angles are therefore used for the subsequent comparisons.

Figure 4.14 shows the 1D flux profiles for all filters at $\Delta x = 0.8$ and $\Delta x = 0.4$. Figure 4.14a is the 1D flux profiles for $\Delta x = 0.8$ and this shows an obvious difference between all filters,

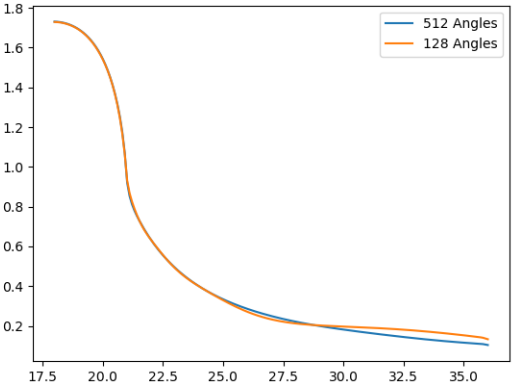


(a) Scalar flux (neutrons $\mathrm{cm}^{-2}\,\mathrm{s}^{-1}$) across the straight duct problem with the Petrov-Galerkin.

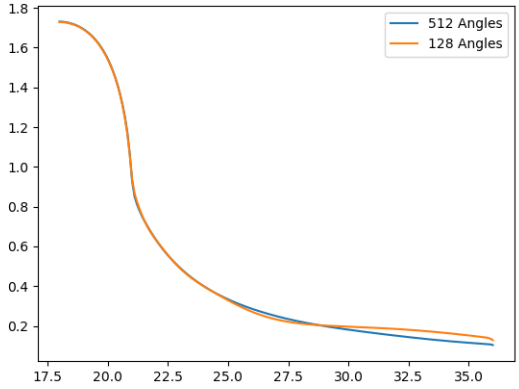


(b) Scalar flux (neutrons $\mathrm{cm}^{-2}\,\mathrm{s}^{-1}$) vs y(cm) at $x = 14$cm.

Figure 4.12: Solution for the straight duct problem with the Petrov-Galerkin generated using a neural network with Quintic ConvFEM filters with a spatial resolution of $\Delta x = 0.2$cm and using 128 angular directions.



(a) Scalar flux (neutrons $\mathrm{cm}^{-2}\,\mathrm{s}^{-1}$) vs y (cm) at $x = 14$cm generated using a neural network with Quadratic ConvFEM filters.

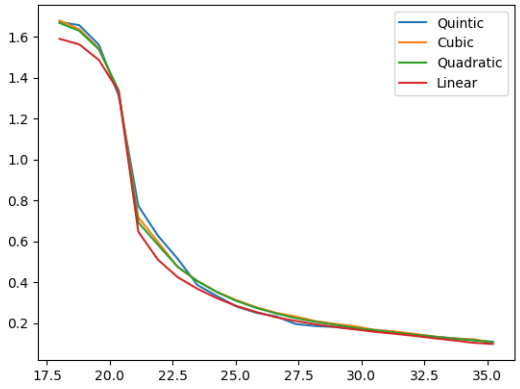


(b) Scalar flux (neutrons $\mathrm{cm}^{-2}\,\mathrm{s}^{-1}$) vs y (cm) at $x = 14$cm generated using a neural network with Cubic ConvFEM filters.

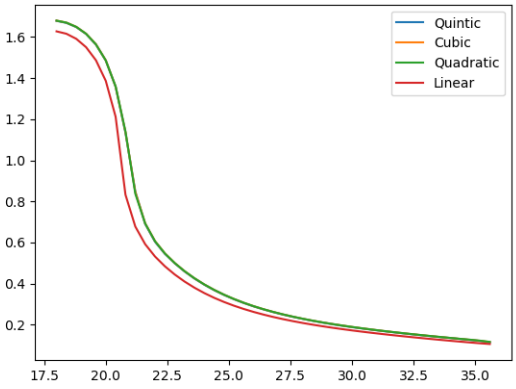
Figure 4.13: Scalar Flux generated using a neural network with Quadratic and Cubic ConvFEM filters for $\Delta x = 0.1$ using 128 and 512 angular directions.

with the linear filter solution being much lower at the centre. When Δx is decreased to 0.4, the linear filter is the only one that shows a difference, with the quadratic, cubic and quintic all producing the same solution. Figure 4.15 shows the 1D flux profiles for all filters at $\Delta x = 0.2$ and $\Delta x = 0.1$. At these values of Δx all four filters converge to the same solution.

Figure 4.16a shows the Quadratic ConvFEM filter solutions for the four different spatial resolutions. As the spatial mesh is refined the peak flux in the centre tends to increase. The

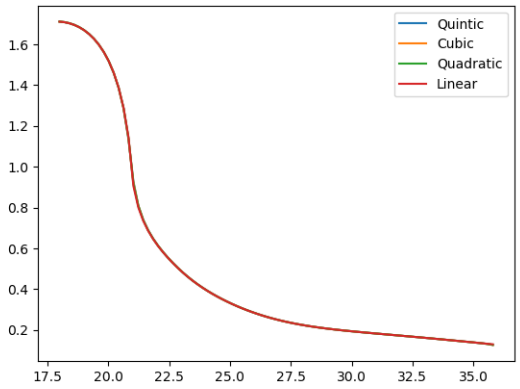


(a) Scalar flux (neutrons $\mathrm{cm}^{-2}\,\mathrm{s}^{-1}$) vs y (cm) at $x = 14$cm for $\Delta x = 0.8$ using 128 angular directions.

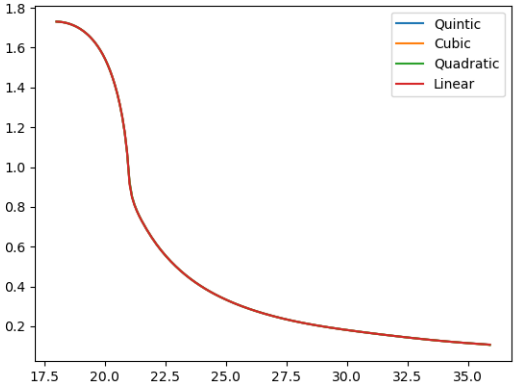


(b) Scalar flux (neutrons $\mathrm{cm}^{-2}\,\mathrm{s}^{-1}$) vs y (cm) at $x = 14$cm for $\Delta x = 0.4$ using 128 angular directions.

Figure 4.14: Scalar Flux generated using a neural network with Linear, Quadratic, Cubic and Quintic for $\Delta x = 0.8$ and $\Delta x = 0.4$.



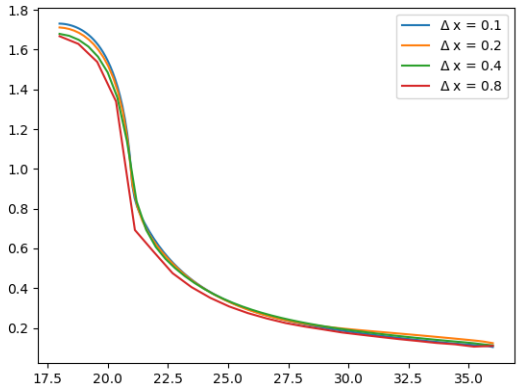
(a) Scalar flux (neutrons $\mathrm{cm}^{-2}\,\mathrm{s}^{-1}$) vs y (cm) at $x = 14$cm for $\Delta x = 0.2$ using 128 angular directions.



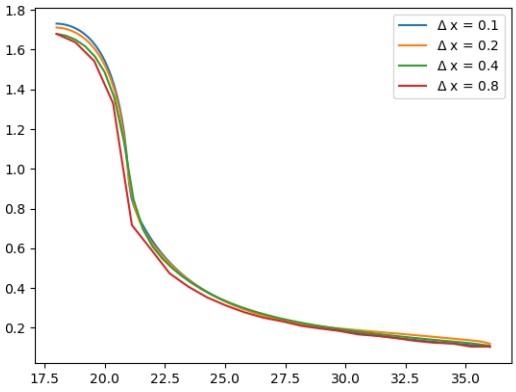
(b) Scalar flux (neutrons $\mathrm{cm}^{-2}\,\mathrm{s}^{-1}$) vs y (cm) at $x = 14$cm for $\Delta x = 0.1$ using 512 angular directions.

Figure 4.15: Scalar Flux generated using a neural network with Linear, Quadratic, Cubic and Quintic ConvFEM filters for $\Delta x = 0.2$ and $\Delta x = 0.1$.

same pattern is seen when the cubic filters are used, shown in Figure 4.16b. Figure 4.17 shows the error at the domain centre: $x = 18$cm and $y = 14$cm for each solution compared to the highest resolution solution, which is generated using the ConvFEM Quintic 9×9 filter with $\Delta x = 0.1$. It can be observed that the error decreases as the order of the filter increases, with the exception of $\Delta x = 0.2$.



(a) Scalar flux (neutrons $\mathrm{cm}^{-2}\,\mathrm{s}^{-1}$) vs y (cm) at $x = 14$cm generated using a neural network with Quadratic ConvFEM filters.



(b) Scalar flux (neutrons $\mathrm{cm}^{-2}\,\mathrm{s}^{-1}$) vs y (cm) at $x = 14$cm generated using a neural network with Cubic ConvFEM filters.

Figure 4.16: Scalar Flux generated using a neural network with Quadratic and Cubic ConvFEM filters for $\Delta x = 0.2, 0.4$ and 0.8 using 128 angular directions and $\Delta x = 0.1$ using 512 angular directions.

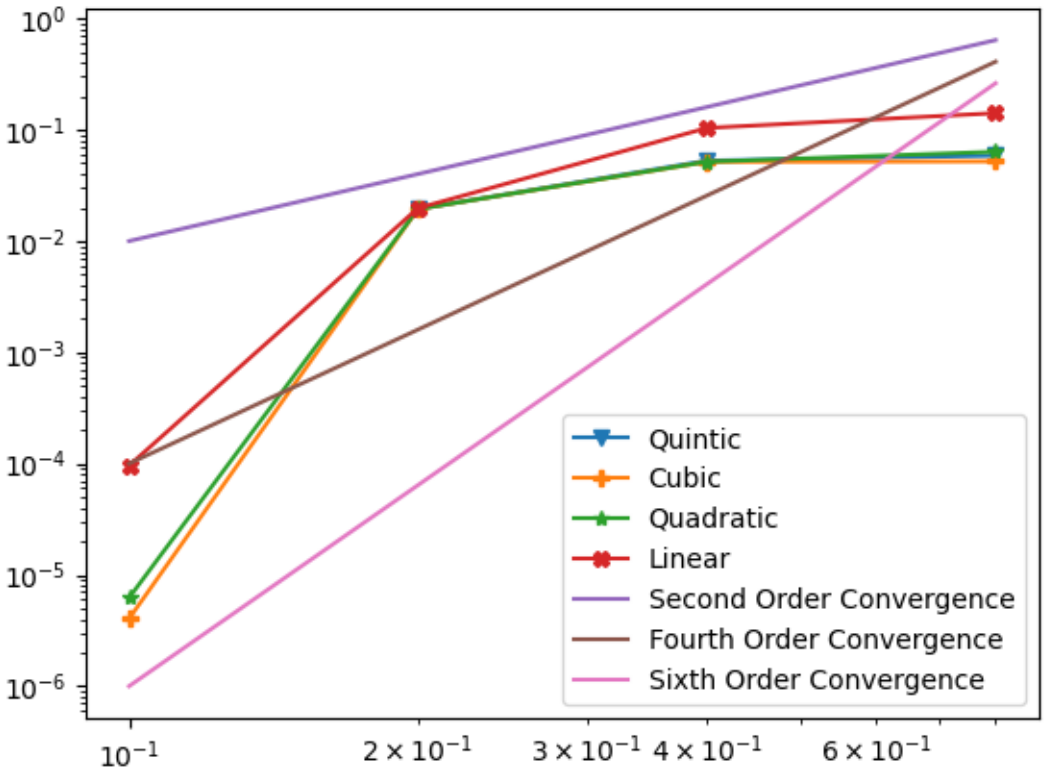


Figure 4.17: Error at $x = 18$cm and $y = 14$cm compared to the ConvFEM Quintic filter solution at $\Delta x = 0.1$ vs Δx. Second, Fourth and Sixth Order Convergence are plotted for reference.

4.3.2 Fuel Assembly

The multi-group iteration network is now used to produce solutions for a 2D fuel assembly. This fuel assembly is based on the KAIST benchmark [149] and uses their cross-sections.

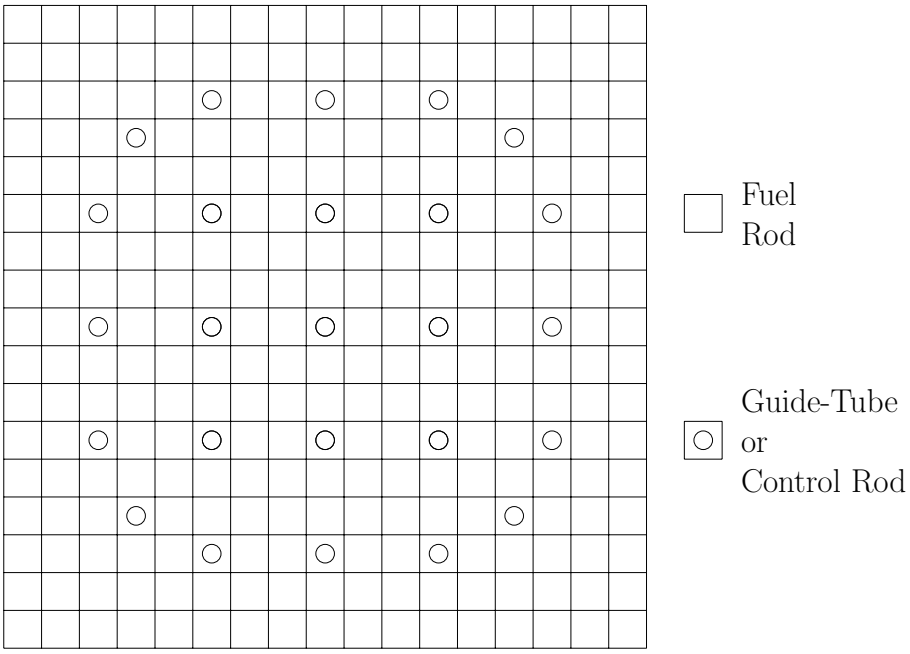


Figure 4.18: Geometry of UOX fuel assembly.

Figure 4.18 contains the geometry for the UOX fuel assembly. This consists of a 17×17 lattice containing either UOX fuels rods, guide-tubes or control rods. Each square in the figure contains 8×8 cells, forming a total of $18,496$ cells along with 544 cells to enforce the boundary conditions. The energy was discretised into seven groups, meaning the fuel assembly has $133,280$ degrees of freedom. Figure 4.19 contains the material parameters for the nodes in each pin. Each side of the fuel assembly is length 21.42cm meaning each cell is 0.1575cm $\times$ 0.1575cm. Each side also has Vacuum boundary conditions applied to it. The material parameters for each pin are shown in figure 4.19. For solutions of the fuel assembly, two configurations are used to produce the solutions. Figure 4.18 contains spaces that can either be guide-tubes or control rods. In the first configuration, all of these spaces are guide-tubes, representing a system where the control rods are fully withdrawn. In the second configuration, all of these spaces are control rods, representing a system where the control rods are fully inserted. This test case is multi-group and the power method [31] is the method chosen here to determine the dominant eigenvalue for this problem. The implementation of the power method used here is the same as described

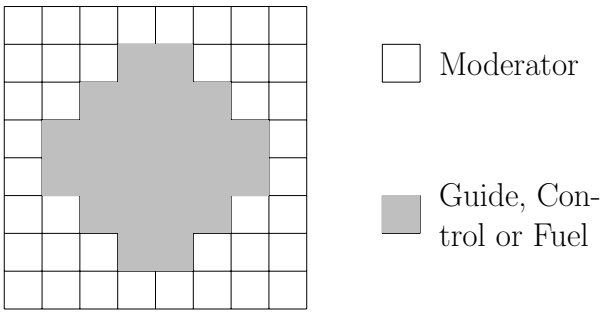
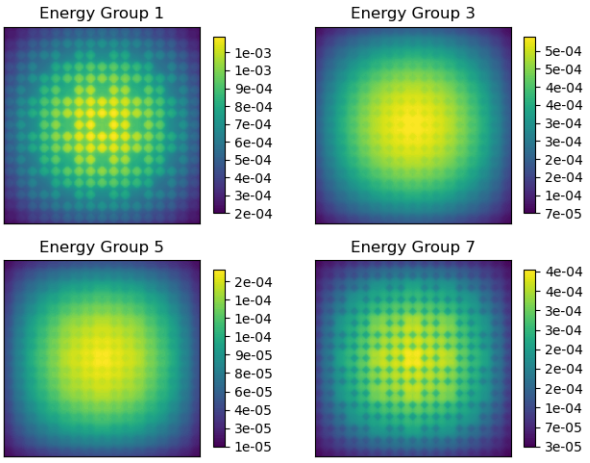
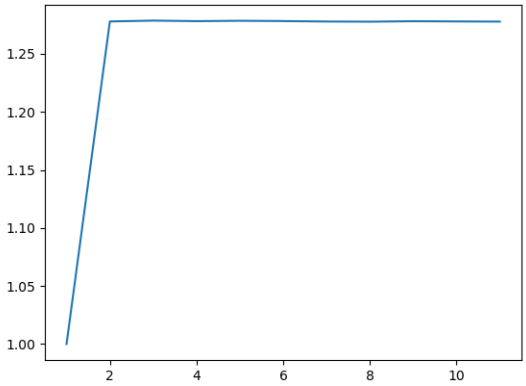


Figure 4.19: Geometry of pins.

in chapter 2 and the implementation of the multi-group network is the same as [174]. For all solutions in this section, 3 Jacobi iterations, 100 multigrid iterations and 100 multi-group iterations are performed.



(a) Scalar flux (neutrons $\mathrm{cm}^{-2}\,\mathrm{s}^{-1}$) across the fuel assembly for four energy groups for all control rods withdrawn.

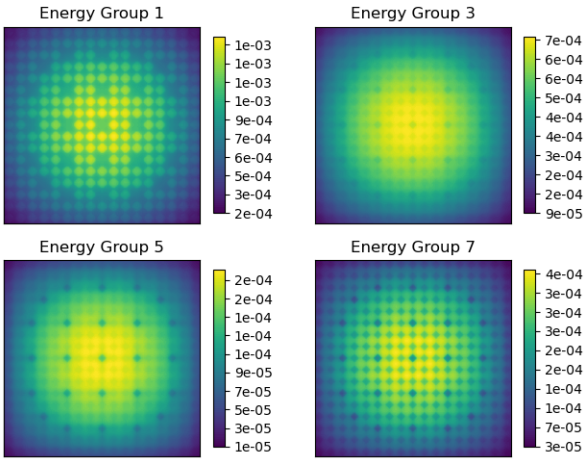


(b) k_{eff} convergence for a fuel assembly solution for all control rods withdrawn.

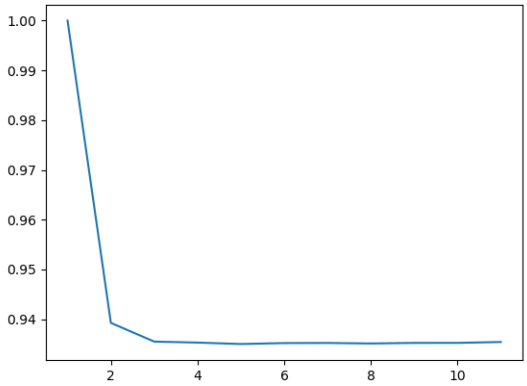
Figure 4.20: Fuel assembly flux and k_{eff} convergence for a fuel assembly solution with control rods fully withdrawn, generated using a neural network with the upwind method.

Figures 4.20 and 4.21 contain the scalar flux solution for a fuel assembly with control rods fully withdrawn and fully inserted respectively. Both solutions were generated using the upwind method. k_{eff} is notably lower when the control rods are fully inserted, as would be expected.

Figures 4.22 and 4.23 contain the scalar flux solution for a fuel assembly with control rods fully withdrawn and fully inserted respectively. Both solutions were generated using a neural network with Quadratic ConvFEM filters and the Petrov-Galerkin method. The solutions are similar to the upwind, although k_{eff} is slightly higher for both solutions. The solution using the upwind method, Figures 4.20 and 4.21, are similar to the Petrov-Galerkin method with quadratic ConvFEM filters, see Figures 4.22 and 4.23. Although as expected the added

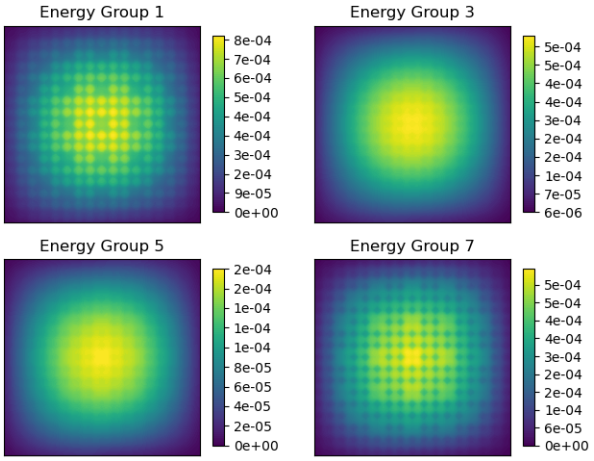


(a) Scalar flux (neutrons $\mathrm{cm}^{-2}\,\mathrm{s}^{-1}$) across the fuel assembly for four energy groups for all control rods inserted.

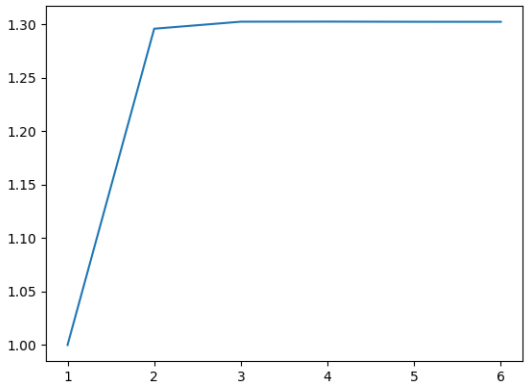


(b) k_{eff} convergence for a fuel assembly solution for all control rods inserted.

Figure 4.21: Fuel assembly flux and k_{eff} convergence for a fuel assembly solution with control rods fully inserted, generated using a neural network with the upwind method.



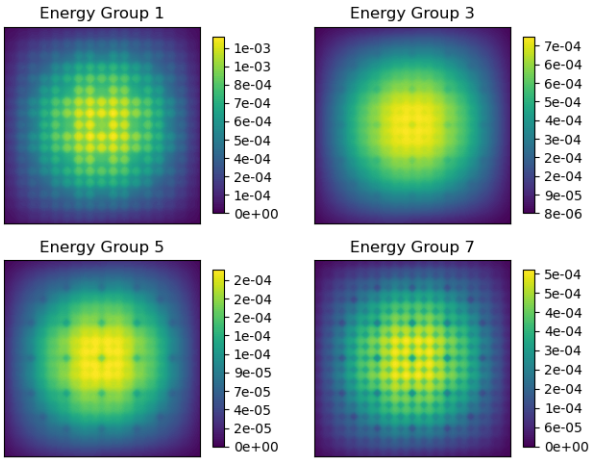
(a) Scalar flux (neutrons $\mathrm{cm}^{-2}\,\mathrm{s}^{-1}$) across the fuel assembly for four energy groups for all control rods withdrawn.



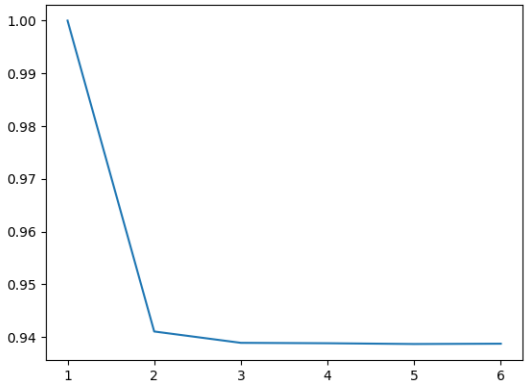
(b) k_{eff} vs power iteration for a fuel assembly solution for all control rods withdrawn.

Figure 4.22: Fuel assembly flux and k_{eff} convergence for a fuel assembly solution with control rods fully withdrawn, generated using a neural network with Quadratic ConvFEM filters and the Petrov-Galerkin method.

dissipation associated with the upwind method results in a slight reduction in the maximum scalar flux in all the energy groups.



(a) Scalar flux (neutrons $\mathrm{cm}^{-2}\,\mathrm{s}^{-1}$) across the fuel assembly for four energy groups for all control rods inserted.



(b) k_{eff} vs power iteration for a fuel assembly solution for all control rods inserted.

Figure 4.23: Fuel assembly flux and k_{eff} convergence for a fuel assembly solution with control rods fully inserted, generated using a neural network with Quadratic ConvFEM filters and the Petrov-Galerkin method.

4.4 Summary

In this chapter, we extend the methods of chapter 3 to the Boltzmann transport equation. These methods are extended to a space-angle multigrid solution method, which can extract the level of parallelism necessary to run efficiently on GPUs or new AI computers. Demonstrating that the neural network solvers presented here work on the full transport problem means that, at a minimum, the computational speed-up would be equivalent to those presented for the diffusion approximation in chapter 3. This occurs if angles are resolved sequentially, whereas if the architecture is set up to solve all angles simultaneously, the computational speed up would be much larger.

Additionally, the Convolutional Finite Element Method (ConvFEM) is developed. This enables one to have the same stencil at every node of the finite element mesh, unlike other high-order FEM approaches. This is important as it enables one to have simpler implementations using AI software, as the number of filters is greatly reduced. One would need a different filter for every FEM node stencil for conventional FEM approaches. The accuracy of the higher order (quadratic, cubic and quintic elements with 5×5, 7×7 and 9×9 stencils compared to the 3×3 stencils of linear FEM) methods was shown here. The linear FEM and ConvFEM are the same method and the only difference is between the higher order FEM and ConvFEM approaches.

Finally, A new non-linear Petrov-Galerkin method that introduces dissipation anisotropically is introduced. The solutions are not stable when purely central or Bubnov-Galerkin discretisations are used but the additional Petrov-Galerkin dissipation helps stabilise them.

The use of AI libraries is an attractive approach due to several benefits. It allows one to utilize highly optimised software, run on various computer architectures, and benefit from the large quantity of community-developed AI and ML applications (e.g. mixed arithmetic precision or model parallelism). Taking the lead from large neural networks that have been created and new AI computers (such as Cerebras 2 with nearly a million cores), this approach can be used for exascale computation in theory. Furthermore, as the overall method is essentially a neural network, experience with other neural network applications suggests it can be combined with other neural networks that solve other equations (e.g. fluid equations) to solve multi-physics problems by coupling different physics-based models for a more comprehensive view of the reactor.

Future directions include optimizing the code and methods to be able to run large 3D problems on multiple GPUs or new AI computers and testing the performance of the methods on these computers, as well as extending the approach to unstructured meshes.

Chapter 5

Reduced-Order Modelling

This chapter covers Proper-Orthogonal Decomposition which is used in Chapters 6, 7 and 8. It also covers autoencoders and their variants, which are used in chapters 6 and 7.

5.1 Introduction

This section briefly describes the key stages of constructing a projection-based reduced-order model, including explanations of proper orthogonal decomposition (POD) and the method of snapshots, singular value decomposition, selecting POD basis functions and constructing a projection-based reduced-order model. Finally, an overview of autoencoders is given, along with a description of variational and hybrid SVD-AE is given.

5.2 Proper Orthogonal Decomposition

Proper Orthogonal Decomposition (POD), introduced by Lumley [175], is a method used to identify the most energetic coherent structures within turbulent flow fields. This technique can be used for data acquired from experiments as well as from numerical simulations. In the case of numerical data, the numerical data is stored in a matrix, $\boldsymbol{S}$, with columns as individual snapshots of the solution of the high-fidelity model for a particular set of parameters. The matrix $\boldsymbol{S}$ is N by M, where N is the number of spatial degrees of freedom and M is the number of snapshots. POD allows a snapshot to be decomposed into a finite number of coefficients (α_i)

and POD basis functions or modes ($\boldsymbol{\psi}_i$) as follows:

$$\boldsymbol{\phi} = \sum_i \alpha_i \boldsymbol{\psi}_i \,, \tag{5.1}$$

where α_i is a scalar and $\boldsymbol{\phi}$, $\boldsymbol{\psi}_i \in \mathbb{R}^N$. POD gives the optimal linear representation of the data in the snapshots' matrix. In order to find the POD basis functions, the average squared error between the snapshots and their projection onto the basis functions is minimised [57]. This leads to an eigenvalue problem, which was originally written in the form

$$\boldsymbol{S}\boldsymbol{S}^T\boldsymbol{\psi}_j = \mu_j\boldsymbol{\psi}_j \quad \forall j \in N \,, \tag{5.2}$$

where μ_j is the eigenvalue, $\boldsymbol{\psi}_j$ is the eigenvector, and, without loss of generality, it is assumed that the eigenvalues are given in descending order and that the eigenvectors have been orthonormalised. For a large number of degrees of freedom, this problem quickly becomes challenging to solve, and it was Sirovich [54] who noted that the same (non-zero) eigenvalues could be found by instead solving the following, more tractable problem

$$\boldsymbol{S}^T\boldsymbol{S}\boldsymbol{\varphi}_k = \mu_k\boldsymbol{\varphi}_k \quad \forall k \in M \,, \tag{5.3}$$

for eigenvectors $\boldsymbol{\varphi}_k$. Called the method of snapshots, its lower computational burden has led to a wide uptake of POD for analysing results from numerical simulations [5, 176]. Assuming, again, that the eigenvalues are arranged in descending order, the eigenvectors of the two problems (5.2) and (5.3) are related through

$$\boldsymbol{\psi}_k = \frac{1}{\sqrt{\mu_k}}\boldsymbol{S}\boldsymbol{\varphi}_k \quad \forall k \in M \,. \tag{5.4}$$

Finding the POD basis functions can also be done by taking a singular value decomposition of the snapshots' matrix [57], which we now describe briefly. For a real $N \times M$ matrix S, where $N \geqslant M$, the singular value decomposition of $\boldsymbol{S}$ is as follows:

$$\boldsymbol{S} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^{\boldsymbol{T}}, \tag{5.5}$$

where $\boldsymbol{U}$ is an $N \times N$ matrix consisting of orthonormalised eigenvectors associated with the M largest eigenvalues of $\boldsymbol{S}\boldsymbol{S}^T$ and $\boldsymbol{V}$ is an $M \times M$ matrix consisting of orthonormalised eigenvectors associated with $\boldsymbol{S}^T\boldsymbol{S}$. Here, the POD basis functions are the columns of $\boldsymbol{U}$. The matrix Σ

contains the non-negative square roots of the eigenvalues of $\boldsymbol{S}^T\boldsymbol{S}$ (called the singular values) on its diagonal. These singular values are ordered as follows:

$$\sigma_1 \geqslant \sigma_2 \geqslant \cdots \geqslant \sigma_M \geqslant 0\,. \tag{5.6}$$

A reduced rank approximation to $\boldsymbol{S}$ can be found by setting to zero all but the first P largest singular values in the matrix Σ, and then pre- and post-multiplying by $\boldsymbol{U}$ and $\boldsymbol{V}^T$ respectively. This will give the optimal rank P approximation of the matrix $\boldsymbol{S}$ according to the Eckart-Young-Mirsky theorem [177].

5.3 POD Basis Functions

For some problems, a few basis functions will capture much of the behaviour of the system (as represented by the snapshots), and this is seen by the magnitude of the first few singular values being much larger than the remaining singular values. In such cases, the POD basis can be truncated without introducing much error. To determine how many basis functions should be kept, an empirical expression can be used that relates the number of basis functions retained to the fraction of information captured by them. The amount of information carried by each basis function depends on the square of its singular value. Suppose that the fraction of information of the original system to be captured is γ, where $0 \leqslant \gamma \leqslant 1$, then the lowest integer value of P is sought, such that the following is satisfied:

$$\frac{\sum_{i=1}^{P} \sigma_i^2}{\sum_{i=1}^{M} \sigma_i^2} \geqslant \gamma\,. \tag{5.7}$$

After truncation is applied (if indeed it is), the basis functions are stored in the matrix $\boldsymbol{R} \in \mathbb{R}^{N\times P}$. If the SVD has been used then the first P basis functions from $\boldsymbol{U}$ are retained to make up the columns of the matrix $\boldsymbol{R}$. If no truncation is applied then $P = M$ (i.e. the number of basis functions is equal to the number of snapshots). If either of the related eigenvalue problems are solved to determine the basis functions, then, once orthonormalised, the first P eigenvectors are retained to make up $\boldsymbol{R}$, either from Equation (5.2) or Equations (5.3) and (5.4). The complexity of solving the eigenvalue problems is $\mathcal{O}(N^3)$ for Equation (5.2) and $\mathcal{O}(M^3)$ for Equation (5.3). For an SVD, the complexity is $\mathcal{O}(NM^2)$ (see Golub et al. [31]), thus, for large problems (many snapshots but where $N \gg M$), the eigenvalue problem in Equation (5.3) will

be the cheapest to solve.

Once $\boldsymbol{R}$ has been determined, the P reduced variables $\boldsymbol{\alpha}$ associated with a snapshot $\boldsymbol{\phi}$ can be determined from the basis functions by:

$$\boldsymbol{\alpha} = \boldsymbol{R^T \phi}, \tag{5.8}$$

and the cell-based scalar flux values can be recovered from the reduced variables by:

$$\boldsymbol{\phi} = \boldsymbol{R\alpha}\,, \tag{5.9}$$

where $\boldsymbol{R}$ represents the POD basis functions and $\boldsymbol{\alpha}$ the POD coefficients or reduced variables. Equation (5.9) is the matrix equivalent of Equation (5.1).

5.4 Constructing a Reduced-Order Model for Criticality with POD

Having found the POD basis functions for the system, the discretised governing equations can be projected onto the reduced space. By inserting Equation (5.9) into Equation (2.12) and pre-multiplying by $\boldsymbol{R}^T$, the reduced system of equations is found:

$$\boldsymbol{R^T A R\alpha} = \boldsymbol{\lambda R^T B R\alpha} \tag{5.10}$$

where the matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ both depend on the material parameters Σ^a, Σ^s, Σ^f. This reduced system of equations has matrices of size P by P, whereas the original system given in Equation (2.12) is N by N where $N \gg P$.

In this research, the aim is to construct a reduced-order model based on a set of snapshots, each of which corresponds to a particular configuration, and to use this model to predict solutions for previously unseen configurations. An important point to address is how to modify the reduced system in Equation (5.10) for unseen parameters. One possibility is to exploit an affine decomposition of the matrices $\boldsymbol{A}$ and $\boldsymbol{B}$, by calculating $\boldsymbol{R}^T\boldsymbol{A}_i\boldsymbol{R}$ and $\boldsymbol{R}^T\boldsymbol{B}_i\boldsymbol{R}$ as part of the offline stage, where $\boldsymbol{A}_i$ is the ith matrix in the affine decomposition of $\boldsymbol{A}$, similarly for $\boldsymbol{B}_i$. These matrices could be interpolated in order to approximate the matrices corresponding to

an unseen parameter [85]. Although accurate for the problem solved in Heaney et al. [85], this method can in general be inaccurate [178] for unseen parameters. Another approach would be to use the Matrix Discrete Empirical Interpolation Method [179], which involves sampling the high-fidelity model matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ at a relatively low number of points before pre- and post-multiplying by $\boldsymbol{R}^T$ and $\boldsymbol{R}$ respectively, in order to estimate $\boldsymbol{A}$ and $\boldsymbol{B}$ for the unseen parameter. For both these methods, the sampling of $\boldsymbol{A}$ and $\boldsymbol{B}$, and their pre- and post-multiplication by the POD basis functions would be part of the offline stage, whereas the interpolation of the resulting reduced matrices would be part of the online stage. As the aim of this research is to demonstrate an autoencoder-based ROM for eigenvalue problems, a simple method is chosen which avoids approximation due to sampling but would be impractical for large problems. To evaluate $\boldsymbol{R}^T\boldsymbol{A}\boldsymbol{R}$ and $\boldsymbol{R}^T\boldsymbol{B}\boldsymbol{R}$ for an unseen parameter, the high-fidelity model is used to assemble $\boldsymbol{A}$ and $\boldsymbol{B}$ (Equations (2.8) and (2.12)) and then pre- and post-multiplied by $\boldsymbol{R}^T$ and $\boldsymbol{R}$ respectively. This would not be practical for systems with large number of degrees of freedom, as the online stage, in which $\boldsymbol{R}^T\boldsymbol{A}\boldsymbol{R}$ and $\boldsymbol{R}^T\boldsymbol{B}\boldsymbol{R}$ are approximated for the unseen parameter, should be independent of the high-fidelity model for reasons of computational efficiency. Future work will involve larger-scale problems, for which more computationally efficient methods will be investigated.

For the POD-based ROM, the offline stage consists of solving the high-fidelity model many times for different material parameters (cross-sections), see Equation (2.12) and Algorithm 2. In this research, the POD basis functions are determined by solving the eigenvalue problem associated with the method of snapshots, see Equation (5.3). The online stage of the POD-based ROM consists of assembling the matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ for a given set of parameters, projecting these matrices onto the reduced space and then solving with the power method. The outer iteration algorithm for the POD-based ROM is very similar to that of the high-fidelity model (Algorithm 2), except for the passing down of the POD basis functions from the outer to the inner iterations. The so-called inner iterations are described for POD-based ROM in Algorithm 3. (Although there are no iterations, the description 'inner iterations' is still used for Algorithm 3). Here the reduced-order model, Equation (5.10), is used to solve for the scalar fluxes. The right-hand side of Equation (5.10) is evaluated using the current value of scalar flux, resulting in a system that can be solved for the reduced variables. As the system is small, the problem is solved directly with Gaussian elimination. There is no need to use the eigenvalue in the source term (see line 2) as, once passed back to the outer iterations, the flux is normalised.

However, it is included here as it is needed in the inner iterations for the autoencoder-based reduced-order models seen in the next chapter.

Algorithm 3 POD-based reduced-order model: inner iterations

1: **function** POD_INNER_ITERATIONS($\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{\phi}$, λ, $\boldsymbol{R}$)
2: $\boldsymbol{s} = \lambda \left(\boldsymbol{R}^T \boldsymbol{B}\right) \boldsymbol{\phi}$! *set a source with values from the outer iterations*
3: solve the reduced-order model for $\boldsymbol{\alpha}$:
4: $\left(\boldsymbol{R}^T \boldsymbol{A} \boldsymbol{R}\right) \boldsymbol{\alpha} = \boldsymbol{s}$
5: $\boldsymbol{\phi} = \boldsymbol{R}\boldsymbol{\alpha}$! *find the updated scalar flux from the reduced variables*
6: **return** $\boldsymbol{\phi}$

5.5 A Standard Autoencoder

An autoencoder is a special type of feed-forward neural network that is trained to learn the identity map. A bottleneck at its central layer forces the autoencoder to learn a reduced representation of the data. The simplest (vanilla) autoencoder has an input layer with n neurons, a hidden layer with m neurons where $m < n$, and an output layer with n neurons. The values of the neurons in the hidden layer define the latent space and can be referred to as the latent variables. Networks with additional hidden layers can learn more complicated patterns creating a more effective representation of complex data. An autoencoder can be split into two networks: the encoder, which maps an input to the latent variables, and a decoder, which maps from the latent variables to the output. If the input vector is given by $\boldsymbol{x} \in \mathbb{R}^n$, the output vector by $\hat{\boldsymbol{x}} \in \mathbb{R}^n$ and the latent variables of the hidden layer by $\boldsymbol{y} \in \mathbb{R}^m$, then the autoencoder can be written as a composition of functions as follows

$$\hat{\boldsymbol{x}} = f^{\mathrm{AE}}(\boldsymbol{x}; \boldsymbol{w}^{\mathrm{AE}}) = f^{\mathrm{DEC}}\left(\boldsymbol{y}; \boldsymbol{w}^{\mathrm{DEC}}\right) = f^{\mathrm{DEC}}\left(f^{\mathrm{ENC}}(\boldsymbol{x}; \boldsymbol{w}^{\mathrm{ENC}}); \boldsymbol{w}^{\mathrm{DEC}}\right) \tag{5.11}$$

where f^{AE}, f^{ENC} and f^{DEC} represent the autoencoder, encoder and decoder respectively, with associated weights $\boldsymbol{w}^{\mathrm{AE}}$, $\boldsymbol{w}^{\mathrm{ENC}}$ and $\boldsymbol{w}^{\mathrm{DEC}}$.

Figure 2.1 illustrates a fully-connected autoencoder with 6 layers (only layers with trainable weights are included in the total), 5 of these are hidden layers. Neurons can have connections with sending neurons from the previous layer (except for those in the input layer) and with receiving neurons from the following layer (except for those in the output layer). In this diagram, the input and output layers have 6 neurons each and the latent space is of dimension 1. The grey arrows indicate how the information passes through the autoencoder from left to right.

The input to a neuron is calculated as a weighted sum of the outputs of the sending neurons to which it is connected and a bias term. The output of the neuron is the evaluation of the activation function for this weighted sum. For example, the output of the ith neuron of the hidden layer of the vanilla autoencoder described above, is given by

$$y_i = g\left(\sum_{j=1}^{n} w_{ij}^{\mathrm{ENC}} x_j + b_i\right) \tag{5.12}$$

where g is the activation function, b_i is the bias of the ith neuron, $\{x_j\}_{j=1}^{n}$ are the values of the sending neurons connected to the ith neuron and w_{ij} is the weight of the jth sending neuron of the ith neuron in the hidden layer.

The data available to build the network is customarily split into three parts: training data, validation data and test data. The use of these data sets in the training process will now be described. The activation function and number(s) of neurons in the hidden layer(s) are examples of hyper-parameters of the network, and the weights and biases are referred to as parameters of the network. For given hyper-parameters, the objective of training the autoencoder is to find weights that minimise the so-called loss function, which, for regression problems, is often taken as the mean squared error of the reconstruction:

$$\frac{1}{N^{\mathrm{ex}}} \sum_{j=1}^{N^{\mathrm{ex}}} ||\boldsymbol{x}_j - \hat{\boldsymbol{x}}_j||^2 = \frac{1}{N^{\mathrm{ex}}} \sum_{j=1}^{N^{\mathrm{ex}}} ||\boldsymbol{x}_j - f^{\mathrm{ae}}\left(\boldsymbol{x}_j; \boldsymbol{w}^{\mathrm{ae}}\right)||^2 \ , \tag{5.13}$$

where N^{ex} is the number of samples or examples used to train the autoencoder and the norm used is the Euclidean norm. Based on gradient descent-type methods, a back-propagation algorithm is used to optimise the weights by solving

$$\boldsymbol{w}^{\mathrm{ae}} = \arg\min_{\tilde{\boldsymbol{w}}^{\mathrm{ae}}} \frac{1}{2} \sum_{j=1}^{N^{\mathrm{ex}}} ||\boldsymbol{x}_j - f^{\mathrm{ae}}\left(\boldsymbol{x}_j; \tilde{\boldsymbol{w}}^{\mathrm{ae}}\right)||^2 \ . \tag{5.14}$$

In order to find the optimal hyper-parameters, less sophisticated approaches than back-propagation are employed, such as cross-validation [180]. This involves training a number of networks each with a different set of hyper-parameters and comparing the performance of these networks on the validation data. The best-performing network is selected and can be re-trained with both the training and validation data combined. The performance of this network is then assessed on the test data.

5.5.1 Variational Autoencoder

The variational autoencoder (VAE) is a type of generative network. The latent space of any autoencoder can be sampled and then decoded, however, for a VAE, the latent space is structured so as to produce an output that looks realistic for every sampled value. To develop this generative capability, the latent space is regularised to ensure it possesses two properties [181]: continuity (two points close in latent space will be 'similar' when decoded) and realism (any point in the latent space will be meaningful when decoded). The training process for a VAE [182] involves encoding the input as a distribution over latent space; sampling a point from latent space; and then decoding the sampled point. As usual, the loss is calculated and back-propagated to update the weights of the network. For N^{ex} training examples, the loss of the VAE is written as

$$\sum_{i=1}^{N^{\text{ex}}} -\mathbb{E}[\log p_\eta(x|y)] + \mathbb{KL}[q_\theta(y|x), p(y)] \tag{5.15}$$

where x is the input, y is the latent variable, $\mathbb{E}$ is the expectation, p and q represent the encoder and decoder respectively, with their associated weights θ and η, and $\mathbb{KL}$ is the Kullback-Leibler (KL) divergence. The first term is the negative log likelihood, which measures how closely the decoded samples match the inputs, and the second term is the regularisation term, which ensures that the network develops a well-structured latent space. Although such autoencoders, by construction, will not fit the training data perfectly, they might be able to generalise better than a standard autoencoder, being more likely to produce meaningful results for arbitrary values in the latent space. the difference between the encoder's distribution $q_\theta(z|x)$ and the Gaussian distribution $p(y)$ and regularises the distribution of the encoded inputs.

5.5.2 SVD-Autoencoder

The length of the input to the autoencoder is one factor that governs the difficulty of training an autoencoder. The longer these vectors are, the more parameters (weights) there are to be adjusted during optimisation, which increases the computational cost. To combat this, the dimensionality reduction can be split into two steps. The first step is to apply an SVD to the inputs of the autoencoder (i.e. the snapshots). A basis is obtained and the snapshots are projected onto the resulting low-dimensional space yielding reduced variables $\boldsymbol{\alpha}$ (just as is done

in POD-based ROM), see Equation (5.8). This reduces the length of the input vectors from the number of degrees of freedom of the problem (N) to the number of basis functions (P where $P \ll N$). The second step is to train a fully-connected autoencoder with the reduced variables of the snapshots (or the subset of these that have been selected for training), thereby further reducing the dimensionality.

Chapter 6

Autoencoder-based ROM

The content within this chapter is based on:

T. R. F. Phillips, C. E. Heaney, P. N. Smith, and C. C. Pain. An autoencoder-based reduced-order model for eigenvalue problems with application to neutron diffusion. *International Journal for Numerical Methods in Engineering* [10]

Chapter 2 contains methods for how the high-fidelity snapshots of this chapter were generated. Chapter 5 contains methods for Proper-Orthogonal Decomposition that were used within this chapter.

6.1 Introduction

Chapter 5 describes Proper-Orthogonal Decomposition (POD), a commonly used method for Reduced-Order Modelling [5, 176]. The POD basis functions are found by taking a Singular Value Decomposition (SVD) of the snapshots or by solving a related eigenvalue problem. Using the basis functions, a Galerkin projection can be applied to the discretised governing equations (i.e. the high-fidelity model) to produce the reduced system of equations. For HFMs whose operators can be affinely decomposed, the Galerkin projection will be applied to all the matrices in the decomposition. Finding the reduced system for an unseen parameter can be done by interpolating the resulting reduced system matrices. Reduced-order models produced by this method are referred to as projection-based reduced-order models [65].

One type of neural network ideal for dimensionality reduction is the autoencoder. A type of unsupervised (or self-supervised) feed-forward neural network, the autoencoder learns the identity map with a network architecture that includes a bottleneck. The central bottleneck layer forces the autoencoder to learn a reduced or low-dimensional data representation. The neurons in this layer determine the so-called latent space and are referred to as latent variables. These types of networks are common in image classification and identification [112, 113], and have been used with great success to fill in gaps in images [114] and to remove noise from data [115]. Several authors [117–119] have highlighted the connection between autoencoders and SVD (or PCA), namely that an autoencoder with one hidden layer and *linear* activation functions can produce basis functions than span the same space as basis functions derived from an SVD.

This connection between the autoencoder and the SVD paves the way for an autoencoder to be seen as a tool to perform *nonlinear* dimensionality reduction (by choosing nonlinear activation functions) and therefore to be used as an alternative to the SVD/PCA [112, 120]. Amongst those early to recognise the potential of the autoencoder for dimensionality reduction within a model reduction framework were Milano et al. [121], who reconstructed the near wall field from computational solutions for pressure and shear stress at the wall. The suitability of autoencoders for dimensionality reduction has been established, and several authors have since explored their use in reduced-order models, mostly non-intrusive reduced-order models. One of the strengths of the autoencoder is it is a nonlinear embedding which also means that the reduced-order system will be nonlinear and is, therefore, more complex to solve than for a POD-based ROM. For non-intrusive models, this nonlinearity does not present any additional challenge and no change in the algorithm is required, hence the greater uptake of autoencoders in NIROM-type methods. For example, Gonzalez et al. [116] and Wiewel et al. [122] both use a convolutional autoencoder to reduce the dimension of their problems and Long Short-Term Memory networks to learn the dynamics. The former demonstrates their method of interacting vortices in a 2D box and 2D lid-driven cavity flow. The interacting-vortices problem aims to demonstrate the location invariance properties of convolutional networks. The long-term statistics of the flow, as embodied by the turbulent kinetic energy, are captured much better by the autoencoder-based method than by the projection-based POD-Galerkin model. Coming from the field of computer graphics, Wiewel et al. [122] solves 3D problems with millions of spatial degrees of freedom. Their reduced-order model can produce realistic simulations

of complex dynamical systems such as sloshing waves, colliding bodies of fluid and smoke convection. Their reduced model runs approximately two orders of magnitude faster than their high-fidelity solver.

Although in many areas, adding diffusion or solving diffusion problems often leads to smooth and well-represented fields with POD basis functions. However, in other areas, such as reactor physics, one is often confronted by problems with abruptly changing fields similar to advection problems. A good example is a control rod that is partially inserted into a reactor, where one would see a near zero flux within the control rod and a much higher flux a small distance away from the control rod; see the scalar flux solutions shown in Buchan et al. [183] for instance. This similarity between advection and diffusion problems is also seen in discretisation methods such as the Self-Adjoint Angular Flux method [184], in which the first order transport equations are transformed into a series of diffusion-like equations with the application of a least squares principle and a particular weighting.

This chapter describes the first application of an autoencoder-based reduced-order model to an eigenvalue problem, producing a method that utilises the governing equations of the system with a non-linear form of dimensionality reduction. Also presented is a novel hybrid SVD-autoencoder method that combines an SVD with an autoencoder. This is useful as training a fully-connected autoencoder with a long input vector is computationally expensive. Without the SVD, the length of the input vector would be equal to the number of degrees of freedom of the problem. Applying the SVD reduces the length of the input vector to, at most, the number of snapshots. It is demonstrated that the autoencoder-based method can achieve a higher degree of accuracy at smaller numbers of latent variables than POD. ROM methods utilising autoencoders for dimensionality reduction should result in the same benefits as the POD methods listed before this, primarily a reduction in computational time of multiple orders of magnitude.

The chapter is organised as follows: Section 6.2 contains the methodology, section 6.3 contains an overview of the 1D case, section 6.4 contains an overview of the 2D case and finally, a summary of the chapter is given in section 6.5.

6.2 Methodology

POD defines a linear map between the scalar fluxes $\boldsymbol{\phi}$, and the corresponding reduced variables, $\boldsymbol{\alpha}$, see Equation (5.8). Using an autoencoder for the dimensionality reduction means that the map between these variables will, in general, be nonlinear. For eigenvalue problems, which are inherently nonlinear [185], using an autoencoder rather than POD for the dimensionality reduction, adds an additional level of non-linearity. To construct a reduced-order model, the nonlinear mapping between the reduced variables and the scalar fluxes needs to be linearised and the reduced system of equations needs to be derived. Both points are now elucidated.

Suppose that we have a known state of reduced variables $\tilde{\boldsymbol{\alpha}}$ and scalar fluxes $\tilde{\boldsymbol{\phi}}$, and wish to find the change in the scalar flux $\Delta\boldsymbol{\phi}$ due to a small change in the reduced variables $\Delta\boldsymbol{\alpha}$. We can linearise the map $\boldsymbol{C}$ between these variables using a first order Taylor series as follows:

$$\Delta\boldsymbol{\phi} = \boldsymbol{C}(\tilde{\boldsymbol{\alpha}})\,\Delta\boldsymbol{\alpha}\,, \tag{6.1}$$

$$\text{where } \Delta\boldsymbol{\phi} = \boldsymbol{\phi} - \tilde{\boldsymbol{\phi}}\,, \tag{6.2}$$

$$\Delta\boldsymbol{\alpha} = \boldsymbol{\alpha} - \tilde{\boldsymbol{\alpha}}\,, \tag{6.3}$$

$$\text{and } \boldsymbol{C}(\tilde{\boldsymbol{\alpha}}) = \left.\frac{\mathrm{d}\boldsymbol{\phi}}{\mathrm{d}\boldsymbol{\alpha}}\right|_{\tilde{\boldsymbol{\alpha}}}\,. \tag{6.4}$$

The scalar flux $\boldsymbol{\phi}$ can therefore be approximated as

$$\boldsymbol{\phi} = \boldsymbol{C}(\tilde{\boldsymbol{\alpha}})\boldsymbol{\alpha} - \boldsymbol{C}(\tilde{\boldsymbol{\alpha}})\tilde{\boldsymbol{\alpha}} + \tilde{\boldsymbol{\phi}}\,. \tag{6.5}$$

In order to calculate $\boldsymbol{C}(\tilde{\boldsymbol{\alpha}})$, perturbations are made to $\tilde{\boldsymbol{\alpha}}$ from which the entries of $\boldsymbol{C}$ can be inferred. Each of the P entries in $\tilde{\boldsymbol{\alpha}}$ are perturbed in turn by a small value, which yields a column of entries in $\boldsymbol{C}$. The kth perturbation simply adds a small number ϵ to the kth component of $\tilde{\boldsymbol{\alpha}}$, so, for example, the second perturbation is given by

$${}^{2}\Delta\boldsymbol{\alpha} = \underbrace{(0, \epsilon, 0, 0, \ldots, 0)}_{P \text{ entries}}{}^{T}\,. \tag{6.6}$$

The decoder can be used to calculate the scalar flux corresponding to the perturbed reduced

variables ${}^{k}\boldsymbol{\alpha}$:

$$ {}^{k}\boldsymbol{\phi} = f^{\mathrm{DEC}}\left({}^{k}\boldsymbol{\alpha}\right) = f^{\mathrm{DEC}}\left(\tilde{\boldsymbol{\alpha}} + {}^{k}\Delta\boldsymbol{\alpha}\right). \tag{6.7} $$

The change in the scalar flux can be calculated by subtracting the known state $\tilde{\boldsymbol{\phi}}$ from the above. The kth column of $\boldsymbol{C}$ is now known:

$$ C_{jk}(\tilde{\boldsymbol{\alpha}}) = \frac{{}^{k}\phi_j - \tilde{\phi}_j}{\epsilon} = \frac{{}^{k}\Delta\phi_j}{\epsilon} \qquad \forall j \ , \tag{6.8} $$

where ϕ_j is the jth component of the scalar flux. Once every component of the reduced variables $\boldsymbol{\alpha}$ has been perturbed, the $\boldsymbol{C}$ matrix will be fully determined. Having found $\boldsymbol{C}$, the map between the reduced variables, $\boldsymbol{\alpha}$, and the high-fidelity model variables, $\boldsymbol{\phi}$, can be calculated using Equation (6.5). In this equation, the nonlinearity is introduced by $\boldsymbol{C}$ as its values depend on the reduced variables. When using a dimensionality-reduction method such as POD, this map is linear and is independent of the reduced variables, see Equation (5.9).

With the map between the reduced and high-fidelity model variables now established, the projection-based ROM, using an autoencoder, can now be developed. When solving the high-fidelity model, an inner iteration loop solves for the scalar flux, and an outer iteration loop normalises the flux and solves for the eigenvalue. Instead of solving a reduced-order model of the generalised eigenvalue problem, directly, shown in Equation (5.10), we replace the inner iterations with a reduced-order model to approximate the flux, but use the outer iterations, unmodified, to normalise the flux and find the corresponding eigenvalue. Therefore, we look to develop a reduced-order model for the equation solved in the inner iterations of the power method, see line 3 of Algorithm 1,

$$ \boldsymbol{A}\boldsymbol{\phi} = \boldsymbol{s} \, , \tag{6.9} $$

where the right-hand side term, $\boldsymbol{s} = \lambda^{(\cdot)}\boldsymbol{B}\boldsymbol{\phi}^{(\cdot)}$, uses the latest values for the eigenvalue and scalar flux that have been calculated in the outer iterations. Substituting Equation (6.5) into Equation (6.9), rearranging and pre-multiplying the result by $\boldsymbol{C}^T$ gives the reduced system of equations for the autoencoder-based ROM:

$$ \left(\boldsymbol{C}^T(\tilde{\boldsymbol{\alpha}})\boldsymbol{A}\boldsymbol{C}(\tilde{\boldsymbol{\alpha}})\right)\Delta\boldsymbol{\alpha} = \boldsymbol{C}^T(\tilde{\boldsymbol{\alpha}})\boldsymbol{s} - \boldsymbol{C}^T(\tilde{\boldsymbol{\alpha}})\boldsymbol{A}\tilde{\boldsymbol{\phi}} \, . \tag{6.10} $$

As this system is relatively small for the examples used in this paper, it is solved with Gaussian elimination. The algorithm for the inner iterations, which solve for the scalar flux, is given

in Algorithm 4 in which the dependence of $\boldsymbol{C}$ on $\tilde{\boldsymbol{\alpha}}$ has been dropped for readability. Regularisation has been applied to the reduced-order model by adding a small number, ε, to the diagonal of $\boldsymbol{C}^T\boldsymbol{A}\boldsymbol{C}$ as, without this, there is no guarantee that this matrix will be non-singular. The regularisation will also ensure that the iterative change in reduced variables ($\Delta\boldsymbol{\alpha}$) will not become too large. Note that this is a modified power iteration, as the eigenvalue is included in the source term. For the standard power method, this is not required when determining the flux, as normalisation renders this unnecessary. However, in line 11 of the algorithm, the right-hand side of this expression has a source term and an additional term, meaning that the size of the source term is now important. Both autoencoders described in Sections 5.5 and 5.5.2 can be used in the reduced-order model framework described in this section.

We remark that in order to solve the intrinsically nonlinear eigenvalue problem, (whether from the high-fidelity model or reduced model) the power method is deployed. This method uses an inner iteration to solve for the scalar flux, and an outer iteration to normalise the flux and solve for the eigenvalue. Due to the linear map $\boldsymbol{R}$ between the reduced variables and the high-fidelity model variables, the inner iterations for the high-fidelity model and POD-based ROM solve a linear system (line 3 in Algorithm 1 and line 4 in Algorithm 3 respectively). However, for the AE-based ROM, the map between the high-fidelity model variables and the reduced variables is nonlinear, which results in an additional iteration loop (lines 8 to 19 in Algorithm 4), within which, is the linear solve (line 11).

The methods outlined in chapter 5 (POD-based ROM) and section 6.2 (autoencoder-based ROMs) are applied to two test cases. Before presenting the results, the error measures that are used to compare the methods are introduced, followed by an explanation of how the material parameters (cross-sections) are homogenised. The first test case is a 1D slab reactor and results from a POD-based ROM and an AE-based ROM are presented. Further comparisons are made between a POD-based ROM and an AE-based ROM constructed with fewer reduced variables. The second test case is a simplified 2D reactor. For this case, results are presented for ROMs based on POD, on an autoencoder and a hybrid SVD-autoencoder. The reduced-order models were developed in python using Keras [147] with the TensorFlow backend [90].

Algorithm 4 AE-based reduced-order model: inner iterations

function INNER_ITERATIONS($\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{\phi}$, λ)
  $k\_max = 100$
  $\alpha\_tol = 10^{-8}$
  $\boldsymbol{s} = \lambda \boldsymbol{B}\boldsymbol{\phi}$! *set a source with values from the outer iterations*
  $\boldsymbol{\phi}^{(0)} = \boldsymbol{\phi}$; $\lambda^{(0)} = \lambda$; $\boldsymbol{\alpha}^{(0)} = f^{\text{ENC}}(\boldsymbol{\phi}^{(0)})$! *initialise with values from the outer iterations*
  $k = 0$
  `not_converged = True`
  while `not_converged` **do**
    calculate $\boldsymbol{C}^{(k)}$ for $\boldsymbol{\alpha}^{(k)}$ using Equation (6.8)
    solve the reduced-order model:
    $\left((\boldsymbol{C}^{(k)})^T \boldsymbol{A} \boldsymbol{C}^{(k)} + \varepsilon \boldsymbol{I}\right) \Delta\boldsymbol{\alpha}^{(k+1)} = (\boldsymbol{C}^{(k)})^T \boldsymbol{s} - (\boldsymbol{C}^{(k)})^T \boldsymbol{A} \boldsymbol{\phi}^{(k)}$
    $\boldsymbol{\alpha}^{(k+1)} = \boldsymbol{\alpha}^{(k)} + \Delta\boldsymbol{\alpha}^{(k+1)}$! *update variables*
    $\Delta\boldsymbol{\phi}^{(k+1)} = \boldsymbol{C}^{(k)} \Delta\boldsymbol{\alpha}^{(k+1)}$
    $\boldsymbol{\phi}^{(k+1)} = \boldsymbol{\phi}^{(k)} + \Delta\boldsymbol{\phi}^{(k+1)}$
    if $(\Delta\boldsymbol{\alpha}^{(k+1)} < \alpha\_tol$ **or** $k = k\_max)$ **then**
      `not_converged = False`
      $\boldsymbol{\phi} \leftarrow \boldsymbol{\phi}^{(k+1)}$
    else
      $k \leftarrow k + 1$
  return $\boldsymbol{\phi}$

6.2.1 Error measures

To assess the accuracy of the results, a number of error measures are introduced. In all the expressions, the reduced-order model solutions are measured against the high-fidelity model results. First, the error committed in the dimensionality reduction step is evaluated. For the POD-based ROM, this error is incurred by projecting the high-fidelity model solutions onto the reduced space and for the AE-based ROM this error occurs during the compression and de-compression of the high-fidelity model solutions by the autoencoder. For the SVD-AE-based ROM, the error is due to both projecting the solution onto a space defined by POD basis functions, and compressing and de-compressing with an autoencoder. The normalised

maximum errors in the reconstruction of the solution are defined here using

$$e_{\max}^{\mathrm{POD}}(\boldsymbol{\phi}^{\mathrm{HFM}}) = \frac{\phi_k^{\mathrm{HFM}} - \left(\boldsymbol{R}\boldsymbol{R}^T\boldsymbol{\phi}^{\mathrm{HFM}}\right)_k}{\|\boldsymbol{\phi}^{\mathrm{HFM}}\|_\infty} \qquad \text{where } k = \underset{i\,\in\,\text{cells}}{\operatorname{argmax}} \mid \phi_i^{\mathrm{HFM}} - \left(\boldsymbol{R}\boldsymbol{R}^T\boldsymbol{\phi}^{\mathrm{HFM}}\right)_i \mid , \tag{6.11}$$

$$e_{\max}^{\mathrm{AE}}(\boldsymbol{\phi}^{\mathrm{HFM}}) = \frac{\phi_k^{\mathrm{HFM}} - \left(f^{\mathrm{AE}}(\boldsymbol{\phi}^{\mathrm{HFM}})\right)_k}{\|\boldsymbol{\phi}^{\mathrm{HFM}}\|_\infty} \qquad \text{where } k = \underset{i\,\in\,\text{cells}}{\operatorname{argmax}} \mid \phi_i^{\mathrm{HFM}} - \left(f^{\mathrm{AE}}(\boldsymbol{\phi}^{\mathrm{HFM}})\right)_i \mid , \tag{6.12}$$

$$e_{\max}^{\text{SVD-AE}}(\boldsymbol{\phi}^{\mathrm{HFM}}) = \frac{\phi_k^{\mathrm{HFM}} - \left(\boldsymbol{R}f^{\mathrm{AE}}\left(\boldsymbol{R}^T\boldsymbol{\phi}^{\mathrm{HFM}}\right)\right)_k}{\|\boldsymbol{\phi}^{\mathrm{HFM}}\|_\infty} \qquad \text{where } k = \underset{i\,\in\,\text{cells}}{\operatorname{argmax}} \mid \phi_i^{\mathrm{HFM}} - \left(\boldsymbol{R}f^{\mathrm{AE}}\left(\boldsymbol{R}^T\boldsymbol{\phi}^{\mathrm{HFM}}\right)\right)_i \mid , \tag{6.13}$$

for the POD-based ROM, the AE-based ROM and SVD-AE-based ROM respectively, where $\|\cdot\|_\infty$ represents the maximum norm, i and k represent cell indices, and the superscript 'HFM' indicates the flux is associated with the high-fidelity model.

Second, the error in the scalar flux of the reduced-order models is calculated. The normalised maximum error in the flux profile is determined by

$$e_{\max}(\boldsymbol{\phi}^{\mathrm{ROM}}) = \frac{\phi_j^{\mathrm{HFM}} - \phi_j^{\mathrm{ROM}}}{\|\boldsymbol{\phi}^{\mathrm{HFM}}\|_\infty} \quad \text{where } j = \underset{i\,\in\,\text{cells}}{\operatorname{argmax}} \mid \phi_i^{\mathrm{HFM}} - \phi_i^{\mathrm{ROM}} \mid , \tag{6.14}$$

in which the superscript 'ROM' indicates the solution is from a reduced-order model (either POD-based or autoencoder-based), and i and j represent cell indices.

Calculating the error in k_{eff} is done by comparing the value calculated by the high-fidelity model with that of the reduced-order model:

$$e(k_{\mathrm{eff}}^{\mathrm{ROM}}) = k_{\mathrm{eff}}^{\mathrm{HFM}} - k_{\mathrm{eff}}^{\mathrm{ROM}}. \tag{6.15}$$

As k_{eff} is usually close to one, no normalisation is applied.

Finally, an average maximum error is defined, where, for N sets of material parameters resulting in N solutions of the high-fidelity model and N solutions of one of the reduced-order models:

$$\bar{e}_{\max}(\boldsymbol{\phi}^{\mathrm{ROM}}) = \frac{1}{N}\sum_{l=1}^{N} \frac{\left|\, \phi_j^{\mathrm{HFM}}(\boldsymbol{\mu}_l) - \phi_j^{\mathrm{ROM}}(\boldsymbol{\mu}_l) \,\right|}{\|\phi^{\mathrm{HFM}}(\boldsymbol{\mu}_l)\|_\infty} \tag{6.16}$$

where $\boldsymbol{\mu}_l$ refers to the parameters (i.e. the macroscopic cross-sections) used for the lth problem. Until this point, dependence of the solution on the material parameters has not been explicitly indicated (for readability), however here it is required. A similar error measure can also be used to find the average maximum error in k_{eff}:

$$\bar{e}(k_{\text{eff}}^{\text{ROM}}) = \frac{1}{N}\sum_{l=1}^{N} \left| \, k_{\text{eff}}^{\text{HFM}}(\boldsymbol{\mu}_l) - k_{\text{eff}}^{\text{ROM}}(\boldsymbol{\mu}_l) \, \right| \ , \tag{6.17}$$

and the average maximum error in the reconstruction:

$$\bar{e}_{\text{max}}^{\text{DR}}(\boldsymbol{\phi}^{\text{HFM}}) = \frac{1}{N}\sum_{l=1}^{N} \left| \, e_{\text{max}}^{\text{DR}}(\boldsymbol{\phi}^{\text{HFM}}) \, \right| \ , \tag{6.18}$$

where DR represents one of the three dimensionality reduction methods used: POD, AE or SVD-AE.

6.2.2 Homogenising the Material Parameters

In the test cases that follow, there are fuel regions and also control-rod regions, within which the control rods can be fully inserted, completely withdrawn or partially inserted. In the latter case, the cross-sections will be calculated by combining the cross-sections for fuel and control rod. Control-rod regions are assumed to have a uniform distribution of the averaged properties of the mixture within the system based on a mixing coefficient for each region. This can be written as follows

$$\Sigma_{\text{hom}}^{a} = r\Sigma_{\text{cr}}^{a} + (1-r)\Sigma_{\text{fuel}}^{a}, \tag{6.19}$$

$$\Sigma_{\text{hom}}^{s} = r\Sigma_{\text{cr}}^{s} + (1-r)\Sigma_{\text{fuel}}^{s}, \tag{6.20}$$

where the mixing coefficient, r, for a given region, lies between 0 and 1, and the subscripts 'hom', 'cr' and 'fuel' indicate a homogenised cross-section, and cross-sections of the control rod and the fuel respectively. If r were chosen to be proportional to the amount of control rod present in a control-rod region, the high absorption of the control rods would dominate the homogenised cross-section. In order to address this, the reciprocal of the absorption cross-sections are averaged:

$$\frac{1}{\Sigma_{\text{hom}}^{a}} = \frac{z}{\Sigma_{\text{cr}}^{a}} + \frac{1-z}{\Sigma_{\text{fuel}}^{a}} \qquad \text{where } z \in [0,1]\,. \tag{6.21}$$

By combining Equations (6.19) and (6.21) it can be shown that the mixing coefficient is given by

$$r = \frac{z\Sigma^a_{\text{fuel}}}{z\Sigma^a_{\text{fuel}} + (1-z)\Sigma^a_{\text{cr}}}\,. \tag{6.22}$$

In this way, the desired amount of control rod in a region can be chosen by setting z, converting this into a value for r by using Equation (6.22), and then calculating the homogenised absorption and scattering cross-sections from Equations (6.19) and (6.20).

6.3 1D Slab Reactor

The length of the slab reactor is 10 cm, and it consists of fuel and two control-rod regions labelled as 1 and 2 in Figure 6.1. Region 1 is located between 2.2 cm and 2.5 cm on the x-axis, region 2, between 7.5 cm and 7.8 cm. This test case was used by Buchan et al. [186]

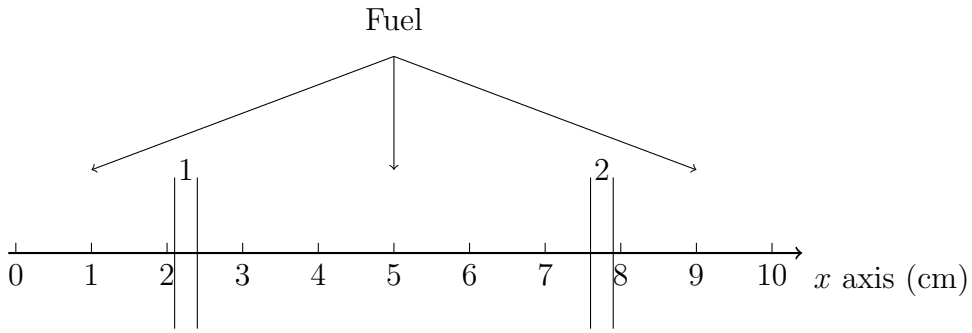


Figure 6.1: Geometry of a 1D slab reactor consisting of fuel and two control-rod regions (regions 1 and 2).

Table 6.1 contains the macroscopic cross-sections used for the control rods and the fuel. The control rods can be fully inserted, completely withdrawn or partially inserted into regions 1 and 2. In the latter case, the cross-sections for these regions will be calculated by combining the cross-sections for fuel and control rod through mixing coefficients as described in the previous section. The amount of control rod in region 1 will, in general, be different from that of region 2, henceforth, z_1 and r_1 refer to region 1, and z_2 and r_2 refer to region 2. 144 cells were used in the control-volume discretisation although 44 of these were used to enforce the boundary conditions, resulting in a system with 100 degrees of freedom.

To generate the data necessary to build the reduced-order models, 200 problems were solved using the high-fidelity model with different control-rod settings. This was done by choosing values for z_1 and z_2 at random from the interval $[0, 1]$ and using Equations (6.22), (6.19)

	Σ^a	Σ^s	Σ^f
Fuel	0.45	2	0.5
Control Rod	0.9	2	0

Table 6.1: Macroscopic cross-sections for the 1D slab reactor (units cm^{-1}).

and (6.20) to calculate the corresponding cross-sections. The control-volume discretisation along with the cross-sections determine the matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ in Equation (2.12). Given an initial guess for the scalar flux, the system in (2.12) is solved with the power method as described in Section 2.3 yielding 200 solutions of the high-fidelity model. One hundred of the high-fidelity model solutions were treated as snapshots and were used to generate basis functions (either by applying POD or by training an autoencoder). These snapshots are referred to as 'seen' data (in the context of ROM) or as training data (in the context of neural networks). The remaining 100 solutions were used in the online stage to test how good the models were at predicting solutions for unseen data (i.e. parameter sets whose corresponding solutions were not included in the snapshots). This data is referred to as unseen data (in the context of ROM) and as test data (in the context of neural networks). Two reduced-order models are compared: a POD-based ROM and an autoencoder-based ROM. The models have 10 reduced variables (10 POD coefficients for the POD-based model and 10 latent variables for the autoencoder-based model). Following this, a POD-based model and an autoencoder-based model are developed with just two reduced variables.

6.3.1 POD-based Reduced-Order Model

The first method to be tested is a POD Galerkin method, which uses POD and the SVD to determine basis functions, projects the discretised governing equations onto the subspace defined by the POD basis functions, and results in a generalised eigenvalue problem of reduced dimension, Equation (5.10), to be solved in the online stage for a given set of macroscopic cross-sections. This method is described in Section 5.4. The 200 sets of parameters that generated the high-fidelity model results are used again to obtain the matrices $\boldsymbol{A}$ and $\boldsymbol{B}$. The system of equations describing the POD-based ROM, given in (5.10), is solved for each problem using the power method described in Algorithms 2 and 3, and Section 5.4 using numpy's Gaussian elimination function to solve for the reduced variables. Half of the results have been seen by the model and half have not been seen.

6.3.2 Autoencoder-based Reduced-Order Model

In this method the dimensionality reduction (or compression) is performed by an autoencoder instead of an SVD. Making this change requires an additional iteration loop when solving the reduced-order model as explained in Section 6.2. For the online prediction, an initial flux guess is compressed using the encoder. Following this, the $\boldsymbol{C}$ matrix is formed by using Equation (6.4). The system in Equation (6.8) is then solved, the flux is decoded and passed to the outer iterations, and the whole process is repeated until k_{eff} converges. In order to compare with results from the previous section, the autoencoder compresses to 10 latent variables.

Autoencoders with different architectures were trained in order to optimise the hyper-parameters. Once trained, the networks were evaluated on the validation data and the network with the lowest loss (or mean square error in this case) was assumed to have the optimal hyper-parameters. From 4 to 16 layers were tested with up to 100 neurons in each layer. Their effect was less significant on model accuracy than some of the other hyper-parameters, although smaller networks were seen to outperform larger networks. It was found that the exponential linear unit (ELU) [187] was the best performing activation function, giving better results than the Rectified Linear Unit (ReLU), the Scaled Exponential Linear Unit (SELU) and the Hyperbolic tangent activation function (tanh). Batch sizes of 10, 25 and 100 were also compared during training and it was found that the smallest batch size resulted in the lowest validation loss. The initial learning rate of the optimiser was set at 10^{-3} and the minimum value this can reduce to on stagnation of the loss was varied from 10^{-4} to 10^{-7}.

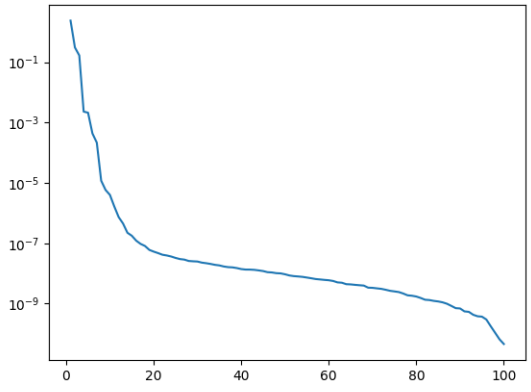
The best performing architecture comprised 4 fully-connected layers with the following number of neurons in each layer:

$$100 \rightarrow 32 \rightarrow 10 \rightarrow 32 \rightarrow 100 \ .$$

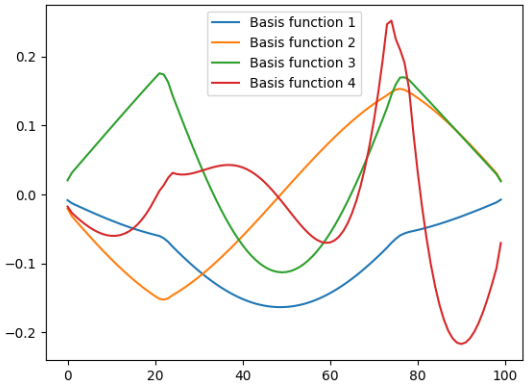
The activation function for every layer was the exponential linear unit (ELU); the optimiser used was 'Nadam' [188] with an initial learning rate of 10^{-3} and a minimum learning rate of 10^{-6}; and the loss function was defined to be the mean squared reconstruction error. The snapshots were scaled between 0 and 1. The network was trained over 20,000 epochs using mini-batch gradient descent with a batch size of 10.

6.3.3 Comparison of POD and AE as compression methods

For the POD-based ROM, an SVD is applied to the 100 snapshots and from a possible 100 basis functions, 10 are retained, which corresponds to capturing 99.999% of the information contained in the snapshots, see Figure 6.2a. As can be seen from this figure, the magnitude of the singular values decreases rapidly from the 1st to the 20th singular value (by about 7 orders of magnitude) before levelling off, and it can be expected from this that a POD model with a sufficient number of basis functions will perform well. The first four basis functions can be seen in Figure 6.2b. Notice that the fourth basis function has more detail or structure than the other basis functions, following the general trend that the higher-order basis functions tend to have more structure.



(a) Singular value against singular value index.

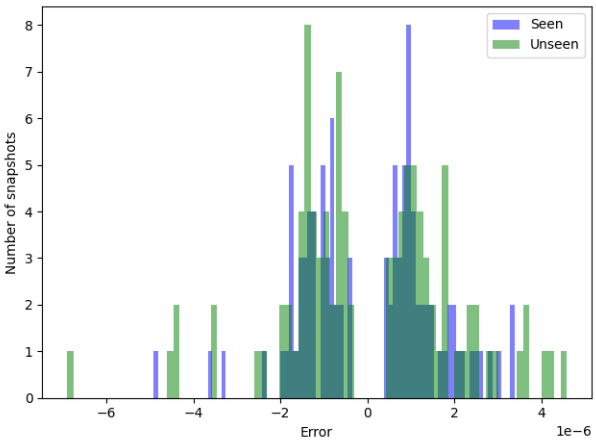


(b) First four basis functions.

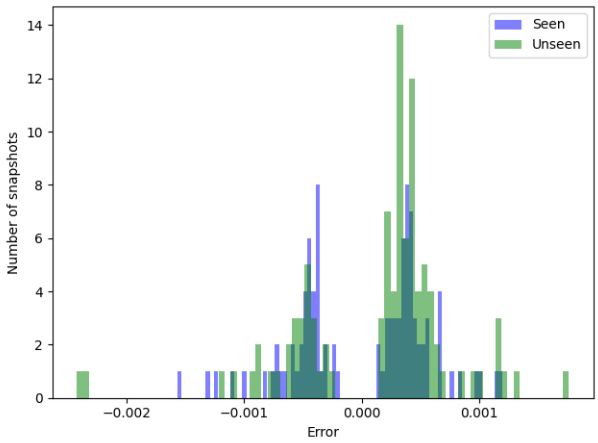
Figure 6.2: Singular values and basis functions of snapshots from the 1D slab reactor for the POD-based ROM

.

Figure 6.3a shows the reconstruction or compression error, $e_{\max}^{\text{POD}}(\boldsymbol{\phi}^{\text{HFM}})$, resulting from projecting high-fidelity model solutions onto the reduced space for both the seen data and the unseen data, see Equation (6.11). Both seen and unseen data sets yield a similar range of errors and the errors themselves are low: $\mathcal{O}(10^{-6})$. The autoencoder was trained on the same data to which the SVD was applied, i.e. the 100 snapshots. The unseen high-fidelity model solutions were used as test data, to assess the prediction power of the autoencoder. Figure 6.3b shows the error (as defined by Equation (6.12)) between each HFM solution and the corresponding output of the autoencoder for seen and unseen data sets. The autoencoder performs well, achieving, at worst,



(a) Reconstruction error from the SVD for the 1D test case. Seen data (snapshots) in blue, unseen data in green.



(b) Compression error from the autoencoder for the 1D test case. Training data (snapshots) in blue, test data in green.

Figure 6.3: Errors for the two methods of compression used here for the 1D test case, compressing from 100 variables to 10.

an absolute error of just over 0.2% in the flux, with almost all of the absolute values of the error less than 0.15% . Although the seen and unseen data sets have a similar range of errors, it should be noted that these errors are much larger than those of the SVD (see Figure 6.3a). Shown in Figure 6.4 is a box and whisker plot of the values of the 10 latent variables over the training data set. It can be observed that 5 of the latent variables show very little variation across the seen data set.

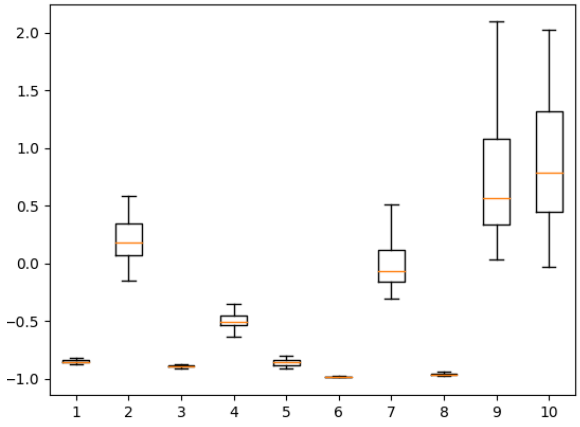
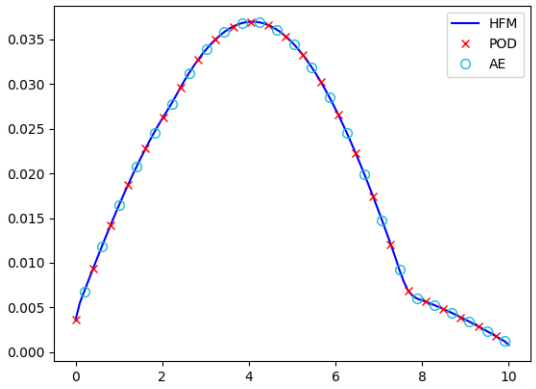


Figure 6.4: Box and whisker plot of the 10 latent variables generated from the autoencoder over the training data for the 1D test case. The orange line shows the median value, the box shows the upper and lower quartiles, and the whiskers indicate the minimum and maximum values.

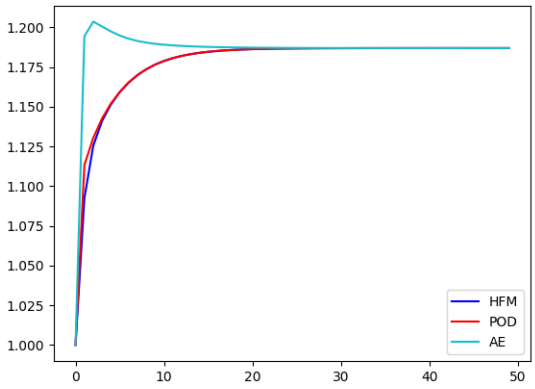
6.3.4 Comparison of POD-based and AE-based ROMs

Scalar fluxes and k_{eff} values from the POD-based and AE-based reduced-order models are now compared, where both ROMs have 10 reduced variables. Figure 6.5 shows the flux profile and the convergence of k_{eff} for the mixing coefficients $r_1 = 0.957$ and $r_2 = 0.115$. The high-fidelity model solution corresponding to these values was included in the snapshots, so this is a seen solution. Figure 6.5a shows that the flux profiles of the POD-based ROM and AE-based ROM very agree well with the high-fidelity model solution. Both ROMs also converge to the k_{eff} predicted by the high-fidelity model as shown in Figure 6.5b. The rates of convergence are similar, however the AE-based ROM converges from above, whereas the high-fidelity model and POD-based ROM converge from below.

Figure 6.6 shows the flux profile and the convergence of k_{eff} for the material parameters $r_1 = 0.458$ and $r_2 = 0.932$. The solution corresponding to these values has not been seen by the reduced-order models (i.e. is unseen). From Figure 6.6a, it can be seen that the flux profiles of both POD-based and AE-based ROMs demonstrate excellent agreement with the high-fidelity model. The convergence of k_{eff} for both ROMs is also in good agreement with the value predicted by the high-fidelity model, see Figure 6.6b, and the rates of convergence are also similar.



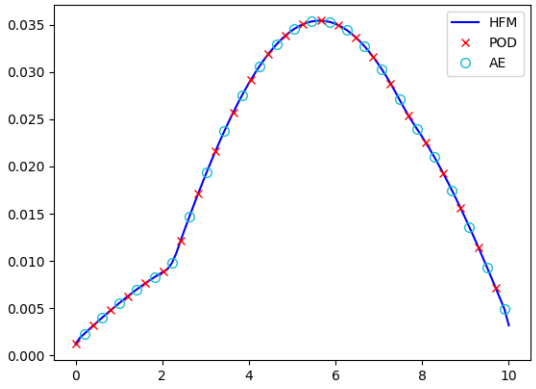
(a) Scalar flux (neutrons $\text{cm}^{-2}\,\text{s}^{-1}$) vs x (cm), HFM solution in blue, POD-based ROM in red crosses and AE-based ROM in light blue circles.



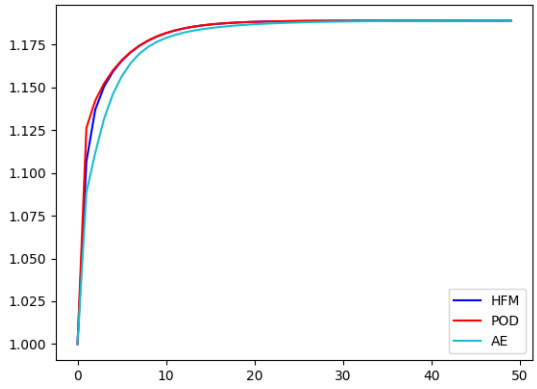
(b) k_{eff} vs iteration number, convergence of the POD-based ROM (red) and and AE-based ROM (light blue) to the converged HFM value (blue).

Figure 6.5: Scalar flux and convergence of k_{eff} for a seen problem with material parameters (cross-sections) determined by $r_1 = 0.957$ and $r_2 = 0.115$, comparing the POD-based and AE-based ROMs with the high-fidelity model. Compression from 100 variables to 10.

To further analyze the results, the 200 solutions from both the seen and unseen data sets for the



(a) Scalar flux (neutrons $\mathrm{cm}^{-2}\,\mathrm{s}^{-1}$) vs x (cm), HFM solution in blue, POD-based ROM in red crosses and AE-based ROM in light blue circles.

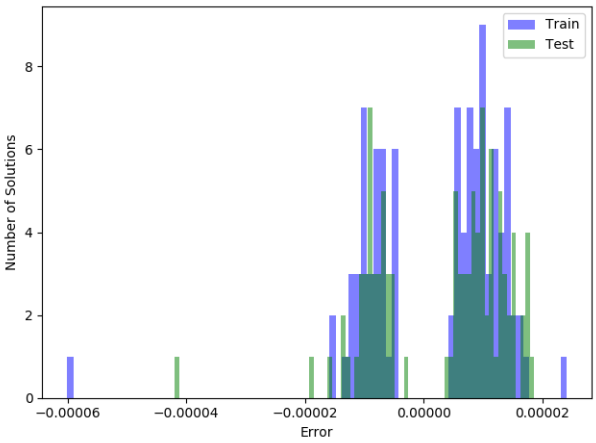


(b) k_{eff} vs iteration number, convergence of the POD-based ROM (red) and and AE-based ROM (light blue) to the converged HFM value (blue).

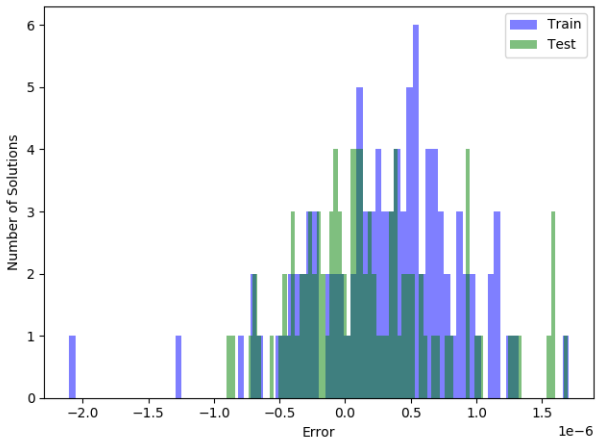
Figure 6.6: Scalar flux and convergence of k_{eff} for a unseen problem with material parameters (cross-sections) determined by$r_1 = 0.458$ and $r_2 = 0.932$, comparing the POD-based and AE-based ROMs with the high-fidelity model. Compression from 100 variables to 10.

POD-based and AE-based ROMs were compared with solutions from the high-fidelity model using the error measures for the scalar flux and k_{eff} as given by Equations (6.14) and (6.15) respectively. Figure 6.7a shows a histogram of the error in the scalar flux profile and Figure 6.7b shows a histogram of the error in k_{eff}. Excluding outliers, most of the flux solutions have an absolute error of 0.002% or less, and the absolute error in the k_{eff} values is less than 1.5×10^{-6}. The plots demonstrate both the accuracy of the POD-based model (for seen data) and its ability to predict (for unseen parameters). The ranges of values of the errors for both the seen and unseen parameters are similar. Histograms were also produced to study the error of the AE-based ROM. Figure 6.8a shows the error in flux and Figure 6.8b shows the error in k_{eff} based on the error measures defined in Equations (6.14) and (6.15). Excluding outliers, most of the flux solutions have an absolute error of 0.2% or less, and the absolute error in the k_{eff} values is less than 10^{-4}. These values are higher than for the POD-based reduced-order model, and there is also a larger spread of errors for the AE-based model than for the POD-based method, although the ranges of the values of the errors are also similar for both seen and unseen data sets.

Table 6.2 displays the average maximum errors in the compression, scalar flux and k_{eff} for POD-based and AE-based ROMs as defined in Equations (6.16), (6.17) and (6.18). The reconstruction or compression error (for one solution) is given by Equation (6.11) for POD and Equation (6.12)

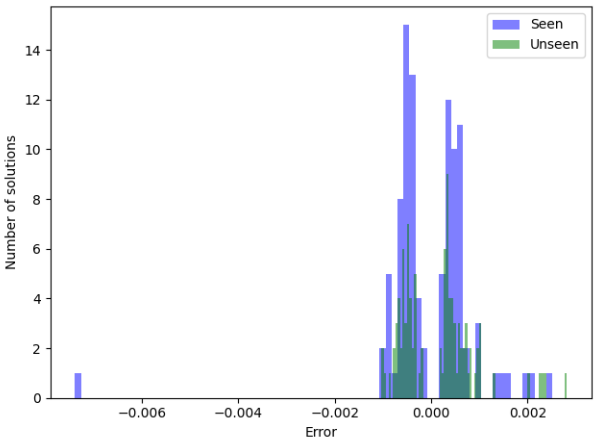


(a) Histogram of the error in the flux profile.

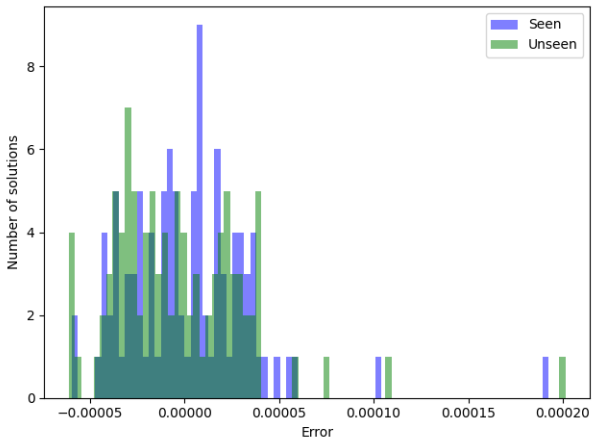


(b) Histogram of the error in k_{eff} (scale of x-axis , 10^{-6}).

Figure 6.7: Errors in the seen (blue) and unseen (green) results from the POD-based reduced-order model for the 1D test case compressing from 100 variables to 10.



(a) Histogram of the error in the flux profile.



(b) Histogram of the error in k_{eff}.

Figure 6.8: Errors in the seen and unseen results from the autoencoder-based reduced-order model for the 1D test case compressing from 100 variables to 10.

for the autoencoder. The error in the scalar flux solution will contain this error and other numerical errors made when solving the reduced generalised eigenvalue problem. Therefore we expect that the compression error would be lower than the flux error, borne out by the results shown in Table 6.2. Also noteworthy is the similarity between the range of errors for the seen and unseen data. This may mean that the snapshots represent well the behaviour that was tested by the 100 unseen problems. For k_{eff}, the error for the unseen results of the POD-based ROM is very slightly lower that than of the seen results. This is surprising but could be explained by several outliers and the averaging of the maximum errors.

	POD-based results			Autoencoder-based results		
		Seen	Unseen		Seen	Unseen
Compression Error	$\bar{e}^{\mathrm{POD}}_{\mathrm{max}}(\boldsymbol{\phi}^{\mathrm{HFM}})$	1.4344×10^{-6}	1.7004×10^{-6}	$\bar{e}^{\mathrm{AE}}_{\mathrm{max}}(\boldsymbol{\phi}^{\mathrm{HFM}})$	4.9554×10^{-4}	5.4829×10^{-4}
Flux Error	$\bar{e}_{\mathrm{max}}(\boldsymbol{\phi}^{\mathrm{POD}})$	1.0270×10^{-5}	1.0401×10^{-5}	$\bar{e}_{\mathrm{max}}(\boldsymbol{\phi}^{\mathrm{AE}})$	6.5843×10^{-4}	6.3030×10^{-4}
k_{eff} Error	$\bar{e}(k^{\mathrm{POD}}_{\mathrm{eff}})$	5.2956×10^{-7}	4.4348×10^{-7}	$\bar{e}(k^{\mathrm{AE}}_{\mathrm{eff}})$	2.4642×10^{-5}	2.7540×10^{-5}

Table 6.2: Average maximum errors for seen and unseen data using the POD-based reduced-order model and the autoencoder-based reduced-order model for the 1D test case compressing from 100 variables to 10.

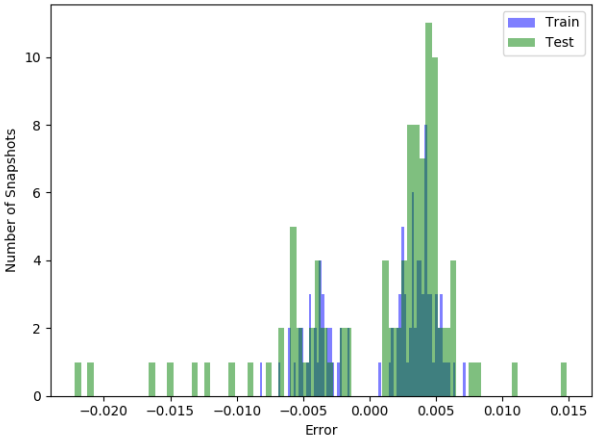
6.3.5 Variational autoencoder reduced-order model

In this section we report results of a reduced-order model which uses a variational autoencoder (VAE) to perform the dimensionality reduction. The structure of the variational autoencoder used here is identical to that used in Section 6.3.2 but with the modifications made to the central layer and the loss function as described in Section 5.5.1. These modifications mean that, in the encoder, two layers take the input from the layer with 16 neurons (or variables) and both of these compress down to 10 neurons. Sampling is applied to these two layers to form a single layer of 10 neurons and this is fed into the decoder. The reduced-order model based on this variational autoencoder is referred to as the VAE-based ROM.

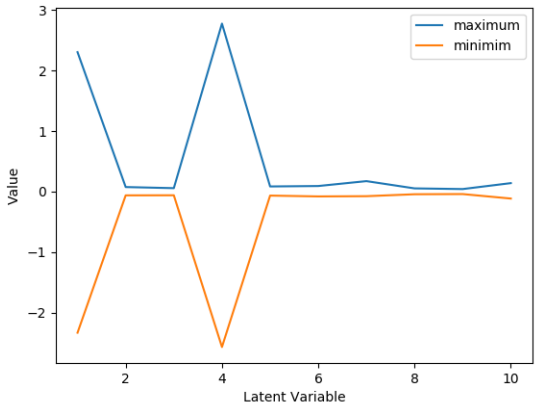
The training of the variational autoencoder and the method of prediction is otherwise identical to the previous section. Figure 6.9a shows the error between the output of the variational autoencoder and its input for both the data on which it was trained (the 100 snapshots) and the test data (the 100 unseen high-fidelity model solutions), see Equation (6.12). The range of absolute errors for the training data lies within a narrower interval than those of the test data. It can be seen from Figure 6.9b that two of the latent variables hold almost all of the information and that the remaining latent variables have values close to zero for the entire range of the training data. The VAE has compressed the data into two variables, which, as shown later, can be associated directly with the insertion and removal of the two control rods.

It can be seen from figure 6.9b that generally only two of the latent variables hold any significant meaning.

Both Figure 6.10 and Figure 6.11 show the flux profile and the convergence of k_{eff} for one seen problem and one unseen problem respectively. For both these problems, the VAE-based



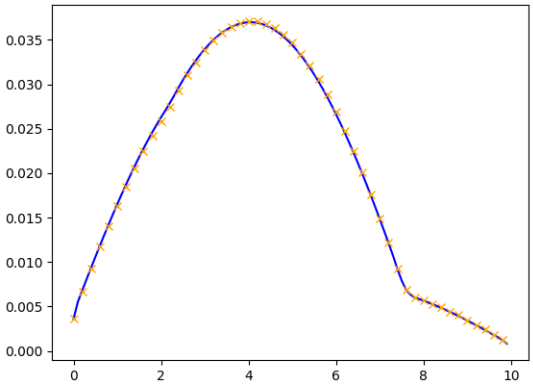
(a) Error in the flux profile output by the variational autoencoder.



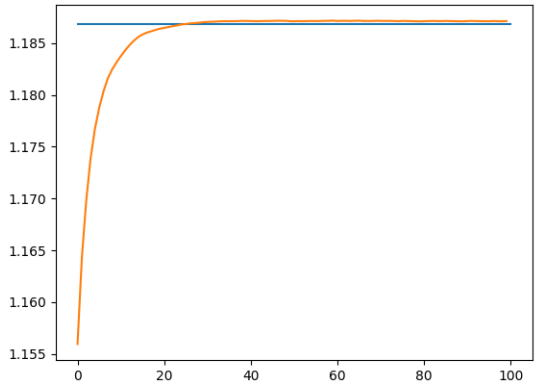
(b) Minimum and maximum values of the 10 latent variables generated by the VAE for the training data.

Figure 6.9: Variational autoencoder trained on the 1D slab reactor test case.

reduced-order model gives solutions that are very close to those of the high-fidelity model.



(a) Scalar flux (neutrons $\mathrm{cm}^{-2}\,\mathrm{s}^{-1}$) vs x (cm). HFM solution in blue, VAE-based ROM in red crosses.

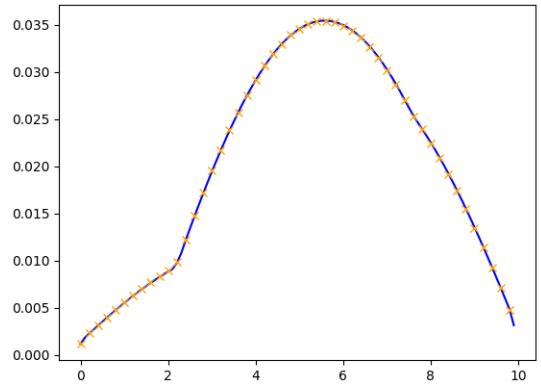


(b) k_{eff} vs iteration number, convergence of the VAE-based ROM (in red) to the converged HFM value (in blue).

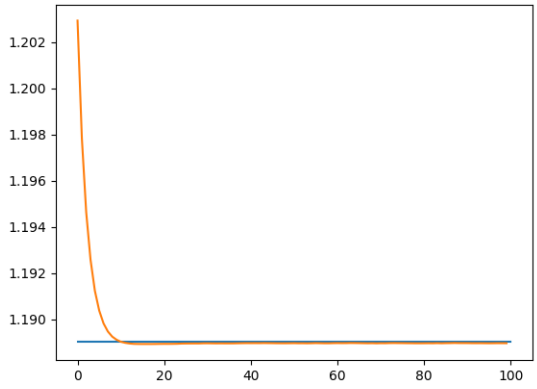
Figure 6.10: Scalar flux and k_{eff} convergence for a seen problem with material parameters $r_1 = 0.957$ and $r_2 = 0.115$ using the high-fidelity model and the VAE-based reduced-order model.

Figure 6.13 presents histograms of the errors in flux and k_{eff} based on the error measures defined in Equations (6.14) and (6.15). From these, it can be seen that the error resulting from using the VAE-based ROM is similar for both the training data and the test data.

Table 6.3 shows the average maximum error (see Equation (6.16)) for the variational autoencoder-based reduced-order model. Although the average maximum errors of the AE-based ROM and

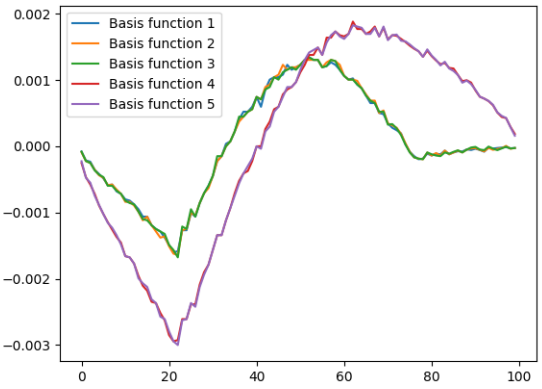


(a) Scalar flux (neutrons $\mathrm{cm}^{-2}\,\mathrm{s}^{-1}$) vs x (cm). HFM solution in blue, VAE-based ROM in red crosses.

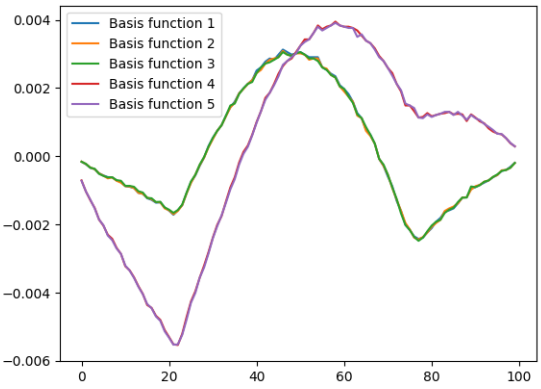


(b) k_{eff} vs iteration number, convergence of the VAE-based ROM (in red) to the converged HFM value (in blue).

Figure 6.11: Scalar flux and k_{eff} convergence for an unseen problem with material parameters $r_1 = 0.458$ and $r_2 = 0.932$ using the high-fidelity model and the VAE-based reduced-order model.



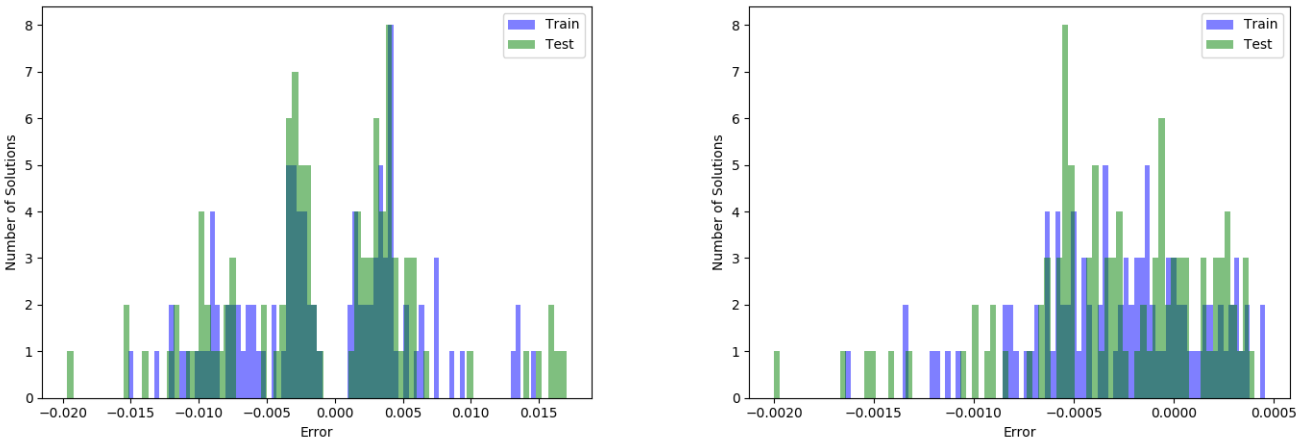
(a) Basis functions for an seen problem with material parameters $r_1 = 0.957$ and $r_2 = 0.115$ produced by the VAE-based reduced-order model at final iteration.



(b) Basis functions for an unseen problem with material parameters $r_1 = 0.458$ and $r_2 = 0.932$ produced by the VAE-based reduced-order model at final iteration.

Figure 6.12: First five basis functions produced at the final iteration by the variational autoencoder.

the VAE-based ROM are very close (see Tables 6.2 and 6.3), the latter has a larger range of errors for k_{eff}. Comparing Figures 6.8 and 6.13 for k_{eff}, it can be seen that for the AE-based ROM, the errors lie in the range $[-1 \times 10^{-3},\ 0.2 \times 10^{-3}]$, and for the VAE-based ROM, the errors lie in double this range $[-2 \times 10^{-3}, 0.5 \times 10^{-3}]$. Most noticeable, for both the flux errors and the k_{eff} errors, is that the standard deviation of the errors for the AE-based ROM is less than that of the VAE-based ROM, so, in this sense the AE-based ROM gives better results.



(a) Error in flux profile.

(b) Error in k_{eff}.

Figure 6.13: Errors for training and test data sets using the VAE-based ROM trained on 1D train dataset.

		Training data	Test data
Compression Error	$e_{\max}^{\text{VAE}}(\boldsymbol{\phi}^{\text{HFM}})$	3.8661×10^{-3}	5.1666×10^{-3}
Flux Error	$e_{\max}(\boldsymbol{\phi}^{\text{VAE}})$	5.3710×10^{-3}	5.7191×10^{-3}
k_{eff} Error	k_{eff}	4.0076×10^{-4}	4.2545×10^{-4}

Table 6.3: Average maximum errors for the training and test data sets using the VAE-based ROM to reduce from 100 variables to 10 variables.

One of the properties unique to a variational autoencoder is its ability to compress down to the smallest possible number of variables. The KL divergence term included in the loss function uses information theory to minimise the number of latent variables used to capture the behaviour of the system. From Figure 6.9b, it can be seen for this problem, reducing 100 variables down to 10, that just two variables are required to represent the system. This means that the entire set of snapshots can be approximated by varying these two variables. Figure 6.14 shows the flux profiles output by the decoder when the two latent variables are varied from their minimum to maximum values. These two variables independently dictate the control-rod height for the rods in region 1 and region 2 respectively. It can be seen that as the variable plotted on the x axis is increased, the left-hand control rod is withdrawn and as the variable plotted on the y-axis is increased, the right-hand control rod is withdrawn.

The next section compares a POD-based ROM with two POD coefficients and an AE-based ROM with two latent variables.

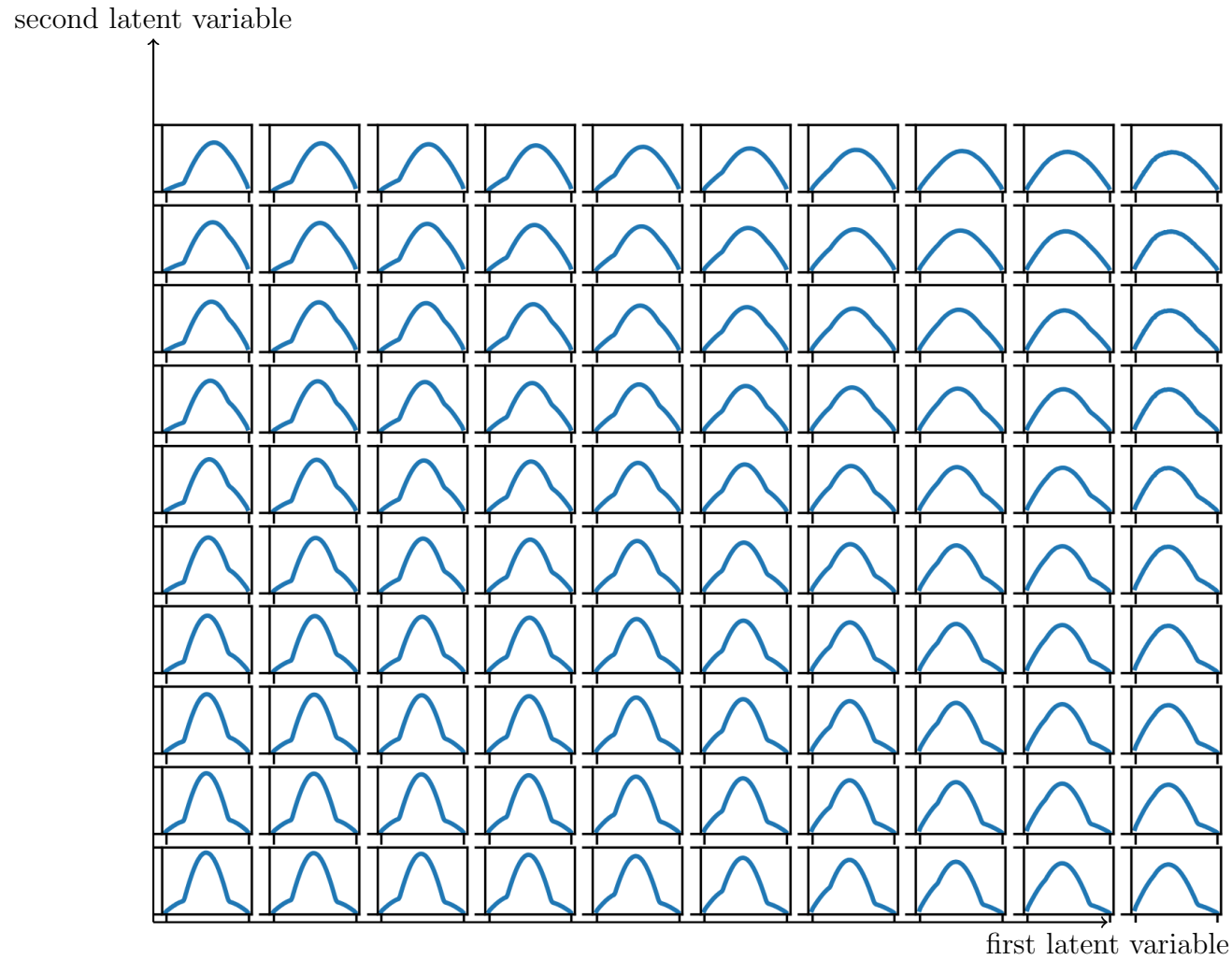


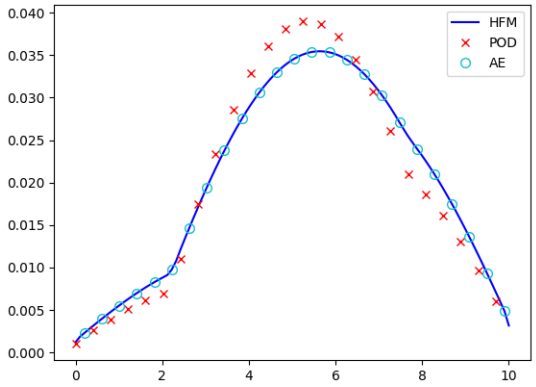
Figure 6.14: Effect on flux profile of varying two of the latent variables from their minimum to maximum values for a variational autoencoder trained on the 100 snapshots. Increasing the first latent variable removes the left-hand control rod and increasing the second variable removes the right-hand control rod.

For this test case, with two control-rod regions, one could postulate that having a low-dimensional space of dimension 2 should be enough to describe the behaviour of the system. The next section compares a projection-based ROM with two basis functions and an autoencder-based ROM with two latent variables.

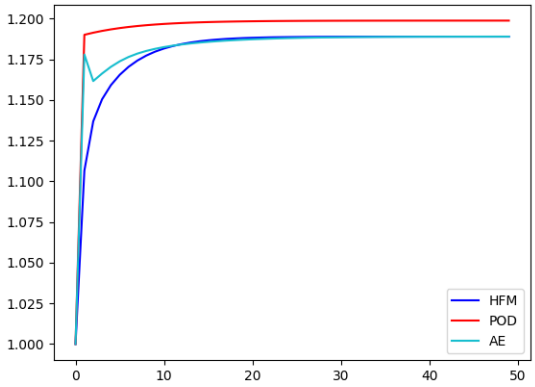
6.3.6 Reduction to two variables

The results of the variational autoencoder in Section 6.3.5 have indicated that two latent variables can be used to capture almost all the information contained in the training data. In order to see how well the models perform when reducing to a low-dimensional space of dimension 2, a POD-based ROM and an AE-based ROM were constructed using just two POD

basis functions for POD, and two latent variables for the autoencoder. Figure 6.15 shows the flux profile and convergence of k_{eff} for both POD-based and AE-based ROMs compared to the high-fidelity model for an unseen set of parameters, $r_1 = 0.458$ and $r_2 = 0.932$. Here it can be seen clearly that the AE-based ROM agrees more closely with the high-fidelity model than the POD-based ROM. Also, the k_{eff} value of the AE-based ROM converges to the value predicted by the high-fidelity model, whereas the POD-based model does not converge to this value.



(a) Scalar flux (neutrons $\text{cm}^{-2}\,\text{s}^{-1}$) vs x (cm), HFM solution in blue, POD-based ROM in red crosses and AE-based ROM in light blue circles.



(b) k_{eff} vs iteration number, convergence of the POD-based ROM (red) and and AE-based ROM (light blue) to the converged HFM value (blue).

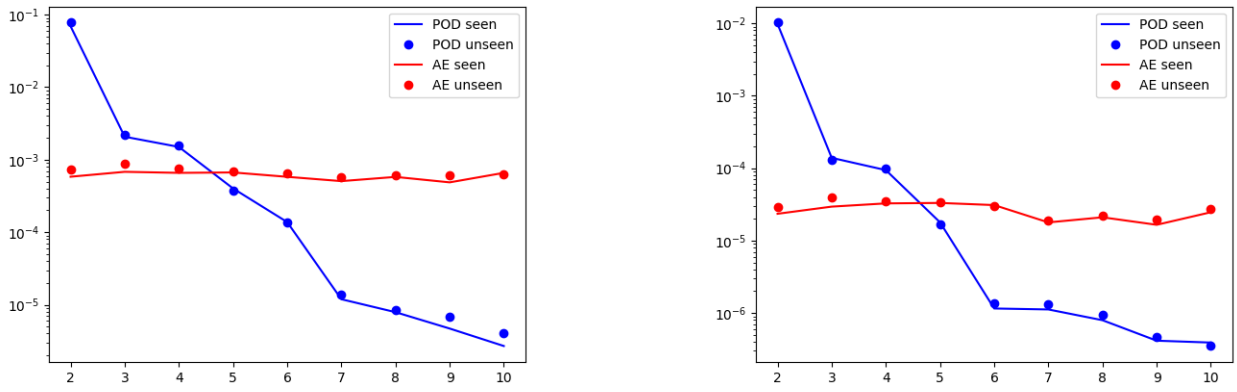
Figure 6.15: Scalar flux and convergence of k_{eff} for an unseen problem with material parameters (cross-sections) determined by $r_1 = 0.458$ and $r_2 = 0.932$, comparing the POD-based and AE-based ROMs with the high-fidelity model. Compression from 100 variables to 2.

Table 6.4 shows the errors in flux and k_{eff} when the system is compressed from 100 variables to two variables. For this extreme case, it can be seen that the autoencoder performs better than POD, with flux errors that are two orders of magnitude lower than those for POD and k_{eff} errors that are more than two orders of magnitude lower than for POD. This suggests that the autoencoder can capture more of the features than POD with a given number of basis functions/latent variables.

To investigate at which point the AE-based ROM becomes more accurate than the POD-based ROM, a number of ROMs were constructed with reduced space dimensions of 2, 3 and so on, up to 10. To generate the AE-based results, several networks were trained for each level of compression with various number of layers and neurons in each layer. All other parameters in training and the architecture were the same as for the autoencoder in Section 6.3.2. The network with the lowest validation loss was selected for each level of compression. Figure 6.16 shows the

	POD-based results			Autoencoder-based results		
		Seen	Unseen		Seen	Unseen
Compression Error	$\bar{e}^{\mathrm{POD}}_{\max}(\boldsymbol{\phi}^{\mathrm{HFM}})$	6.3880×10^{-2}	7.1400×10^{-2}	$\bar{e}^{\mathrm{AE}}_{\max}(\boldsymbol{\phi}^{\mathrm{HFM}})$	5.0533×10^{-4}	6.221×10^{-4}
Flux Error	$\bar{e}_{\max}(\boldsymbol{\phi}^{\mathrm{POD}})$	6.9400×10^{-2}	7.7714×10^{-2}	$\bar{e}_{\max}(\boldsymbol{\phi}^{\mathrm{AE}})$	5.8252×10^{-4}	7.3427×10^{-4}
k_{eff} Error	$\bar{e}(k^{\mathrm{POD}}_{\mathrm{eff}})$	9.8282×10^{-3}	1.0248×10^{-2}	$\bar{e}(k^{\mathrm{AE}}_{\mathrm{eff}})$	2.3471×10^{-5}	2.9228×10^{-5}

Table 6.4: Average maximum errors for seen results / training data and unseen results / test data using POD and a standard autoencoder to reduce from 100 variables to two variables.



(a) Scalar flux error against number of POD basis functions

(b) k_{eff} error against number of latent variables

Figure 6.16: Average maximum errors associated with POD-based and AE-based ROMs for compression to between 2 and 10 POD coefficients (POD-based method) or latent variables (AE-based method) for the 1D test case.

average maximum errors in the scalar flux and in k_{eff} for levels of compression between 2 and 10 variables. The accuracy of the POD-based ROM reduces as the number of POD coefficients decreases, and for 4 POD coefficients and fewer, the error is higher than that of the AE-based ROM. Once again, the errors for the seen and unseen sata sets are very similar. It can also be observed that the errors from the AE-based ROMs do not vary much with the number of latent variables, so that compression to two variables incurs a similar error to the reduction to ten variables.

6.4 2D reactor core eigenvalue problem

The methods are now applied to a simple 2D reactor. The domain is square-shaped with sides of length 90 cm and there are four different materials in this mock reactor, see Figure 6.17. The domain is discretised with 90 cells in both directions, each cell is of length 1 cm, making

8100 cells. In addition to this, 364 cells were used to enforce the boundary conditions. Once again, this test case was used by Buchan et al. [186] who based the cross-sections on IAEA benchmarks. The same macroscopic cross-sections are used here and their values are given in Table 6.5.

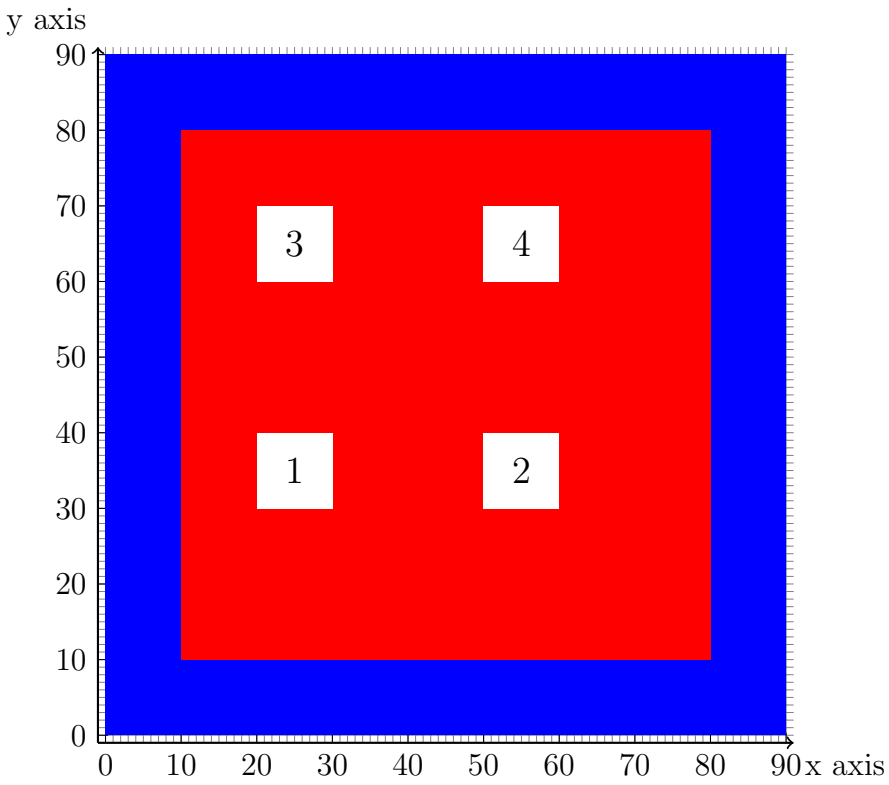


Figure 6.17: Geometry of a 2D reactor core with graphite (blue), fuel (red) and four control-rod regions labelled 1,2,3,4 (white).

	Σ_a	Σ_s	Σ_f
Fuel	0.075	0.53	0.79
Water	0.01	0.89	0
Control Rod	0.38	0.2	0
Graphite	0.15	0.5	0

Table 6.5: Macroscopic cross-sections for the materials in the 2D reactor core (units cm^{-1}).

In Figure 6.17 the blue region represents graphite, the red area contains the fuel and the four control-rod regions are white and labelled 1, 2, 3 and 4. Each control-rod region is 10 cm square with centres at (25,35), (55,35), (25,65) and (55,65) respectively. These are assumed to have a uniform distribution of the averaged properties of the mixture within the system based on a mixing coefficient for each region chosen. The method for determining the mixing coefficient is described in Section 6.2.2, however, instead of mixing fuel and control rod materials in the

control-rod regions as done for the 1D test case, here, water and control rod are mixed, with more weight given to the water in order to prevent domination of the strong absorption of the control rods.

For this system 800 high-fidelity model solutions are generated, each with four distinct values of r, one for each control-rod region. The method for generating the solutions is similar to Section 6.3. Of the 800 solutions, 400 are used during the offline phase as the snapshots (or seen data). The other 400 results make up the unseen data. These high-fidelity model solutions are used to construct a POD-based ROM and two autoencoder-based ROMs: the first uses a fully-connected autoencoder, the second uses the hybrid SVD-autoencoder. The number of degrees of freedom of the high-fidelity model solutions is 8100, and, for all ROMs, the number of POD coefficients or latent variables is chosen as 4, as this is perhaps the minimum number required to represent the system.

6.4.1 POD-based ROM

For the POD-based ROM, an SVD is used to compress the snapshots. From a possible 400 basis functions, 4 are retained, which corresponds to capturing 99.878% of the information contained in the snapshots. In the online stage the parameter values are chosen and matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ of the high-fidelity model are formed. The first 4 basis functions are used to create $\boldsymbol{R}$ and Equation (5.10) is formed. This is solved with the power method as described in Algorithm 3 and Section 5.2.

6.4.2 AE-based ROM

In the autoencoder-based ROM, the SVD is replaced by an autoencoder, which, for this case, compresses from 8100 variables to 4 latent variables.For this problem a range of networks were trained varying the numbers of layers (up to 16 layers), neurons in each layer (to a maximum of 100), batch size (10, 50, 400) and minimum learning rate (from 10^{-4} to 10^{-7}). The network with the lowest validation loss was used.

The best performing autoencoder comprised of 16 fully connected layers with the following

number of neurons in each layer:

$$8100 \to 100 \to 86 \to 72 \to 58 \to 44 \to 30 \to 16 \to 4 \to 16 \to 30 \to 44 \to 58 \to 72 \to 86 \to 100 \to 8100$$

The activation function for every layer is the exponential linear unit; the optimiser used is 'Nadam'; the initial learning rate is 10^{-3} and the minimum learning rate is 10^{-6}; and the loss function is defined to be the mean squared reconstruction error. The snapshots are scaled between 0 and 1, and the network was trained using mini-batch gradient descent with a batch size of 50. To prevent over-fitting an early stopping parameter was used to terminate training if the validation loss did not decrease over 200 successive epochs. In this example, training was stopped due to this at 24,000 epochs. Note here, the large reduction in neurons between the input and the first hidden layer. Although needed to reduce the number of layers which would otherwise be too high, this is expected to cause some difficulties with the accuracy of the network's predictions.

6.4.3 Combined Singular Value Decomposition and Autoencoder approach

As an alternative to having a large difference between the number of neurons in the input and the first hidden layer, a combination of an SVD and an autoencoder is explored. First the SVD is applied to the snapshots, after which, 100 of the 400 possible basis functions are retained. The snapshots are projected onto the basis functions (see Equation (5.8)) and the resulting snapshot coefficients are used to train the autoencoder. As with the autoencoder of the previous section, the network has 4 latent variables. To find the optimal architecture, different networks were trained with different numbers of layers and different numbers of neurons in each layer. The network with the lowest validation loss was then selected.

The best performing autoencoder consisted of 14 fully connected layers with the number of neurons in each layer being:

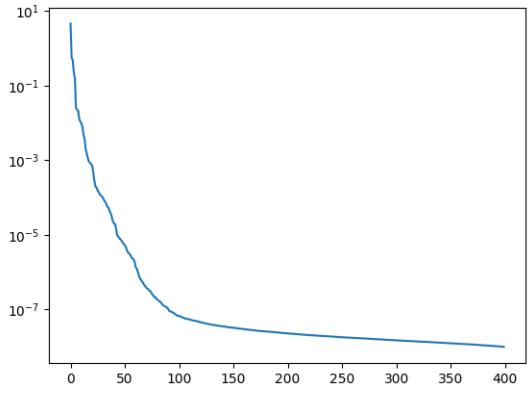
$$100 \to 100 \to 100 \to 45 \to 20 \to 9 \to 4 \to 9 \to 20 \to 45 \to 100 \to 100 \to 100$$

The activation function for every layer was the exponential linear unit; the optimiser used was

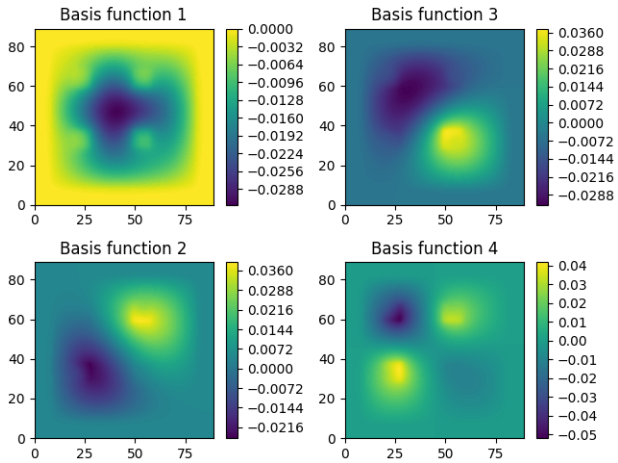
'Nadam' and the loss function was defined to be the mean squared reconstruction error. The snapshots were not scaled, and the network was trained using mini-batch gradient descent with a batch size of 50. The number of epochs was 30,000. We note that, instead of combining an SVD with the autoencoder, two additional linear layers could be added to the autoencoder, one after the input and the other before the output layer, effectively replacing the SVD. However, extra layers would make the training process more difficult and, in any case, the using the SVD is a computationally efficient way of calculating this particular transformation.

6.4.4 Compression using POD, an autoencoder and the SVD-AE

Here we compare POD, an autoencoder and the SVD-AE as methods of compression. Figure 6.18a shows the singular values of the snapshots matrix, of which the first 100 decrease rapidly, the remaining values reach a plateau. The first four POD basis functions can be seen in Figure 6.18b. Figure 6.19 shows four basis functions of the nonlinear map associated with the autoencoder, linearised at convergence, for both the seen and unseen sets of parameter values. These basis functions are columns of the $\boldsymbol{C}$ matrix. Figure 6.20 shows four basis functions of the nonlinear map associated with the SVD-autoencoder, linearised at the point of convergence, for both the seen and unseen sets of parameter values. From all three compression methods, it can be seen the basis functions capture variation associated with the control-rod regions.



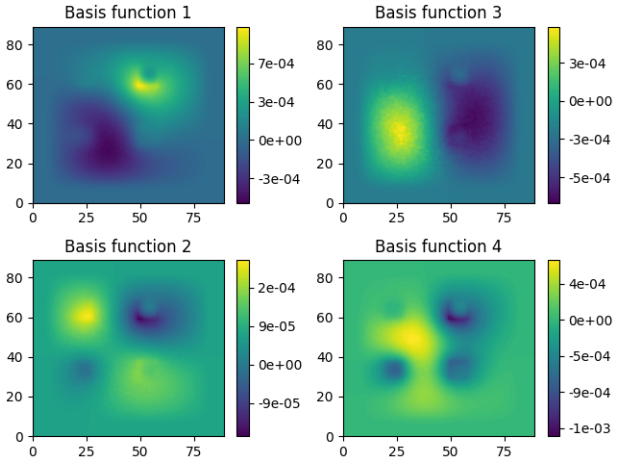
(a) Singular value against singular value index.



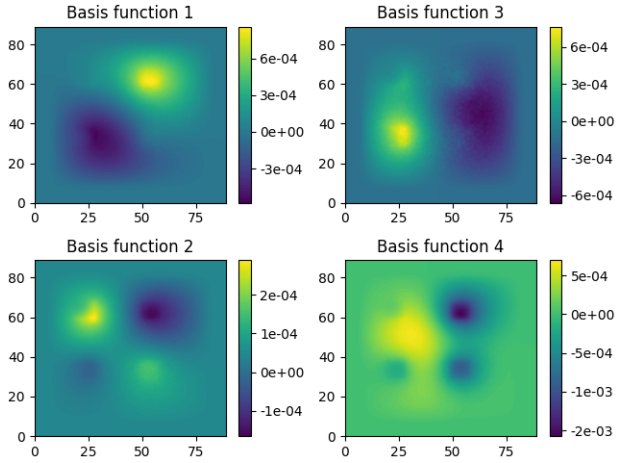
(b) First four basis functions.

Figure 6.18: Singular values and four basis functions for the 2D test case.

Figure 6.21 shows the compression errors for the POD method (see Equation (6.11)), the autoencoder (see Equation (6.12)) and the hybrid SVD-AE (see Equation (6.13)). For POD, this

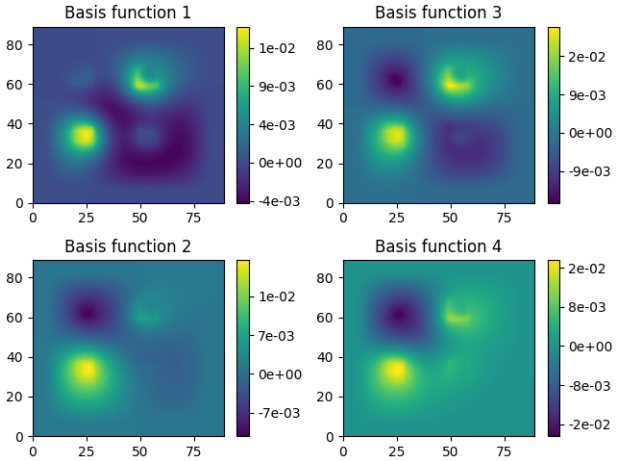


(a) For the seen problem with parameters $r_1 = 0.9782$, $r_2 = 0.9891$, $r_3 = 0.7006$ and $r_4 = 0.8316$.

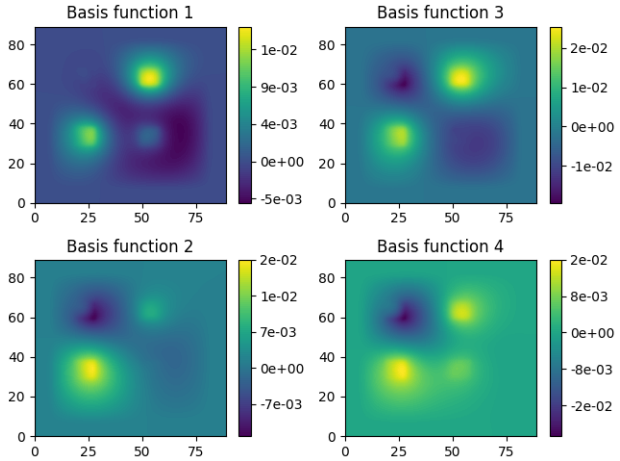


(b) For the unseen problem with parameters $r_1 = 0.9452$, $r_2 = 0.8647$, $r_3 = 0.9996$ and $r_4 = 0.9776$.

Figure 6.19: The four basis functions from the linearised $\boldsymbol{C}$ matrix generated by the autoencoder at the point of convergence.



(a) For the seen problem with parameters $r_1 = 0.9782$, $r_2 = 0.9891$, $r_3 = 0.7006$ and $r_4 = 0.8316$.

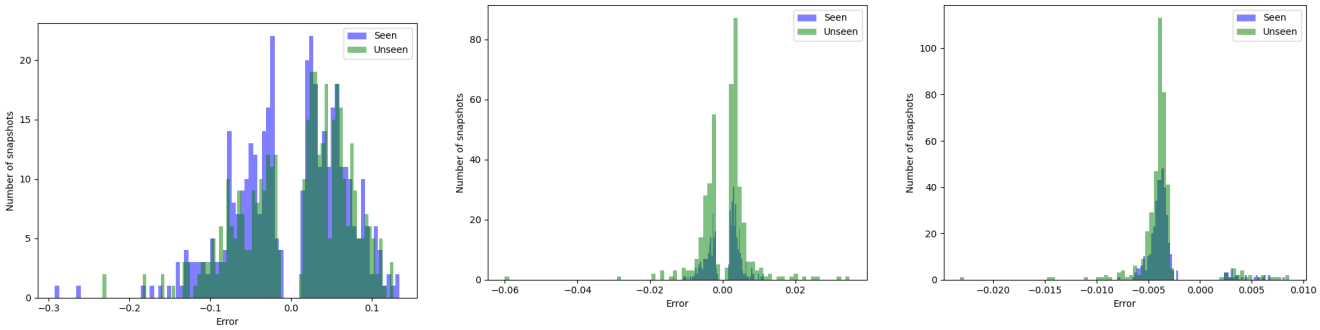


(b) For the unseen problem with parameters $r_1 = 0.9452$, $r_2 = 0.8647$, $r_3 = 0.9996$ and $r_4 = 0.9776$.

Figure 6.20: The four basis functions from the linearised $\boldsymbol{C}$ matrix generated by the SVD-autoencoder at the point of convergence.

represents the error which occurs when truncating the POD basis; for the AE, this represents the difference between each high-fidelity model solution and the corresponding output of the autoencoder after training; and for the SVD-AE, this represents the difference between the high-fidelity model solution and the result of transforming these solutions by the linear map of the SVD, passing them through the autoencoder and applying the inverse linear map of the SVD. In all three cases, the ranges of errors for seen and unseen data sets is similar, indicating for the AE-based methods, that over-fitting has been avoided. Excluding outliers, compressing

with POD introduces, at worst, an error of about 15%, whereas the corresponding errors for the AE and SVD-AE are much lower at 2% and 1% respectively. The errors for POD also exhibit a wider distribution than for the two AE-based methods. Figure 6.22 shows box and whiskers plots of the values of the 4 latent variables over the seen data for the autoencoder (Figure 6.22a) and the SVD-AE (Figure 6.22b). For the autoencoder, two of the variables take a limited range of values, whereas, for the SVD-AE, all four variables have a similar range of values. Intuition suggests that at least four variables are required to describe this problem, so the fact that the SVD-AE fully exploits the four variables might suggest that it will perform better than the AE-based method.

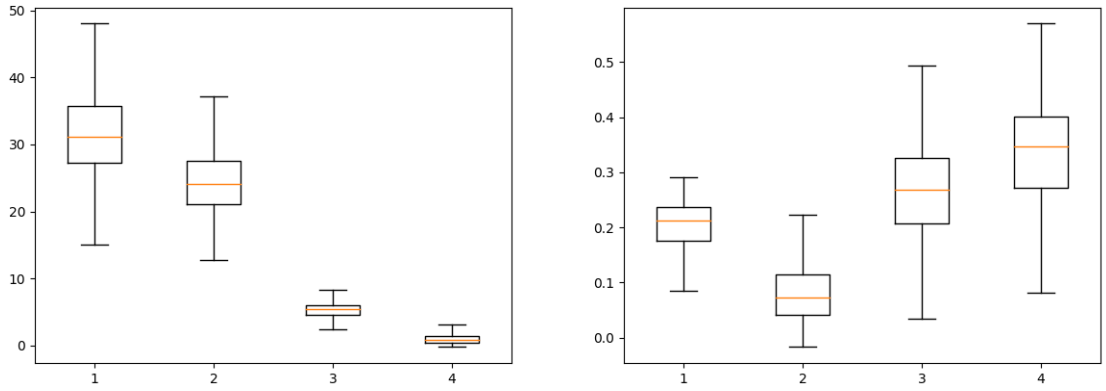


(a) Compression error of POD.

(b) Compression error of the AE.

(c) Compression error of the SVD-AE.

Figure 6.21: Compression error in the flux profile for seen (blue) and unseen (green) data sets for the 2D test case(note the different x-axis scales).



(a) Values taken by the latent variables of the AE.

(b) Values taken by the latent variables of the SVD-AE.

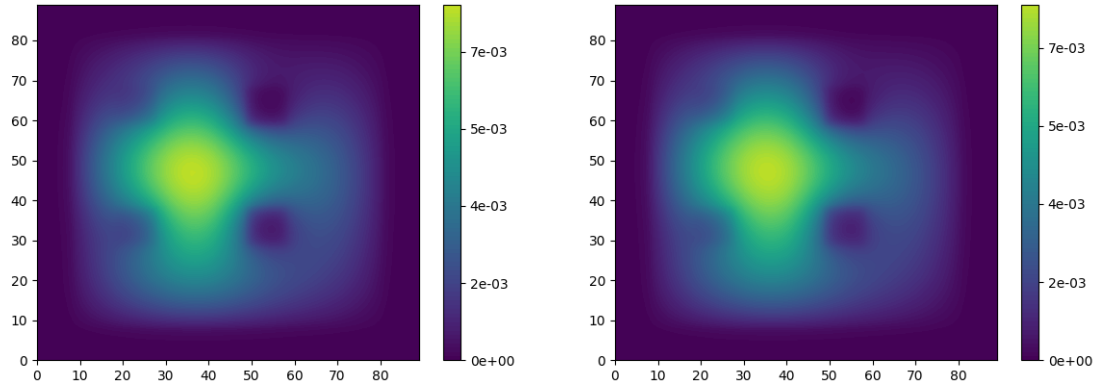
Figure 6.22: Box and whisker plots of the ranges of values taken by the 4 latent variables of the AE and SVD-AE over the seen data set. The orange line shows the median value, the box shows the upper and lower quartiles and the whiskers show the minimum and maximum values.

6.4.5 Comparing results from the ROMs based on POD, AE and SVD-AE

First, the performance of the ROMs is tested on a seen set of parameter values: $r_1 = 0.9782$, $r_2 = 0.9891$, $r_3 = 0.7006$ and $r_4 = 0.8316$. Figure 6.23 shows the scalar flux across the domain for the high-fidelity model and the three reduced-order models using POD, AE and SVD-AE for compression. All ROMs show excellent agreement with the high-fidelity model. Figure 6.24 shows the pointwise error for the three reduced-order models. The error is concentrated around the control rods. For the SVD-AE-based ROM, the error appears to be much more localised than for the POD-based and AE-based models. The maximum error for the SVD-AE-based model is also lower than for the other two models. Figure 6.25 shows the convergence of k_{eff} for the three ROMs compared with that of the high-fidelity model. The three ROMs converge faster than the high-fidelity model. The POD-based model has the highest error, finally achieving a value within 0.0035 of that of the high-fidelity model. Both of the autoencoder-based models converge to a value extremely close to that of the high-fidelity model.

The reduced-order models are now tested on a set of unseen values $r_1 = 0.9452$, $r_2 = 0.8647$, $r_3 = 0.9996$ and $r_4 = 0.9776$. The scalar fluxes across the reactor are shown in Figure 6.26 and, once again, all the ROMs show excellent agreement with the high-fidelity model. The pointwise errors are shown in Figure 6.27, which demonstrate that most of the error is concentrated around the control-rod regions. The error for the SVD-AE-based model is, once again, very localised. The error for the AE-based ROM has a similar magnitude to the other ROMs but is more diffuse and more oscillatory. All ROMs have a similar maximum error for this unseen set of parameter values. Figure 6.28 shows that the k_{eff} values of the three ROMs converge closely, and at a similar rate, to the k_{eff} value predicted by the high-fidelity model.

Figure 6.29 shows histogram plots of the errors from the POD-based ROM for both the 400 snapshots (training data) and the 400 unseen HFM solutions (the test data). Figure 6.29a shows the error in flux profile calculated using Equation (6.14), and Figure 6.29b shows the error in k_{eff} calculated using Equation (6.15). Setting aside some outliers, the error in flux profile has a range similar to the range of errors seen in the truncation of the POD basis, implying that the main contribution to the error for the POD-based ROM is due to the truncation of the POD basis. The absolute maximum error in the fluxes is over 20% and for k_{eff} is about 2%.



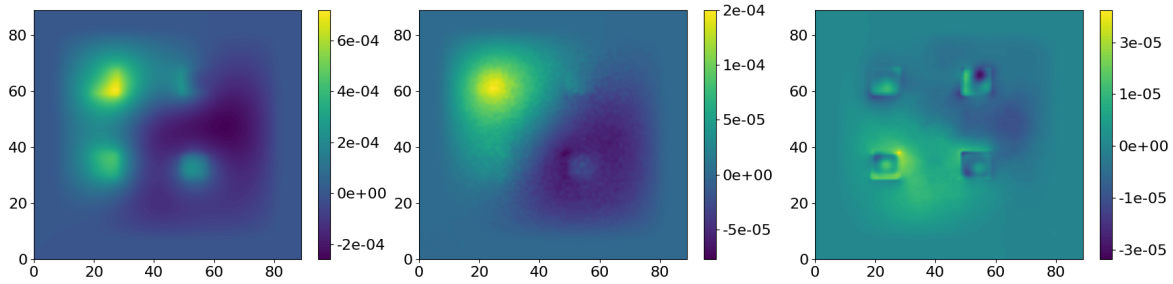
(a) High-fidelity model.

(b) POD-based ROM.

(c) AE-based ROM

(d) SVD-AE-based ROM

Figure 6.23: Scalar flux (neutrons $\mathrm{cm}^{-2}\,\mathrm{s}^{-1}$) across the reactor for a seen problem with material properties $r_1 = 0.9782$, $r_2 = 0.9891$, $r_3 = 0.7006$ and $r_4 = 0.8316$ for the 2D test case.



(a) POD-based ROM

(b) AE-based ROM.

(c) SVD-AE-based ROM

Figure 6.24: Pointwise error of the scalar flux across the reactor for a seen problem with material properties $r_1 = 0.9782$, $r_2 = 0.9891$, $r_3 = 0.7006$ and $r_4 = 0.8316$ for the 2D test case.

Figure 6.30 shows histograms of the errors for the AE-based ROM for both seen and unseen data. Both sets of data exhibit similar ranges of errors as calculated by Equation (6.14) for the error in scalar flux and Equation (6.15) for the error in k_{eff}, demonstrating that over-fitting

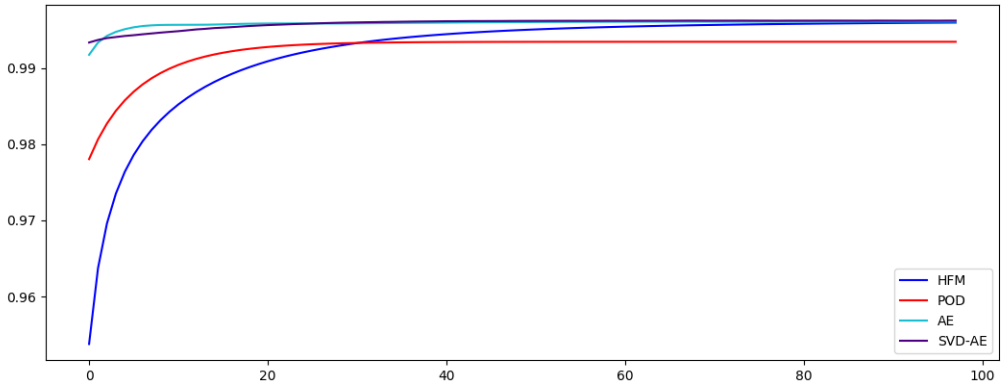
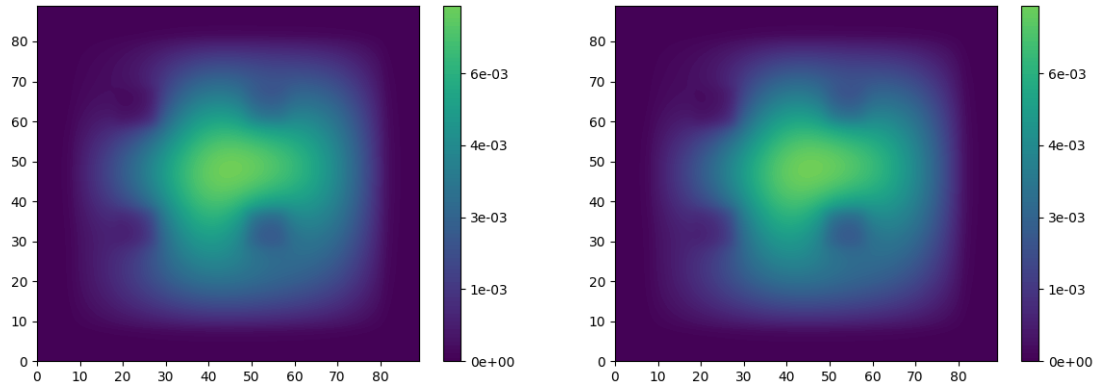
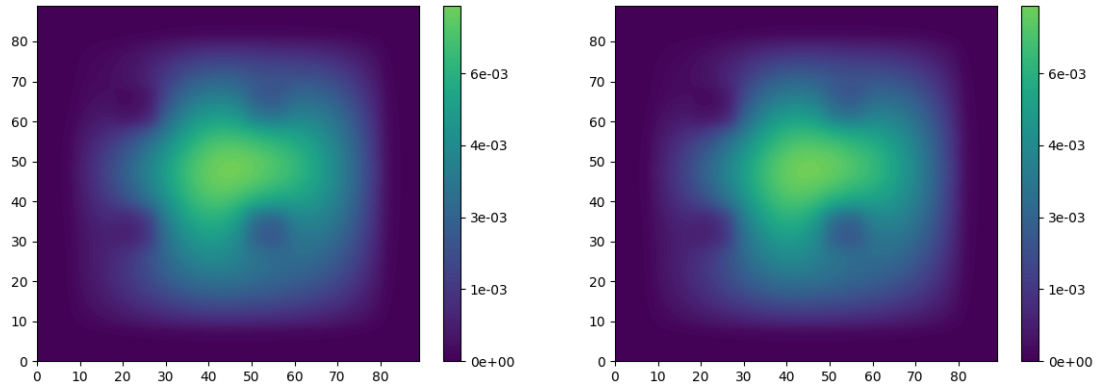


Figure 6.25: Convergence of k_{eff} for the three ROMs for the seen problem with material properties $r_1 = 0.9782$, $r_2 = 0.9891$, $r_3 = 0.7006$ and $r_4 = 0.8316$. Blue represents the high-fidelity model, red the POD-based ROM, cyan the AE-based ROM and indigo the SVD-AE-based ROM.



(a) High-fidelity model.

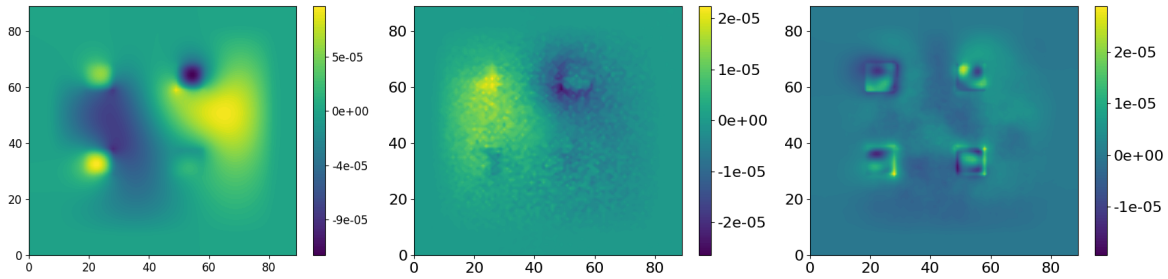
(b) POD-based ROM.



(c) AE-based ROM

(d) SVD-AE-based ROM

Figure 6.26: Scalar flux (neutrons $\text{cm}^{-2}\,\text{s}^{-1}$) across the reactor for an unseen problem with material properties $r_1 = 0.9452$, $r_2 = 0.8647$, $r_3 = 0.9996$ and $r_4 = 0.9776$ for the 2D test case.



(a) POD-based ROM (b) AE-based ROM. (c) SVD-AE-based ROM

Figure 6.27: Pointwise error of the scalar flux across the reactor for an unseen problem with material properties $r_1 = 0.9452$, $r_2 = 0.8647$, $r_3 = 0.9996$ and $r_4 = 0.9776$ for the 2D test case.

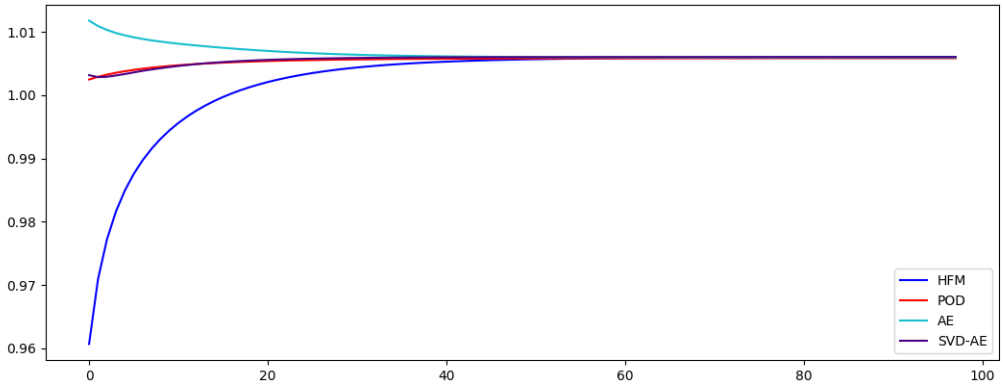
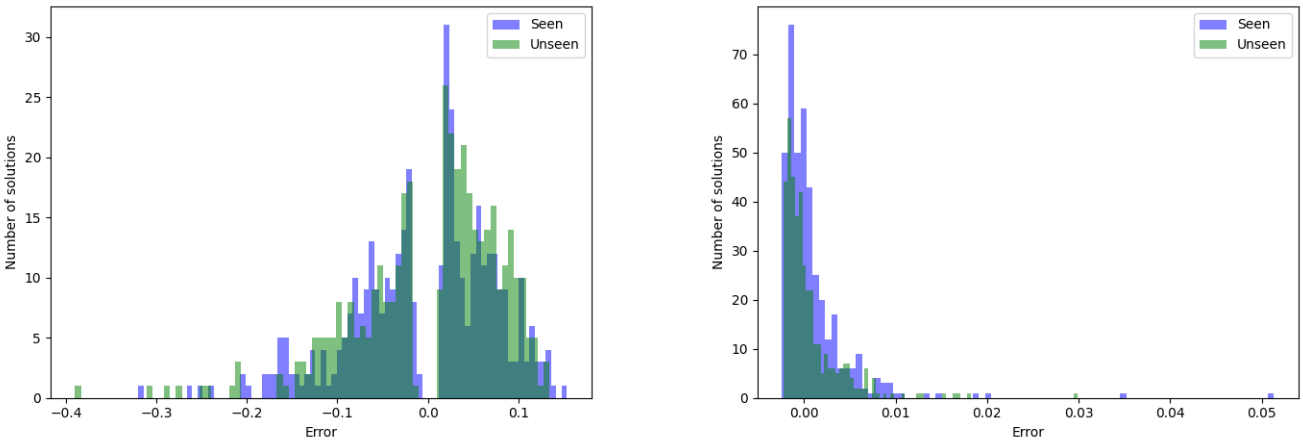


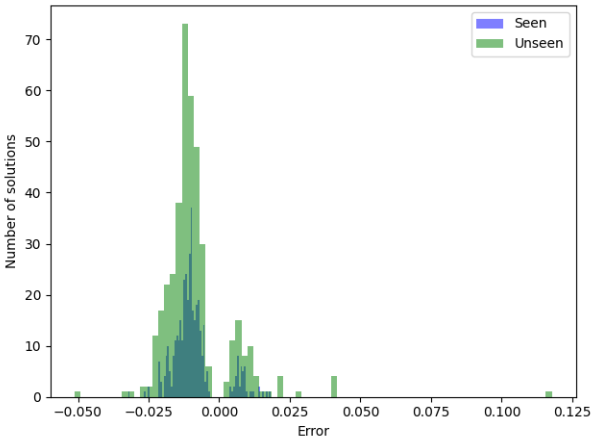
Figure 6.28: Convergence of k_{eff} for the three ROMs for the unseen problem with material properties $r_1 = 0.9452$, $r_2 = 0.8647$, $r_3 = 0.9996$ and $r_4 = 0.9776$. Blue represents the high-fidelity model, red the POD-based ROM, cyan the AE-based ROM and indigo the SVD-AE-based ROM.



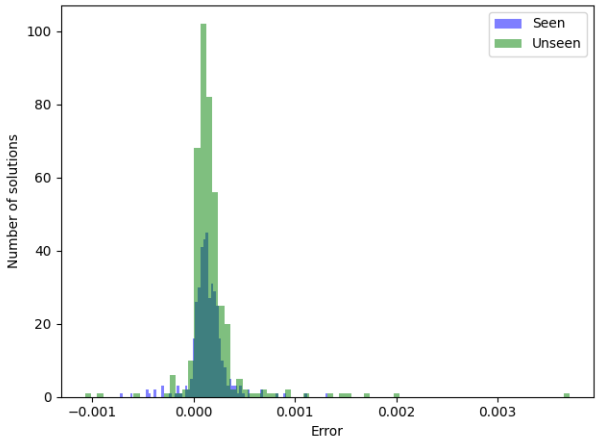
(a) Histogram of the error in the flux profile. (b) Histogram of the error in k_{eff}.

Figure 6.29: Error in the seen (blue) and the unseen (green) results for the POD-based ROM applied to the 2D test case.

has been avoided. In comparison with results from the POD-based ROM, the AE-based model shows lower errors (the absolute maximum error in flux being about 2.5% as opposed to 20% for POD). The errors in k_{eff} are an order of magnitude lower those of POD.



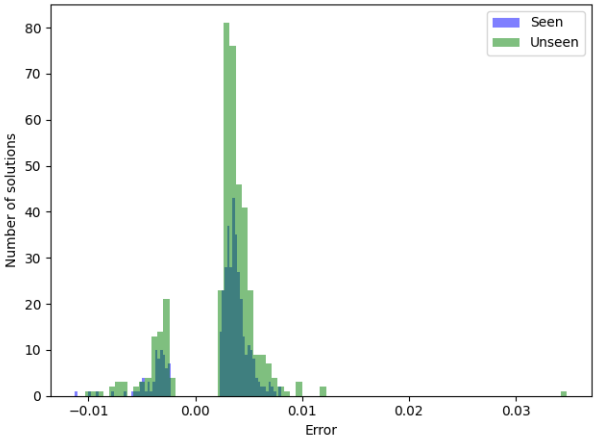
(a) Histogram of the error in the flux profile



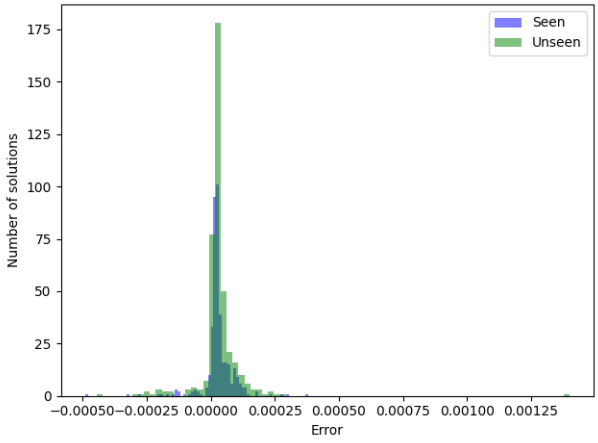
(b) Histogram of the error in k_{eff}

Figure 6.30: Error in the seen (blue) and the unseen (green) results for the autoencoder-based ROM applied to the 2D test case.

Figure 6.31 shows histograms of the errors for the SVD-AE-based ROM for both seen and unseen data. Both sets of data exhibit similar ranges of errors indicating that the SVD-autoencoder is not over-fitting. The errors are much lower than those seen in the POD-based ROM and a little lower than those of the AE-based ROM. The largest flux error is half that of the AE-based model, and the largest k_{eff} is q quarter of that of the AE-based model.



(a) Histogram of the error in the flux profile.



(b) Histogram of the error in k_{eff}

Figure 6.31: Error in the seen (blue) and the unseen (green) results for the SVD-AE-based ROM applied to the 2D test case.

Table 6.6 shows the average maximum errors, defined in Equations (6.16), (6.17) and (6.18), for

all three ROMs applied to the 2D problem. It can be seen that both autoencoder-based ROMs have lower compression errors than POD with both networks having similar compression errors. Both AE-based methods perform better than the POD-based method but the SVD-AE-based method outperforms the pure AE method. The AE-based method achieves k_{eff} errors that are an order of magnitude better than the POD-based method and the SVD-AE has k_{eff} errors lower than the AE-based method.

	POD		AE		SVD-AE	
	Seen	Unseen	Seen	Unseen	Seen	Unseen
Compression Error	5.7397×10^{-2}	5.7756×10^{-2}	3.9602×10^{-3}	5.0071×10^{-3}	4.0081×10^{-3}	4.2799×10^{-3}
Flux Error	6.4829×10^{-2}	6.5539×10^{-2}	1.1773×10^{-2}	1.3256×10^{-2}	4.5225×10^{-3}	4.7314×10^{-3}
k_{eff} Error	2.1332×10^{-3}	1.9947×10^{-3}	1.7579×10^{-4}	2.0100×10^{-4}	4.5224×10^{-5}	4.5921×10^{-5}

Table 6.6: Average maximum errors for seen data (or training data) and unseen data (test data) for the POD-based ROM, AE-based ROM and SVD-AE-based ROM for the 2D test case. The number of compressed variables for all methods is four.

6.5 Summary

A novel projection-based reduced-order model for solving eigenvalue problems has been developed by employing an autoencoder for dimensionality reduction or compression. To assess the model's effectiveness, two simple reactor problems are compared to a reduced-order model (ROM) based on Proper Orthogonal Decomposition (POD). The autoencoder-based ROMs include a unique combination of singular value decomposition (SVD) and an autoencoder.

POD and SVD are often used to obtain a set of basis functions for a low-dimensional space onto which a high-fidelity model is projected. This embedding of a high-dimensional space into a low-dimensional space is linear with POD, while an autoencoder (AE) produces a nonlinear embedding. This nonlinearity makes autoencoders more effective for accurately representing data, particularly in larger, more complex problems.

For the 1D test case, when using a reduced space of dimension two, the AE-based reduced-order model performs better than the POD-based model, with lower errors in compression and for predicting the scalar flux profiles. When predicting k_{eff}, the AE-based ROM has an error of two orders of magnitude lower than the POD-based ROM. This supports the claim that each latent variable of the autoencoder can capture more information than each POD basis function. It is shown that when the compression level is varied, POD suffers heavily from a reduction in the number of POD coefficients used. Similar error levels were achieved across different compression levels when using an autoencoder.

The rapid decay of the singular values in both test cases suggests that, with enough basis functions, the POD-based method will perform well. Indeed, for 10 basis functions, it outperforms the AE-based models in the 1D case. However, for the 1D case, when projecting onto a reduced space of dimension 2, the AE-based ROMs outperform the POD-based method. This choice of dimension 2 was motivated by results from the ROM based on a variational autoencoder: such autoencoders are imbued with the ability to compress the data into the minimum number of latent variables. For problems where many snapshots are used to capture the behaviour of the high-fidelity model, having an efficient representation of the low-dimensional space would be desirable.

For the 2D test case, a standard autoencoder-based ROM was shown to be more accurate

than the POD-based ROM showing k_{eff} errors that are an order of magnitude lower. However, the novel hybrid SVD-autoencoder is more accurate than both the other models, with lower errors for k_{eff}. Combining an SVD with an autoencoder reduces the length of the inputs to the autoencoder, which, in turn, reduces the complexity of the network, making over-fitting less likely.

Although two relatively simple test cases are presented in this chapter, future work will involve increasingly complicated (i.e. more realistic) assembly or reactor configurations, which will benefit from the increased modelling capabilities of autoencoders. For this eigenvalue problem, it can be seen that there is a clear benefit to using autoencoders over the standard POD approach. For minimal numbers of basis functions, autoencoders can be used to achieve a level of compression that cannot be matched in accuracy by POD, allowing the methodologies presented here to replace traditional POD methods for the purpose of real-time analysis or design. The methodologies presented here involve sampling the latent space of the autoencoder multiple times during the solve, a process which can be computationally demanding depending on the size of the neural network. One solution to resolve this is to pre-calculate the matrices that map the reduced variables to the scalar flux, resulting in a faster solution but with slightly lower accuracy depending on how the matrices are sampled.

Chapter 7

Domain Decomposition ROM

The content within this chapter is based on:

T. R. F. Phillips, C. E. Heaney, B. S. Tollit, P. N. Smith, and C. C. Pain. Reduced-order modelling with domain decomposition applied to multi-group neutron transport. *Energies* [11]

Chapter 2 contains methods for how the high-fidelity snapshots of this chapter were generated. Chapter 5 contains methods for Proper-Orthogonal Decomposition that were used within this chapter. Chapter 6 contains methods for the AE-based ROM that were used within this chapter.

7.1 Introduction

Nuclear reactor cores can be easily split into sub-domains, starting with fuel assemblies composed of multiple fuel rods and control rods. This forms a hierarchical structure of sub-domains. Domain decomposition [127] is a method that can take advantage of this structure, allowing for faster neutron transport solutions through parallel computing [7]. Parallel computing can be utilised by solving sub-domains simultaneously and iterating until convergence between sub-domains is achieved.

A fast non-overlapping Schwarz domain decomposition was used to solve the neutron diffusion equation applied to a PWR test case by Jamelot and Ciarlet Jr. [128]. The domain was split up into many sub-domains, and different numbers of cores were experimented with. The results showed that utilising parallel processing significantly improved the computational time required

to solve the problem with no loss in accuracy.

Domain decomposition methods do not change the governing equations of a system but instead turn a single domain problem with boundary conditions into multiple sub-domain problems with different boundary conditions [127]. This means that the internal solver used within a sub-domain can be replaced with a ROM. Combining ROM with domain decomposition methods would enable parallel computing and decrease the computational cost more than either method could achieve alone. Machine learning methods have the potential to be utilised for the reduced-order models used with domain decomposition, having been used within a non-intrusive framework and applied to multi-phase flow [133]. Machine learning methods can require extensive offline training, by separating a problem into sub-domains, the size of the networks being trained decreases, allowing multiple smaller networks to be trained in parallel.

This chapter aims to investigate if there is a loss in accuracy when combining ROM with domain decomposition and if the methods extend to using AE-based ROM. Several POD-based ROMs combined with domain decomposition are developed here, and results from these are compared with a global POD-based ROM that has no domain decomposition. These methods are tested on a simple 1D reactor case. Following the success of these methods, they are extended to a more complicated 2D reactor case, and the AE-based methods from chapter 6 are implemented alongside them.

The chapter is organised as follows: Section 7.2 contains a methodology for domain decomposition using either POD or AE-based methods, section 7.3 contains an overview of the 1D reactor contained within [11], section 7.4 contains an overview of the 2D results utilising POD and AE-based method and finally a summary of the chapter is given in section 7.5.

7.2 Methodology

The domain decomposition methods described here are a 2D expansion of the ones described in [11]. They are also extended for use with the AE-based methods established in chapter 6. A non-overlapping approach is used here to see if parallel computing could be used. Domain decomposition adds a middle iteration, performed inside the outer iteration and outside the inner iteration, outlined in Algorithm 5.

A 2D problem with a global domain Ω is split into $I \times J$ sub-domains where:

$$\Omega = \Omega_{0,0} \cup \Omega_{0,1} \cup \Omega_{1,0} \ldots \cup \Omega_{I,J}, \tag{7.1}$$

where each domain $\Omega_{i,j}$ is neighboured by domains $\Omega_{i-1,j}$,$\Omega_{i,j-1}$,$\Omega_{i+1,j}$ and $\Omega_{i,j+1}$. Each domain $\Omega_{i,j}$ has associated matrices, $\boldsymbol{A_{i,j}}$ and $\boldsymbol{B_{i,j}}$ and flux $\boldsymbol{\phi_{i,j}}$ associated with it. Depending on the ROM method chosen, each domain also has basis functions $\boldsymbol{R_{i,j}}$, if POD is chosen, or a decoder $f_{i,j}^{\text{DEC}}$ and encoder $f_{i,j}^{\text{ENC}}$, if the AE-based method is chosen. If solving just domain $\Omega_{i,j}$, then the boundary conditions must be carefully considered. An alternative to this is to construct larger domains Ω_p using one domain $\Omega_{i,j}$ and its neighbouring domains. given in figure 7.1:

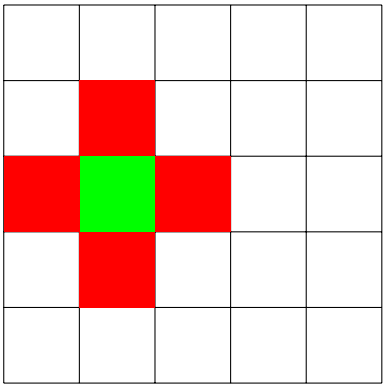


Figure 7.1: 5 by 5 grid of sub-domains, green representing the domain being solved, red representing the domains that directly neighbour the central domain and blue representing the domains diagonally adjacent to the central domain.

Figure 7.1 shows how a domain, highlighted in green, can be solved by including the information in neighbouring domains, highlighted in red. This larger domain can then be solved with only the information in the central domain $\Omega_{i,j}$ being updated upon finding a solution. If updating all domains then the blue domains would also need to be included but this is unnecessary when only the central one is updated, at least for the problem being demonstrated here. If a boundary of Ω_p lie on the edge of the global domain then he conditions are therefore known and are used. If they do not then thought would normally need to be given to what they might be, however because these boundaries are further away from the central domain that is being updated then their effect on the final solution is decreased.

Domain Ω_p has matrices $\boldsymbol{A_p}$ and $\boldsymbol{B_p}$ associated with it, alongside either the basis functions $\boldsymbol{R_p}$ or the non-linear mapping $\boldsymbol{C_p}$ generated by the autoencoders. These matrices are all constructed the same way, from the matrices for individual domains along the diagonal. For

matrix $\boldsymbol{B_p}$ this would like:

$$\boldsymbol{B_p} = \begin{bmatrix} \boldsymbol{B_{i-1,j}} & 0 & 0 & 0 & 0 \\ 0 & \boldsymbol{B_{i,j-1}} & 0 & 0 & 0 \\ 0 & 0 & \boldsymbol{B_{i,j}} & 0 & 0 \\ 0 & 0 & 0 & \boldsymbol{B_{i+1,j}} & 0 \\ 0 & 0 & 0 & 0 & \boldsymbol{B_{i,j+1}} \end{bmatrix}, \tag{7.2}$$

where $\boldsymbol{B}$ can be replaced with $\boldsymbol{R}$ or $\boldsymbol{C}$ depending on the ROM method chosen. $\boldsymbol{A}$ matrices will have additional off-diagonal terms representing interactions across domain interfaces caused by transport terms:

$$\boldsymbol{A_p} = \begin{bmatrix} \boldsymbol{A_{i-1,j}} & 0 & \boldsymbol{A_{i,j\to i-1,j}} & 0 & 0 \\ 0 & \boldsymbol{A_{i,j-1}} & \boldsymbol{A_{i,j\to i,j-1}} & 0 & 0 \\ \boldsymbol{A_{i-1,j\to i,j}} & \boldsymbol{A_{i,j-1\to i,j}} & \boldsymbol{A_{i,j}} & \boldsymbol{A_{i+1,j\to i,j}} & \boldsymbol{A_{i,j+1\to i,j}} \\ 0 & 0 & \boldsymbol{A_{i,j\to i+1,j}} & \boldsymbol{A_{i+1,j}} & 0 \\ 0 & 0 & \boldsymbol{A_{i,j\to i,j+1}} & 0 & \boldsymbol{A_{i,j+1}} \end{bmatrix}, \tag{7.3}$$

The flux in domain $\boldsymbol{\phi_p}$ and POD coefficients/latent variables $\boldsymbol{\alpha_p}$ can be determined by concatenating them together:

$$\boldsymbol{\phi_p} = \begin{bmatrix} \boldsymbol{\phi_{i-1,j}} \\ \boldsymbol{\phi_{i,j-1}} \\ \boldsymbol{\phi_{i,j}} \\ \boldsymbol{\phi_{i+1,j}} \\ \boldsymbol{\phi_{i,j+1}} \end{bmatrix}, \quad \boldsymbol{\alpha_p} = \begin{bmatrix} \boldsymbol{\alpha_{i-1,j}} \\ \boldsymbol{\alpha_{i,j-1}} \\ \boldsymbol{\alpha_{i,j}} \\ \boldsymbol{\alpha_{i+1,j}} \\ \boldsymbol{\alpha_{i,j+1}} \end{bmatrix}. \tag{7.4}$$

The flux in each domain ϕ_p is determined by using the inner iteration algorithms, either using POD or the AE-based method, with only the information required from the domains within

Ω_p. After ϕ_p is determined, only the flux in $\phi_{i,j}$ is updated. For the problems applied here each sub-domain is solved sequentially, with each sub-domain in a row being solved moving left to right and down a row until all the flux in all sub-domains have been updated. This is repeated until the absolute error in the flux is less than the tolerance 10^{-8}, or the maximum number of iterations has been reached, 100. An alternative way is to solve all sub-domains simultaneously in a parallel approach, possible because each solution is independent, further reducing the computational time taken to run. The algorithm for the sequential approach can be seen in figure 5

After the global flux ϕ has converged, then this can then be used within the outer iteration of the power method.

Algorithm 5 Domain decomposition method: middle iterations.

function MIDDLE_ITERATIONS($\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{\phi}^{\text{guess}}$, λ, P_k)
  $\boldsymbol{\phi}^{(0)} = \boldsymbol{\phi}^{\text{guess}}$
  $k\_max = 100$
  $k\_tol = 10^{-8}$
  $k = 0$
  `not_converged = True`
  while `not_converged` **do**
    for `i in` Ω **do**
      for `j in` Ω **do**
        $\Omega_p = \Omega_{i-1,j} \cup \Omega_{i,j-1} \cup \Omega_{i,j} \cup \Omega_{i+1,j} \cup \Omega_{i,j+1}$
        $\boldsymbol{\phi}_p^{(k+1)} = \text{INNER\_ITERATIONS}(\boldsymbol{A}_p, \boldsymbol{B}_p, \boldsymbol{\phi}_p^{(k)}, \lambda)$
  ! *to solve Equation* (2.12)
        $\boldsymbol{\phi}_{i,j} \leftarrow \boldsymbol{\phi}_{i,j}^{(k+1)}$! *update flux in central domain.*

    $\boldsymbol{\phi}_{diff} = |\boldsymbol{\phi}^{(k+1)} - \boldsymbol{\phi}^{(k)}|$

    if ($k = k\_max$ **or** $\overline{\phi}_{diff} < k\_tol$) **then**
      `not_converged = False`
    else
      $k \leftarrow k + 1$
  return $\boldsymbol{\phi}$

7.2.1 Error Measures

Multiple error measures are used to assess the accuracy of the results. The first is the largest normalised maximum error in the flux profile and is determined by:

$$e_{\max}(\boldsymbol{\phi}^{\mathrm{POD}}) = \frac{\phi_j^{\mathrm{HFM}} - \phi_j^{\mathrm{POD}}}{\|\boldsymbol{\phi}^{\mathrm{HFM}}\|_\infty} \quad \text{where } j = \underset{i \in \text{cells}}{\operatorname{argmax}} \mid \phi_i^{\mathrm{HFM}} - \phi_i^{\mathrm{POD}} \mid , \tag{7.5}$$

in which the superscript "POD" indicates the solution was generated by POD and i and j represent cell indices.

The second is the error in k_{eff}, which is just the difference between the HFM and POD solutions:

$$e(k_{\mathrm{eff}}^{\mathrm{POD}}) = k_{\mathrm{eff}}^{\mathrm{HFM}} - k_{\mathrm{eff}}^{\mathrm{POD}}. \tag{7.6}$$

An average maximum error is also used, which for the error in the flux profile is given by:

$$\bar{e}_{\max}(\boldsymbol{\phi}^{\mathrm{POD}}) = \frac{1}{N} \sum_{l=1}^{N} \left| \, e_{\max}(\boldsymbol{\phi}_l^{\mathrm{POD}}) \right| , \tag{7.7}$$

where N is the total number of snapshots.

7.3 1D Reactor

7.3.1 Test Problem and Construction of Snapshot Matrices

To test the method, a simple 1D problem is solved. Here, the problem is split into five domains, as seen in Figure 7.2. Each domain can either be a fuel assembly or the reflector with the same geometry and material properties of the KAISTbenchmark [149]. The 1D geometry of a fuel assembly can be seen in Figure 7.3 and represents a slice through the centre of the 2D geometry given by the KAIST benchmark. The length of each domain is 21.42 cm, the same as the length of a fuel assembly in the KAIST benchmark. Each domain is discretised into 85 cells with each cell being 0.252 cm.

An example mesh of the a domain containing a fuel assembly can be seen in Figure 7.3. Each of the 17 regions contains five cells. Regions A, B, C, D and E represent guide-tubes/control

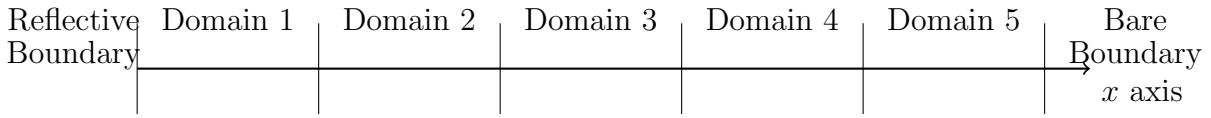


Figure 7.2: Geometry of how a 1D slab reactor may be split into five domains with a reflective boundary condition on the left-hand side and a bare boundary condition on the right-hand side.

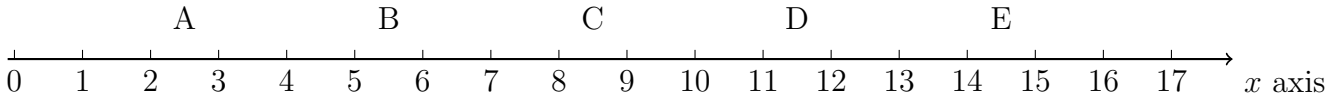


Figure 7.3: Geometry of a 1D slice taken through a fuel assembly. Regions A, B, C, D and E contain guide-tubes that can have control rods inserted. All other regions contain fuel rods with fissionable material.

rod regions. All unlabelled regions are fuel rods whose material depends on what fuel assembly it is. Mixed Oxide (MOX) fuel assemblies contain three different kinds of fuel rods that contain either 4.3%, 7.0% or 8.7% of plutonium. Uranium Oxide (UOX) fuel assemblies only contain one kind of fuel rod.

Each domain has 85 spatial degrees of freedom, and energy is discretised into 22 groups. Therefore, the full system has 9350 degrees of freedom, with each domain containing 1870 of these. The material cross-sections are homogenised across each pin-cell and are generated by WIMS11 [189] using the following:

1. KAIST 1a data specification [149]
2. JEF-2.2nuclear data evaluation
3. Equivalence theory for resonance shielding
4. Multicell collision probability flux spectrum to condense from 172 to 22 groups
5. Method of solving heterogeneous flux characteristics

Four-hundred HFM snapshots are generated for the above problem. Each snapshot has five parameters, D_1, D_2, D_3, D_4 and D_5 representing domains $1, 2, 3, 4$ and 5 respectively. Each of these parameters can either be 0 representing UOX without control rods, 1 representing UOX with control rods, 2 representing MOX without control rods, 3 representing MOX with control rods and 4 representing the reflector. D_1, D_2 and D_3 are randomly assigned a value between zero and three. D_4 and D_5 are randomly assigned a value between zero and four with the

condition that if $D_4 = 4$, then $D_5 = 4$ as well. This is because reflectors are positioned on the outside of reactors and would not be placed in-between fuel assemblies.

These 400 HFM snapshots can be considered to be the snapshot matrix U_{Global}, which is a 9350×400 matrix. All 400 snapshots are decomposed into their five domains forming a matrix U_{decomp} that is 1870×2000. Each of these 2000 snapshots has both a location and material type outlined in Table 7.1. Snapshot matrices to form the basis functions can be constructed through a few different methods. One method is to base it on the location within the global system. U_1, U_2,U_3, U_4 and U_5 are all 1870×400 matrices and contain the snapshots associated with the column in Table 7.1 that corresponds to their subscript.

Table 7.1: Table showing possible domains within a system. The rows represent the material type of domain, and the columns represent the location within the system. UOX, Uranium Oxide; MOX, Mixed Oxide.

	1	**2**	**3**	**4**	**5**
UOX	Yes	Yes	Yes	Yes	Yes unless 4 = Reflector
MOX	Yes	Yes	Yes	Yes	Yes unless 4 = Reflector
Reflector	No	No	No	Yes	Yes

A different way is to base it on the material type of domain within the system. U_{UOX} contains the data from any domain and snapshot where the parameter is zero or one and for these snapshots results in a 1870×913 matrix. U_{MOX} contains the data from any domain and snapshot where the parameter is two or three and for these snapshots results in a 1870×873 matrix. $U_{Reflector}$ contains the data from any domain and snapshot where the parameter is four and for these snapshots results in a 1870×214 matrix. U_{UOX}, U_{MOX} and $U_{Reflector}$ contain the snapshots associated with the rows in Table 7.1 corresponding to their subscript.

Another way is to base it on both the material type of domain and the location of the domain in the system. This results in 12 matrices: $U_{Typ,Loc}$ where Typ is either UOX, MOX or reflector for the material type and Loc is between one and five for the location within the system. $U_{Typ,Loc}$ are $1870 \times Y$ matrices where Y is dependent on the number for that matrix. These matrices contain any snapshots associated with both the row and column in Table 7.1 that matches their subscript.

An additional set of HFM snapshots is generated that range between one and three domains.

U_{1dom} represents the one domain set and is a 1870×8 matrix; U_{2dom} represents the two domain set and is a 3740×40 matrix; and U_{3dom} represents the three domain set and is a 5610×168 matrix. Table 7.2 shows the parameter possibilities for these smaller domains. Three snapshots' matrices are constructed from these based on the material type of domain within each snapshot. $U_{UOX,dom}$ contains the data from any domain and snapshot where the parameter is zero or one and for these snapshots results in a 1870×268 matrix. $U_{MOX,dom}$ contains the data from any domain and snapshot where the parameter is two or three and for these snapshots results in a 1870×268 matrix. $U_{Reflector,dom}$ contains the data from any domain and snapshot where the parameter is four and for these snapshots results in a 1870×56 matrix. $U_{UOX,dom}$, $U_{MOX,dom}$ and $U_{Reflector,dom}$ contain the snapshots associated with the rows in Table 7.2.

Table 7.2: Table showing possible systems within the smaller data set. The rows represent the material type of domain, and the columns represent the location and how many domains are used in the solution.

	$\boldsymbol{U}_{3dom}$			$\boldsymbol{U}_{2dom}$		$\boldsymbol{U}_{1dom}$
	1	**2**	**3**	**1**	**2**	**1**
UOX	Yes	Yes	Yes unless 2 = Reflector	Yes	Yes	Yes
MOX	Yes	Yes	Yes unless 2 = Reflector	Yes	Yes	Yes
Reflector	No	Yes	Yes	Yes	No	No

7.3.2 Construction of Basis Functions

A total of six methods were tested that used POD Galerkin as the form of ROM. The first of these did not use domain decomposition and is labelled G-POD. An SVD was applied to U_{Global}, and 250 basis functions were retained. An additional method labelled G2-POD was done using the same method as G-POD, but only 50 basis functions were retained instead of 250.

The remaining four methods all used the same domain decomposition methods, but involved the basis functions being generated from different sets of snapshot matrices explained in the previous section. The first method is labelled DDT-POD, and three sets of basis functions are constructed. These basis functions are constructed from the snapshot matrices U_{UOX}, U_{MOX} and $U_{Reflector}$ to form basis functions R_{UOX}, R_{MOX} and $R_{Reflector}$, respectively, with 50 basis functions retained in each matrix. For an example, parameter sets $D_1 = 2, D_2 = 2, D_3 =$

$1, D_4 = 1$ and $D_5 = 4$ and the set of global basis functions R could be constructed from the three sets of basis functions as follows:

$$\boldsymbol{R} = \begin{bmatrix} \boldsymbol{R_{MOX}} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{R_{MOX}} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{R_{UOX}} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{R_{UOX}} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{R_{Reflector}} \end{bmatrix} \tag{7.8}$$

The second method is labelled DDL-POD, and five sets of basis functions are constructed based on the domains' location within the system. These basis functions are constructed from the snapshot matrices U_1, U_2, U_3, U_4 and U_5 to form basis functions R_1, R_2, R_3, R_4 and R_5, respectively, with 50 basis functions retained in each matrix. Irrespective of the parameters, the set of global basis functions R could be constructed from the three sets of basis functions as follows:

$$\boldsymbol{R} = \begin{bmatrix} \boldsymbol{R_1} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{R_2} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{R_3} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{R_4} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{R_5} \end{bmatrix} \tag{7.9}$$

The third method is labelled DDTL-POD, and twelve sets of basis functions are constructed based on the material type of domain and location within the system. These basis functions are constructed from the snapshot matrices $U_{Typ,Loc}$ to form basis functions $R_{Typ,Loc}$, respective to their subscript, with 50 basis functions retained in each matrix. For an example, parameter sets $D_1 = 2, D_2 = 2, D_3 = 1, D_4 = 1$ and $D_5 = 4$ and the set of global basis functions R could be constructed from the three sets of basis functions as follows:

$$\boldsymbol{R} = \begin{bmatrix} \boldsymbol{R_{MOX,1}} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{R_{MOX,2}} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{R_{UOX,3}} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{R_{UOX,4}} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{R_{Reflector,5}} \end{bmatrix} \tag{7.10}$$

The fourth method is labelled DDT2-POD and is the same as DDT-POD with three sets of basis functions constructed, one for each material type of domain. These basis functions are constructed from the snapshot matrices $U_{UOX,dom}$, $U_{MOX,dom}$ and $U_{Reflector,dom}$ to form basis functions $R_{UOX,dom}$, $R_{MOX,dom}$ and $R_{Reflector,dom}$, respectively, with 50 basis functions retained in each matrix. For an example, parameter sets $D_1 = 2, D_2 = 2, D_3 = 1, D_4 = 1$ and $D_5 = 4$ and the set of global basis functions R could be constructed from the three sets of basis functions as follows:

$$\boldsymbol{R} = \begin{bmatrix} \boldsymbol{R_{MOX,dom}} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{R_{MOX,dom}} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{R_{UOX,dom}} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{R_{UOX,dom}} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{R_{Reflector,dom}} \end{bmatrix} \tag{7.11}$$

Table 7.3 shows the number of sets of basis functions and from where the data used to construct those basis functions comes. G-POD uses the same data set as G2-POD, but with fewer basis functions retained. DDT-POD and DDT2-POD both have their basis functions constructed determined from the material type of domain, but differing data sets.

7.3.3 Limiting the Number of Snapshots

Three out of the six methods described in Section 7.3.2, G-POD, DDT-POD and DDL-POD, were tested by limiting the number of snapshots available to each methods. Methods with this limitation are referred to as G-POD-lim, DDT-POD-lim and DDL-POD-lim. This limitation

Name	Sets of Basis Functions	Data Used to Construct Basis Functions
G-POD	1	Global data: U_{Global}, 250 basis functions retained
G2-POD	1	Global data: U_{Global}, 50 basis functions retained
DDT-POD	3	Type of domain: $U_{UOX}, U_{MOX}, U_{Reflector}$
DDL-POD	5	Location of domain: U_1, U_2, U_3, U_4, U_5
DDTL-POD	12	Location and type of domain: $U_{Typ,Loc}$
DDT2-POD	3	Type of domain from alternate data set: $U_{UOX,dom}, U_{MOX,dom}, U_{Reflector,dom}$

Table 7.3: Table showing the six methods, how many sets of basis functions are retained and the data from which they are constructed. POD, Proper Orthogonal Decomposition.

was imposed by randomly selecting 50 snapshots from U_{Global} to form the snapshot matrix U_{Lim}. Each method can only use basis functions made from U_{Lim}. G-POD-lim is therefore limited to 50 basis functions, as the number of basis functions cannot exceed the number of snapshots. Dividing U_{Lim} into domains based on the material type results in 112, 114 and 24 snapshots available for UOX, MOX and reflector, respectively. The sets of basis functions for DDT-POD-lim are constructed from these, and the number of basis functions retained are 50, 50 and 24 for R_{UOX}, R_{MOX} and $R_{Reflector}$, respectively. DDL-POD-Lim has 50 snapshots available for each location within the system, allowing 50 basis functions to be retained for each location. G-POD-lim, DDT-POD-lim and DDL-POD-lim are then used to produce solutions for the 400 sets of parameters used in Section 7.3.1 to produce U_{Global}.

7.3.4 Smart Selection of the Number of Basis Functions

G-POD, DDT-POD and DDL-POD are again tested by smart selection of the number of basis functions. Methods with this smart selection are referred to as G-POD-basis, DDT-POD-basis and DDL-POD-basis. The smart selection is formed by finding the P basis functions required to satisfy Equation (5.7) when $\gamma = 0.99999$. These basis functions were constructed from the snapshot matrix U_{Global}. It was found that G-POD-basis required 98 basis functions to capture 99.999% of the energy based on Equation (5.7). For DDT-POD-basis: R_{UOX}, R_{MOX} and $R_{Reflector}$ required 30, 30 and 13 basis functions, respectively, to capture 99.999% of the energy based on Equation (5.7). This means that DDT-POD-basis has between 116 and 150 basis functions depending on what types of domains are contained within the global system, based on the set of parameters being solved. For DDL-POD-basis: R_1, R_2, R_3, R_4 and R_5 required 22, 37, 45, 45 and 28 basis functions, respectively, to capture 99.999% of the energy based on Equation (5.7), for a total of 177 basis functions.

7.3.5 POD-Based Results

A look at how each method performs for a single set of parameters ($R_1 = 2, R_2 = 2, R_3 = 1, R_4 = 1, R_5 = 4$) is shown here. First, k_{eff} convergence is shown and, following this, a look at the flux profiles and pointwise error for a few energy groups.

G-POD uses one set of basis functions to reduce the dimensionality of the full system from 9350 variables to 250. G2-POD uses one set of basis functions to reduce the dimensionality of the full system from 9350 variables to 50. DDT-POD, DDL-POD, DDTL-POD and DD2-POD all use five sets of basis functions that each reduce the dimensionality in a single domain from 1870 variables to 50, with the sets' basis functions used determined by the parameters of the solution being solved.

Figure 7.4 shows the k_{eff} convergence for all methods along with the HFM. All methods converge to a k_{eff} value close to the HFM in under 15 iterations.

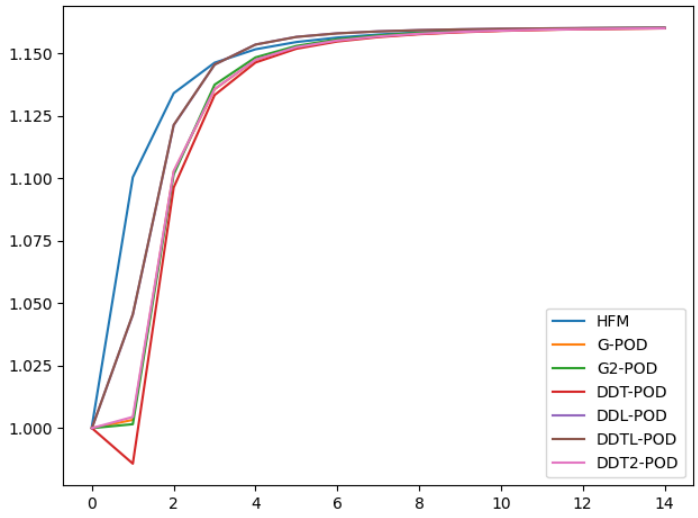
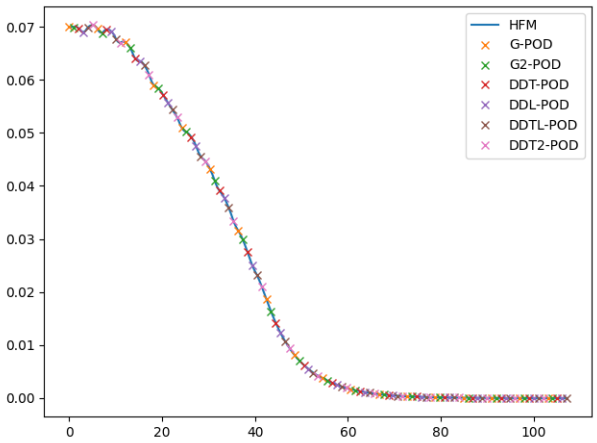
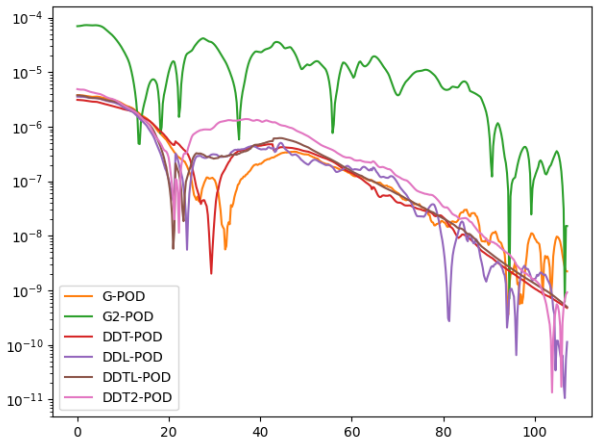


Figure 7.4: k_{eff} vs. iteration number for all methods and the High-Fidelity Model (HFM).

The scalar fluxes and the absolute pointwise error in the profiles are now compared for each ROM. Figures 7.5–7.10 show the flux profile for different energy groups for all methods. Figures 7.5 shows Energy Group 1. Figure 7.6 shows Energy Group 5. Figure 7.7 shows Energy Group 9. Figure 7.8 shows Energy Group 13. Figure 7.9 shows Energy Group 17. Figure 7.10 shows Energy Group 21. All methods show good approximation to the HFM. The absolute pointwise error is similar for all methods except G2-POD, which is an order of magnitude worse.

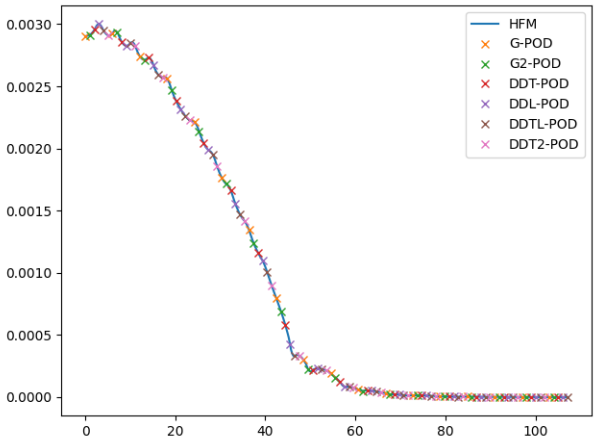


(a) Scalar flux (neutrons $\mathrm{cm}^{-2}\,\mathrm{s}^{-1}$) vs. x (cm), HFM solution in blue, POD-based Reduced-Order Models (ROMs) in crosses.

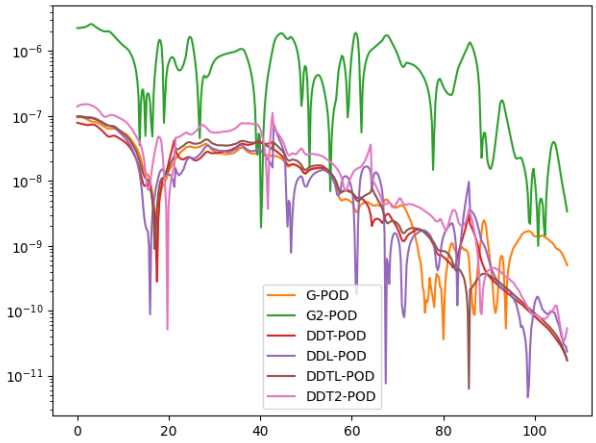


(b) Absolute pointwise error (neutrons $\mathrm{cm}^{-2}\,\mathrm{s}^{-1}$) vs. x (cm) for all methods.

Figure 7.5: Scalar flux for Energy Group 1 for the HFM (blue) and POD methods and the pointwise error between them with parameters $R_1 = 2, R_2 = 2, R_3 = 1, R_4 = 1, R_5 = 4$.



(a) Scalar flux (neutrons $\mathrm{cm}^{-2}\,\mathrm{s}^{-1}$) vs. x (cm), HFM solution in blue, POD-based ROMs in crosses.

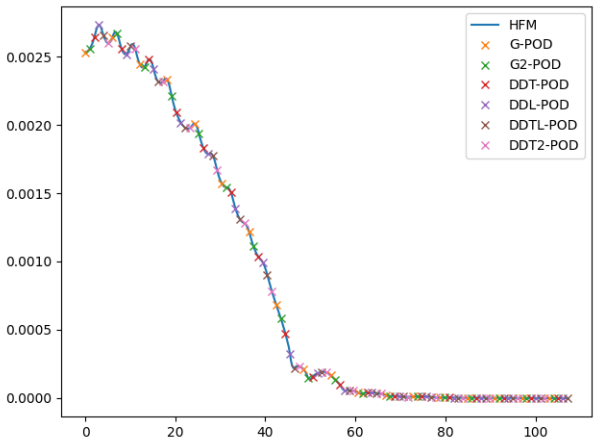


(b) Absolute pointwise error (neutrons $\mathrm{cm}^{-2}\,\mathrm{s}^{-1}$) vs. x (cm) for all methods.

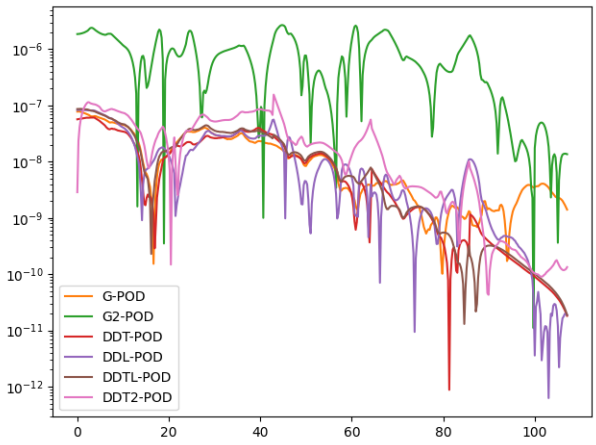
Figure 7.6: Scalar flux for Energy Group 5 for the HFM (blue) and POD methods and the pointwise error between them with parameters $R_1 = 2, R_2 = 2, R_3 = 1, R_4 = 1, R_5 = 4$.

Figure 7.11 shows the errors for all six methods in boxplot format. G-POD, DDT-POD, DDL-POD, DDTL-POD and DDT2-POD all show errors in the same order of magnitude. Out of these methods, DDT2-POD and DDL-POD show the largest range and the largest values. G2-POD shows the worst errors being two orders of magnitude worse than the other four.

Table 7.4 shows the mean absolute maximum error in the flux profile and the mean k_{eff} for the 400 sets of parameters. It can be seen that G-POD, DDT-POD, DDL-POD and DDT2-POD all have mean errors in the same magnitude. Out of these four DDL-POD has the worst mean

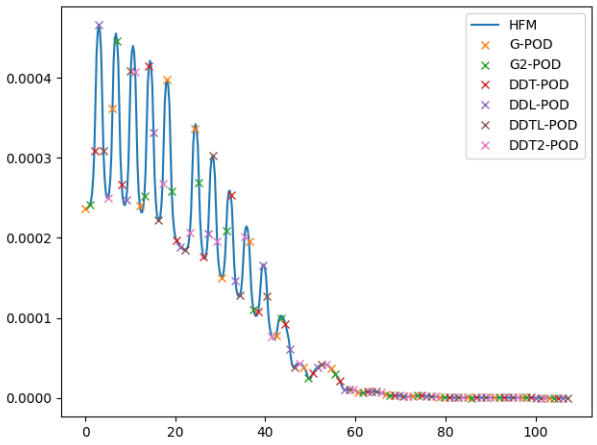


(a) Scalar flux (neutrons $\mathrm{cm}^{-2}\,\mathrm{s}^{-1}$) vs. x (cm), HFM solution in blue, POD-based ROMs in crosses.

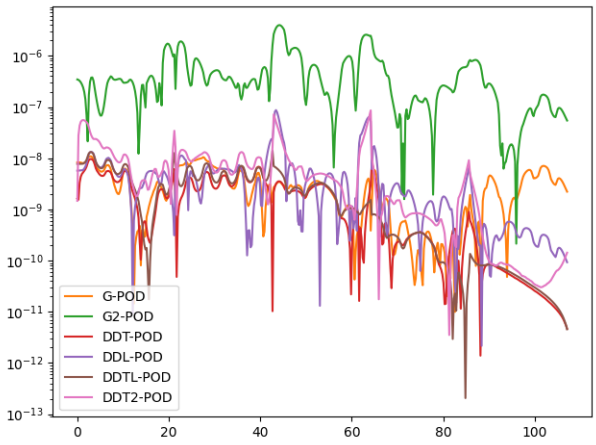


(b) Absolute pointwise error (neutrons $\mathrm{cm}^{-2}\,\mathrm{s}^{-1}$) vs. x (cm) for all methods.

Figure 7.7: Scalar flux for Energy Group 9 for the HFM (blue) and POD methods and the pointwise error between them with parameters $R_1 = 2, R_2 = 2, R_3 = 1, R_4 = 1, R_5 = 4$.



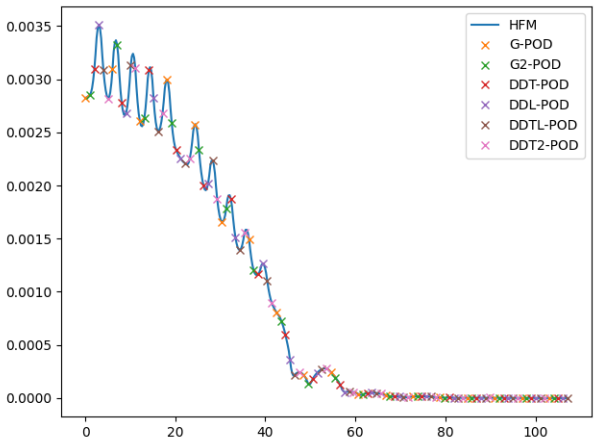
(a) Scalar flux (neutrons $\mathrm{cm}^{-2}\,\mathrm{s}^{-1}$) vs. x (cm), HFM solution in blue, POD-based ROMs in crosses.



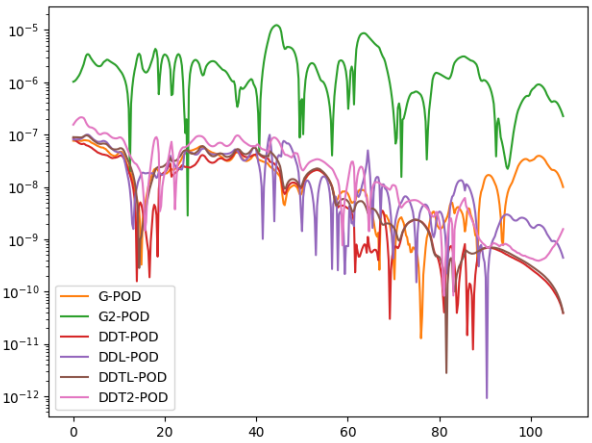
(b) Absolute pointwise error (neutrons $\mathrm{cm}^{-2}\,\mathrm{s}^{-1}$) vs. x (cm) for all methods.

Figure 7.8: Scalar flux for Energy Group 13 for the HFM (blue) and POD methods and the pointwise error between them with parameters $R_1 = 2, R_2 = 2, R_3 = 1, R_4 = 1, R_5 = 4$.

flux error, and DDT2-POD has the worst mean k_{eff} error. G-POD has the best mean flux and k_{eff} errors. G2-POD has a mean flux error that is an order of magnitude worse than the other four and has a mean k_{eff} error that is two orders of magnitude worse.

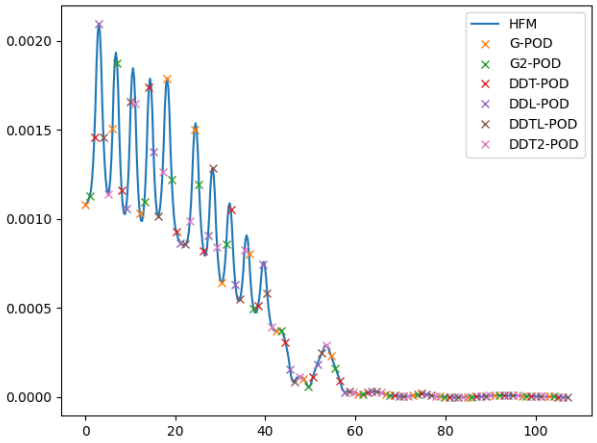


(a) Scalar flux (neutrons $\mathrm{cm}^{-2}\,\mathrm{s}^{-1}$) vs. x (cm), HFM solution in blue, POD-based ROMs in crosses.

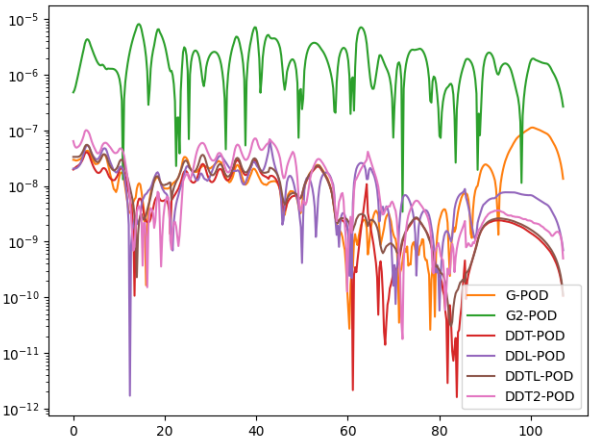


(b) Absolute pointwise error (neutrons $\mathrm{cm}^{-2}\,\mathrm{s}^{-1}$) vs. x (cm) for all methods.

Figure 7.9: Scalar flux for Energy Group 17 for the HFM (blue) and POD methods and the pointwise error between them with parameters $R_1 = 2, R_2 = 2, R_3 = 1, R_4 = 1, R_5 = 4$.



(a) Scalar flux (neutrons $\mathrm{cm}^{-2}\,\mathrm{s}^{-1}$) vs. x (cm), HFM solution in blue, POD-based ROMs in crosses.

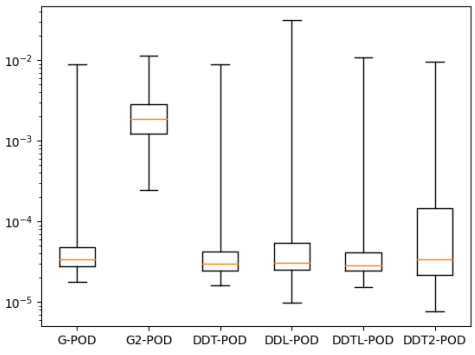


(b) Absolute pointwise error (neutrons $\mathrm{cm}^{-2}\,\mathrm{s}^{-1}$) vs. x (cm) for all methods.

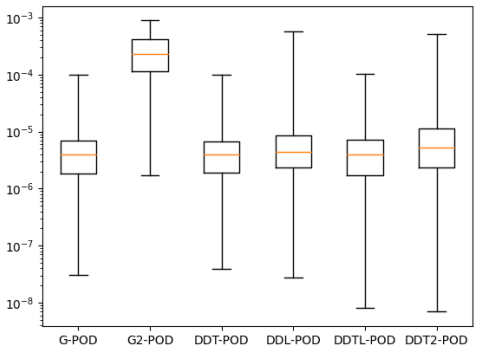
Figure 7.10: Scalar flux for Energy Group 21 for the HFM (blue) and POD methods and the pointwise error between them with parameters $R_1 = 2, R_2 = 2, R_3 = 1, R_4 = 1, R_5 = 4$.

7.3.6 Results for Limiting the Number of Snapshots and Smart Selection of the Number of Basis Functions

This section contains the results from limiting the number of snapshots and smart selection of the number of basis functions. The same set of parameters used in Section 7.3.5 ($R_1 = 2, R_2 = 2, R_3 = 1, R_4 = 1, R_5 = 4$) is used here to investigate how the methods perform with these two



(a) Boxplot of the error in the flux profile vs. the method.



(b) Boxplot of the error in k_{eff} vs. the method.

Figure 7.11: Boxplots showing the errors for all six methods. The median is given by the orange line, the interquartile ranges by the box and the minimum and maximum values by whiskers.

	G-POD	**G2-POD**	**DDT-POD**	**DDL-POD**	**DDTL-POD**	**DDT2-POD**
Flux Error	1.2286×10^{-3}	2.7860×10^{-2}	1.2391×10^{-3}	1.7601×10^{-3}	1.6563×10^{-3}	1.6593×10^{-3}
k_{eff} Error	2.2952×10^{-5}	4.3557×10^{-3}	2.3154×10^{-5}	2.3901×10^{-5}	2.6866×10^{-5}	5.6589×10^{-5}

Table 7.4: Mean absolute maximum errors in the flux profile and k_{eff} for solutions using POD-based reduced-order models.

tests. First, k_{eff} convergence is shown and, following this, a look at the flux profiles and the pointwise error for differing energy groups.

G-POD-lim uses one set of basis functions to reduce the dimensionality of the full system from 9350 variables to 50. DDT-POD-lim reduces the dimensionality of the full system from 9350 variables to between 198 and 250, depending on the types of domains contained within the problem being solved, using five sets of basis functions. The set of parameters being examined here has the dimensionality reduced to 224. DDL-POD-lim reduces the dimensionality of the system from 9350 variables to 250 using five sets of basis functions. G-POD-basis uses one set of basis functions to reduce the dimensionality of the system from 9350 variables to 98. DDT-POD-basis uses five sets of basis functions to reduce the system from 9350 variables to between 116 and 150 depending on the types of domains contained within the problem being solved. For the set of parameters being examined here, the dimensionality is reduced to 133. DDL-POD-basis uses five sets of basis functions to reduce the system from 9350 variables to 177.

Figure 7.12 shows the k_{eff} convergence for the three methods with a limited number of snapshots

and smart selection of the number of basis functions along with the HFM. All tested methods converge to a k_{eff} value close to the HFM in under 15 power iterations.

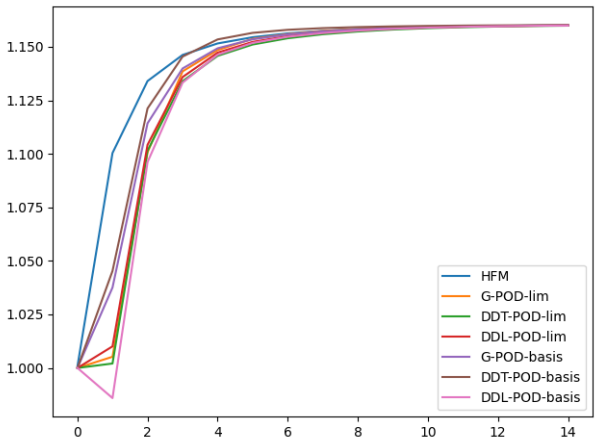
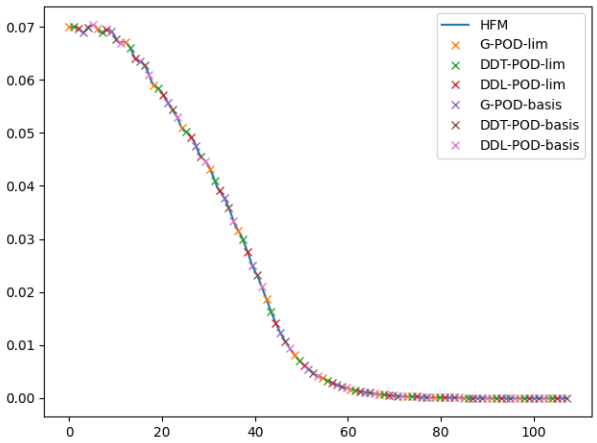


Figure 7.12: k_{eff} vs. iteration number for the HFM and the three methods with limited (lim) number of snapshots or smart selection of number of basis functions.

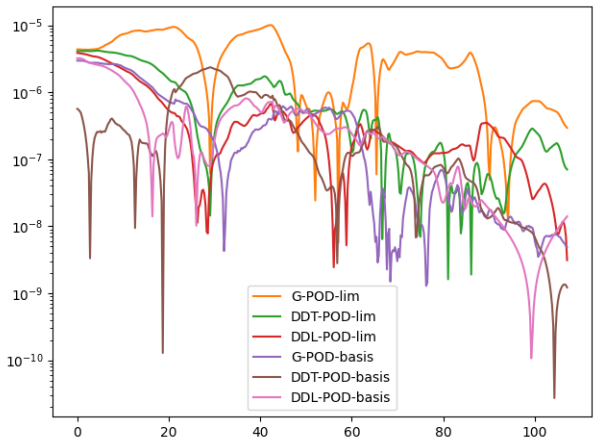
The scalar fluxes and the absolute pointwise error in the profiles are now compared for each ROM. Figures 7.13–7.18 show the flux profile for different energy groups for the HFM and the three methods with a limited number of snapshots and smart selection of number of basis functions. Figure 7.13 shows Energy Group 1. Figure 7.14 shows Energy Group 5. Figure 7.15 shows Energy Group 9. Figure 7.16 shows Energy Group 13. Figure 7.17 shows Energy Group 17. Figure 7.18 shows Energy Group 21. All methods show good approximation to the HFM, although G-POD-lim has higher pointwise errors than the other methods shown. G-POD-basis, DDT-POD-basis and DDL-POD-basis show pointwise errors in the same order of magnitude.

Figure 7.19 shows the errors for all methods with a limited number of snapshots or smart selection of the number of basis functions in boxplot format. When the number of snapshots is limited, it can be observed that G-POD-lim has medians and interquartile ranges that are an order of magnitude worse than either domain decomposition method. DDL-POD-lim has lower medians and interquartile ranges than DDT-POD-lim. When smart selection of a number of basis functions is done, it can be observed that G-POD-basis, DDT-POD-basis and DDL-POD-basis all show very similar medians and interquartile ranges despite having different numbers of basis functions.

Table 7.5 shows the mean absolute maximum error in the flux profile and the mean k_{eff} for

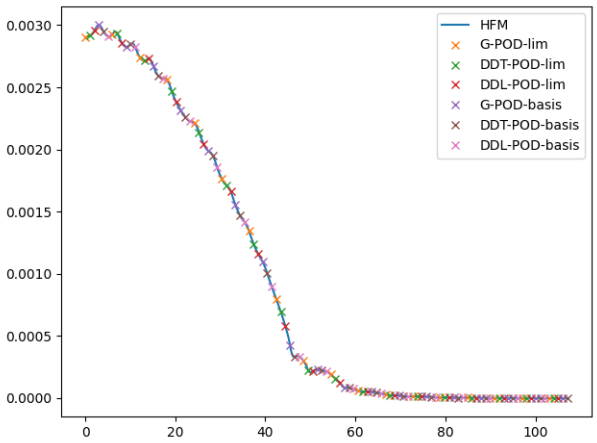


(a) Scalar flux (neutrons $\mathrm{cm}^{-2}\,\mathrm{s}^{-1}$) vs. x (cm), HFM solution in blue, POD-based ROMs, with the limited number of snapshots or smart selection of the number of basis functions, in crosses.

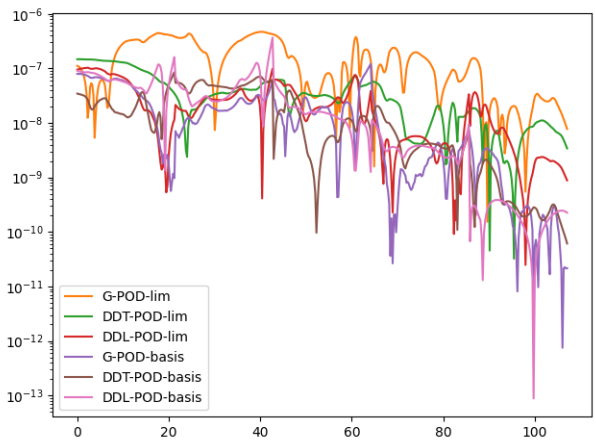


(b) Absolute pointwise error (neutrons $\mathrm{cm}^{-2}\,\mathrm{s}^{-1}$) vs. x (cm) for all POD-based ROMs, with the limited number of snapshots or smart selection of the number of basis functions, in crosses.

Figure 7.13: Scalar flux for Energy Group 1 for the HFM (blue) and POD-based ROMs, with limited number of snapshots or smart selection of number of basis functions, in crosses, and the pointwise error between them with parameters $R_1 = 2, R_2 = 2, R_3 = 1, R_4 = 1, R_5 = 4$.



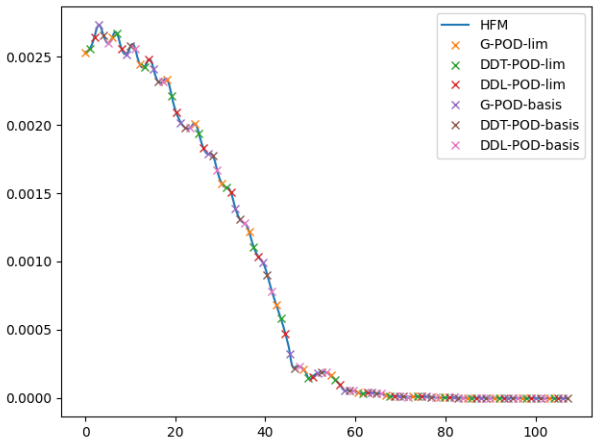
(a) Scalar flux (neutrons $\mathrm{cm}^{-2}\,\mathrm{s}^{-1}$) vs. x (cm), HFM solution in blue, POD-based ROMs, with the limited number of snapshots or smart selection of the number of basis functions, in crosses.



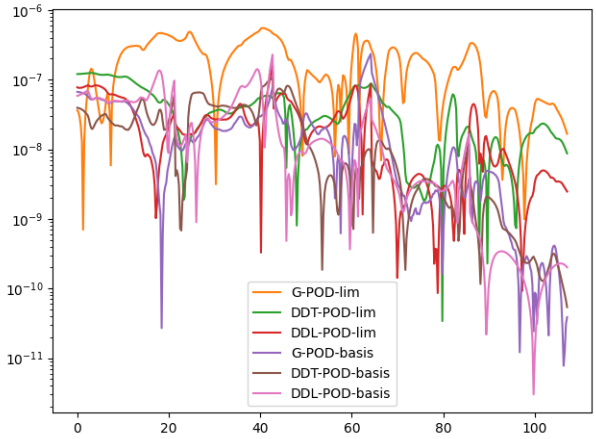
(b) Absolute pointwise error (neutrons $\mathrm{cm}^{-2}\,\mathrm{s}^{-1}$) vs. x (cm) for all POD-based ROMs, with the limited number of snapshots or smart selection of the number of basis functions, in crosses.

Figure 7.14: Scalar flux for Energy Group 1 for the HFM (blue) and POD-based ROMs, with the limited number of snapshots or smart selection of the number of basis functions, in crosses, and the pointwise error between them with parameters $R_1 = 2, R_2 = 2, R_3 = 1, R_4 = 1, R_5 = 4$.

the 400 sets of parameters. It can be observed that G-POD-lim has errors that are at least an order of magnitude worse than DDT-POD-lim and DDL-POD-lim. Of the three methods

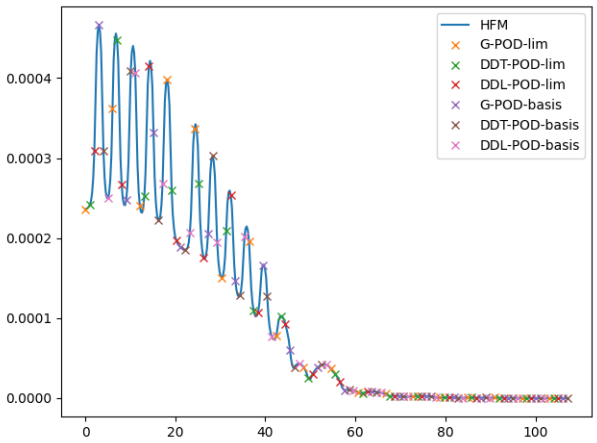


(a) Scalar flux (neutrons $\mathrm{cm}^{-2}\,\mathrm{s}^{-1}$) vs. x (cm), HFM solution in blue, POD-based ROMs, with the limited number of snapshots or smart selection of the number of basis functions, in crosses.

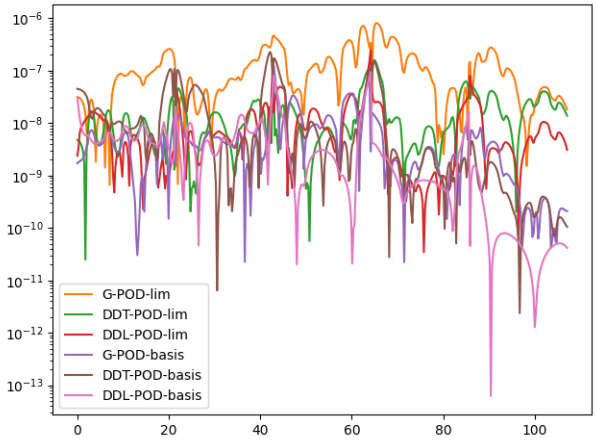


(b) Absolute pointwise error (neutrons $\mathrm{cm}^{-2}\,\mathrm{s}^{-1}$) vs. x (cm) for all POD-based ROMs, with the limited number of snapshots or smart the selection of number of basis functions, in crosses.

Figure 7.15: Scalar flux for Energy Group 1 for the HFM (blue) and POD-based ROMs, with the limited number of snapshots or smart selection of the number of basis functions, in crosses, and the pointwise error between them with parameters $R_1 = 2, R_2 = 2, R_3 = 1, R_4 = 1, R_5 = 4$.



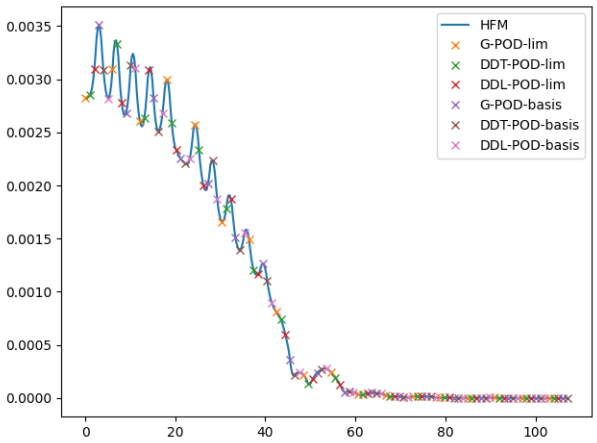
(a) Scalar flux (neutrons $\mathrm{cm}^{-2}\,\mathrm{s}^{-1}$) vs. x (cm), HFM solution in blue, POD-based ROMs, with the limited number of snapshots or smart selection of the number of basis functions, in crosses.



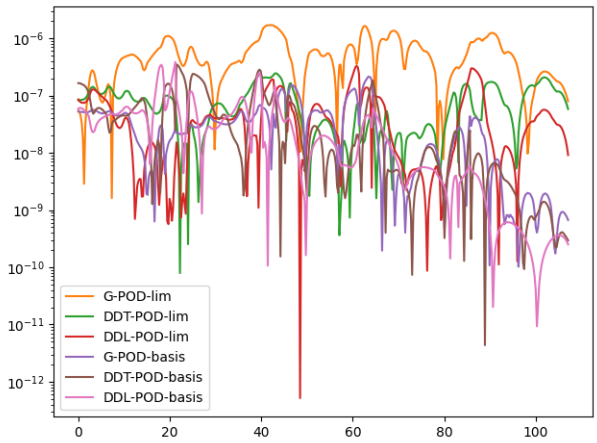
(b) Absolute pointwise error (neutrons $\mathrm{cm}^{-2}\,\mathrm{s}^{-1}$) vs. x (cm) for all POD-based ROMs, with the limited number of snapshots or smart selection of the number of basis functions, in crosses.

Figure 7.16: Scalar flux for Energy Group 1 for the HFM (blue) and POD-based ROMs, with the limited number of snapshots or smart selection of the number of basis functions, in crosses, and the pointwise error between them with parameters $R_1 = 2, R_2 = 2, R_3 = 1, R_4 = 1, R_5 = 4$.

tested by limiting the number of snapshots, DDL-POD-lim has the lowest mean errors. G-POD-basis, DDT-POD-basis and DDL-POD-basis all show mean errors of the same magnitude, with

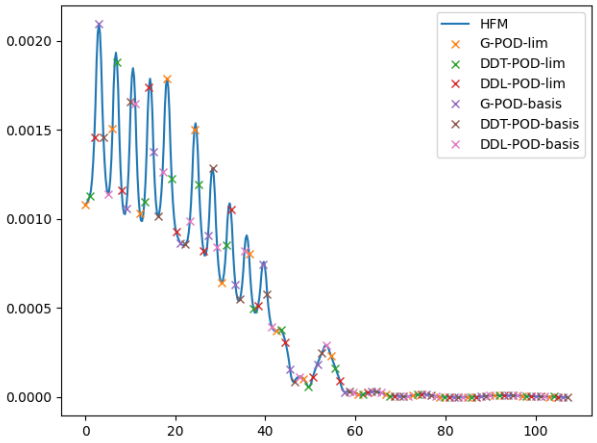


(a) Scalar flux (neutrons $\mathrm{cm}^{-2}\,\mathrm{s}^{-1}$) vs. x (cm), HFM solution in blue, POD-based ROMs, with the limited number of snapshots or smart selection of the number of basis functions, in crosses.

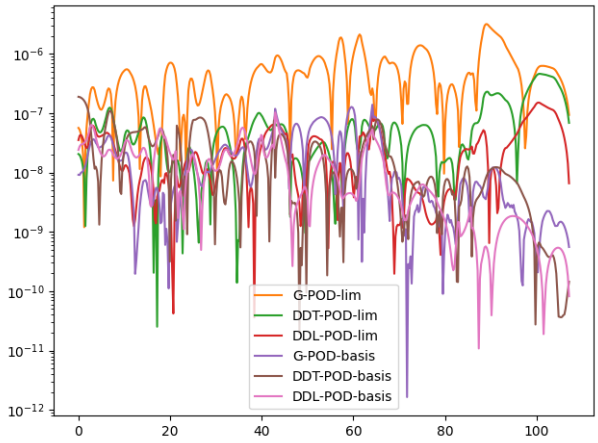


(b) Absolute pointwise error (neutrons $\mathrm{cm}^{-2}\,\mathrm{s}^{-1}$) vs. x (cm) for all POD-based ROMs, with the limited number of snapshots or smart selection of the number of basis functions, in crosses.

Figure 7.17: Scalar flux for Energy Group 1 for the HFM (blue) and POD-based ROMs, with the limited number of snapshots or smart selection of the number of basis functions, in crosses, and the pointwise error between them with parameters $R_1 = 2, R_2 = 2, R_3 = 1, R_4 = 1, R_5 = 4$.



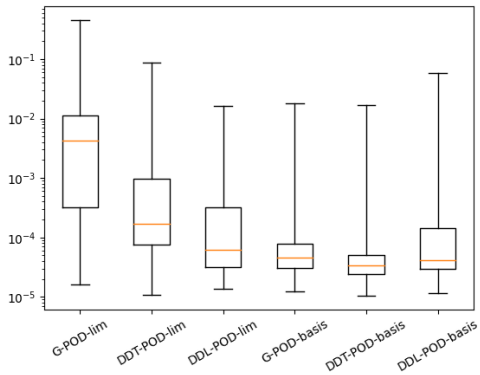
(a) Scalar flux (neutrons $\mathrm{cm}^{-2}\,\mathrm{s}^{-1}$) vs. x (cm), HFM solution in blue, POD-based ROMs, with the limited number of snapshots or smart selection of the number of basis functions, in crosses.



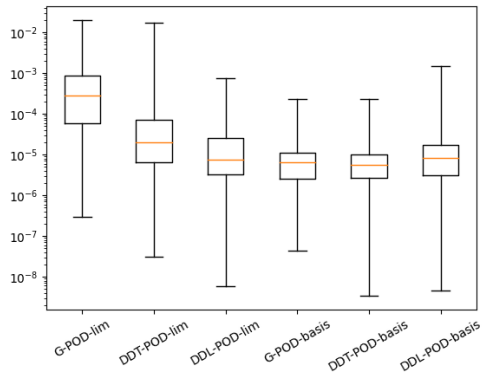
(b) Absolute pointwise error (neutrons $\mathrm{cm}^{-2}\,\mathrm{s}^{-1}$) vs. x (cm) for all POD-based ROMs, with the limited number of snapshots or smart selection of the number of basis functions, in crosses.

Figure 7.18: Scalar flux for Energy Group 1 for the HFM (blue) and POD-based ROMs, with the limited number of snapshots or smart selection of the number of basis functions, in crosses, and the pointwise error between them with parameters $R_1 = 2, R_2 = 2, R_3 = 1, R_4 = 1, R_5 = 4$.

DDL-POD-basis being slightly worse than the other two.



(a) Boxplot of the error in the flux profile vs. the methods with the limited number of snapshots and smart selection of the number of basis functions.



(b) Boxplot of the error in k_{eff} vs. the methods with the limited number of snapshots and smart selection of the number of basis functions.

Figure 7.19: Boxplots showing the errors for methods with the limited number of snapshots and smart selection of the number of basis functions. The median is given by the orange line, the interquartile ranges by the box and the minimum and maximum values by whiskers.

	G-POD-lim	**DDT-POD-lim**	**DDL-POD-lim**	**G-POD-basis**	**DDT-POD-basis**	**DDL-POD-basis**
Flux Error	2.4846×10^{-1}	7.5716×10^{-3}	2.1496×10^{-3}	1.2416×10^{-3}	1.2375×10^{-3}	3.1000×10^{-3}
k_{eff} Error	2.3198×10^{-2}	1.3405×10^{-3}	7.5412×10^{-5}	2.5474×10^{-5}	2.4721×10^{-5}	6.1249×10^{-5}

Table 7.5: Mean absolute maximum errors in the flux profile and k_{eff} for solutions using POD-based reduced-order models with the limited number of snapshots and smart selection of the number of basis functions.

7.4 2D Reactor

7.4.1 Test Problem and construction of snapshot matrices

A 2D reactor core is solved in order to test the method. This 2D reactor core is based on the KAIST benchmark [149]. The possible configurations for the reactor core can be seen in figure 7.20. This is 4×4 grid of sub-domains that can be UOX fuel assemblies, MOX fuel assemblies or reflector material. Each of these sub-domains is $21.42cm \times 21.42cm$, the same size as the KAIST benchmark. Each of these sub-domains has 40×40 cells with each cell being 0.5355×0.5355.

Each fuel assembly is a 5×5 grid of fuel rods or guide tubes/control rods. The configuration for the UOX fuel assembly can be seen in figure 7.21 and the configuration for the MOX fuel

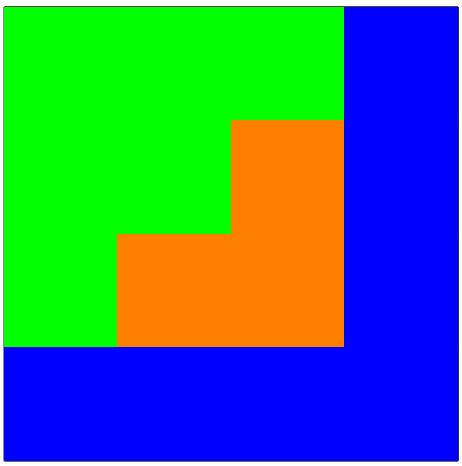


Figure 7.20: 4×4 grid of possible configurations for the reactor. Green represents domains that can be either UOX or MOX fuel assemblies. Orange represents areas that can be either UOX or MOX fuel assemblies or contain reflector material. Blue represents Reflector material.

assembly can be seen in figure 7.21. The fuel assemblies contain 21 fuel rods and have 4 guide tubes, these can either be empty of have control rods fully inserted. Each sub-domain in the 5×5 grid has 8×8 cells.

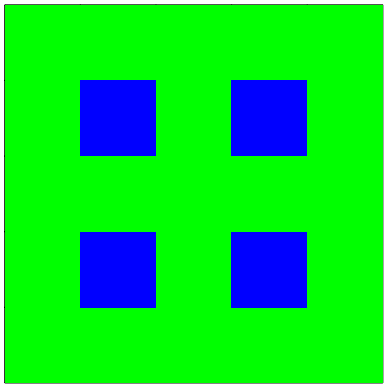


Figure 7.21: 5×5 grid configuration of the UOX fuel assembly. Green represents fuel rods containing Uranium Oxide. Blue represents guide tubes that can either be empty or contain control rods.

The configuration for the fuel rod and guide tube sub-domains can be seen in figure 7.23. These sub-domains have a surrounding grid of coolant material and a central area whose material properties depend on the sub-domain it is representing. For fuel rods this material is either UOX or MOX (4.3%, 7.0% or 8.3%). For guide tubes this is either control rod material, if control rods are inserted, or coolant material, if control rods are removed.

Each possible reactor configuration therefore has 160×160 cells giving 25600 spatial degrees of freedom. Energy is also discretised into 7 groups, raising the total number of degrees of freedom to 179200. The material cross-sections for this are taken from the KAIST benchmark [149].

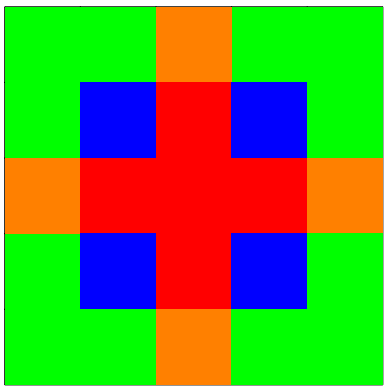


Figure 7.22: 5×5 grid configuration of the MOX fuel assembly. Green, Orange and Red represents fuel rods containing Mixed Oxide fuel with either 4.3% (Green), 7.0% (Orange) or 8.3% (Red) of plutonium. Blue represents guide tubes that can either be empty or contain control rods.

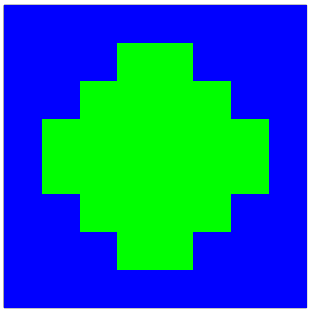


Figure 7.23: 8×8 grid configuration of fuel rods and guide tubes. Green represents either the UOX or MOX material or the guide tube and control rod materials depending on what is represents. Blue represents coolant material.

Four-hundred HFM snapshots are generated for the above problem. Each snapshot has 9 parameters which represent the 9 domains in the top left corner of figure 7.20: $D_{1,1}$, $D_{1,2}$ ... $D_{3,3}$ representing domains $\Omega_{1,1}$, $\Omega_{1,2}$ ... $\Omega_{3,3}$ respectively. These parameters are randomly assigned a value between 0 and either 3 or 4 depending on the position. A value of 0 represents UOX without control rods inserted, a value of 1 represents UOX with control rods inserted, a value of 2 represents MOX without control rods inserted, a value of 3 represents MOX with control rods inserted and a value of 4 represents reflector material.

The snapshot matrix U_{Global} contains these 400 HFM snapshots and is 179200×400. Previous research indicated that one of the best ways to decompose a reactor core into sub-domains was to construct snapshot matrices from the type of material in each sub-domain [11]. The sub-domains used to form the snapshot matrices can be separated into two methods. In the first method the snapshot matrices are constructed from the fuel assemblies forming three different matrices:

1. U_{UOX} containing any sub-domain that is a UOX fuel assembly, size 11200×1783
2. U_{MOX} containing any sub-domain that is a MOX fuel assembly, size 11200×1703
3. $U_{Reflector}$ containing any sub-domain that is reflector material, size 11200×2914

In the second method the snapshot matrices are constructed from the fuel rods forming six different matrices:

1. U_{UOX} containing any sub-domain that is a UOX fuel rod, size 448×37443
2. $U_{MOX4.3}$ containing any sub-domain that is a MOX 4.3% fuel rod, size 448×20436
3. $U_{MOX7.0}$ containing any sub-domain that is a MOX 7.0% fuel rod, size 448×6812
4. $U_{MOX8.3}$ containing any sub-domain that is a MOX 8.3% fuel rod, size 448×8515
5. $U_{Reflector}$ containing any sub-domain that is reflector material, size 448×72850
6. U_{Guide} containing any sub-domain that is a guide tube or control rod, size 448×13944

7.4.2 Additional data set and construction of alternative snapshot matrices

An additional data set is generated here that are also used to construct alternative snapshot matrices. The purpose of this data set is to investigate if ROMs constructed from a smaller data set, a 3×3 of sub-domains can be used to produce solutions of the larger problem outlined in the previous section. The possible configurations for this can be seen in figure 7.24. Four-hundred HFM snapshots are generated for the above alternative data. Each snapshot has 9 parameters which represent the 9 domains in figure 7.24. These again can be be between 0 and 4 with the values representing the same sub-domains as in the previous section.

The snapshot matrix U_{Alt} contains these 400 HFM snapshots and is 100800×400. If using the first method the snapshot matrices are constructed from the fuel assemblies forming three different matrices:

1. $U_{UOX,Alt}$ containing any sub-domain that is a UOX fuel assembly, size 11200×1503

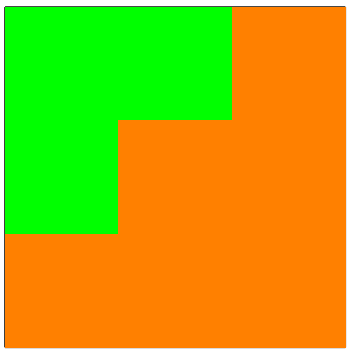


Figure 7.24: 3×3 grid of possible configurations for the alternative dataset. Green represents domains that can be either UOX or MOX fuel assemblies. Orange represents areas that can be either UOX or MOX fuel assemblies or contain reflector material.

2. $U_{MOX,Alt}$ containing any sub-domain that is a MOX fuel assembly, size 11200×1472

3. $U_{Reflector,Alt}$ containing any sub-domain that is reflector material, size 11200×625

If using the second method the snapshot matrices are constructed from the fuel rods forming six different matrices:

1. $U_{UOX,Alt}$ containing any sub-domain that is a UOX fuel rod, size 448×17031

2. $U_{MOX4.3,Alt}$ containing any sub-domain that is a MOX 4.3% fuel rod, size 448×9204

3. $U_{MOX7.0,Alt}$ containing any sub-domain that is a MOX 7.0% fuel rod, size 448×3068

4. $U_{MOX8.3,Alt}$ containing any sub-domain that is a MOX 8.3% fuel rod, size 448×3835

5. $U_{Reflector,Alt}$ containing any sub-domain that is reflector material, size 448×5550

6. $U_{Guide,Alt}$ containing any sub-domain that is a guide tube or control rod, size 448×6312

7.4.3 Construction of ROMs

There are two types of ROM used here and all methods are tested with both types. The first type of ROM is the POD method and the second type of ROM is the autoencoder method. The POD method involves applying SVD to individual snapshot matrices to produce a basis function matrix that corresponds with that snapshot matrix. The autoencoder method involves training an autoencoder on each snapshot matrix. Each method is labelled in the form ROM-LEVEL-NUM where ROM is the type of ROM, either POD or AE, LEVEL is the level at which compression is done, either FA for fuel assemblies or FR for fuel rods, and NUM is the number of variables compressed to, either 10 or 50. An additional -ALT can be included to represent the construction using the alternative data set. An example is AE-FR-10, which uses autoencoders to compress each fuel rod to 10 variables and would involve training 6 autoencoders using the following matrices:

1. U_{UOX} being used to train $f^{\mathrm{AE}_{UOX}}$ with a latent space of 10
2. $U_{MOX4.3}$ being used to train $f^{\mathrm{AE}_{MOX4.3}}$ with a latent space of 10
3. $U_{MOX7.0}$ being used to train $f^{\mathrm{AE}_{MOX7.0}}$ with a latent space of 10
4. $U_{MOX8.3}$ being used to train $f^{\mathrm{AE}_{MOX8.3}}$ with a latent space of 10
5. $U_{Reflector}$ being used to train $f^{\mathrm{AE}_{Reflector}}$ with a latent space of 10
6. U_{Guide} being used to train $f^{\mathrm{AE}_{Guide}}$ with a latent space of 10

Another example would be POD-FA-50-ALT which involves using POD to produce basis functions for each fuel assembly from the alternative data set and retains 50 basis functions:

1. $U_{UOX,Alt}$ being used to construct $R_{UOX,Alt}$ retaining 50 basis functions
2. $U_{MOX,Alt}$ being used to construct $R_{MOX,Alt}$ retaining 50 basis functions
3. $U_{Reflector,Alt}$ being used to construct $R_{Reflector,Alt}$ retaining 50 basis functions

7.4.4 Scaling factor for Autoencoder-based ROMs

It has been noted that a 3×3 reactor core produces flux in a different magnitude than a 4×4 reactor core. POD can handle this difference but autoencoders struggle to match this, requiring that data be scaled into a single magnitude. A method is presented here to scale data to a single magnitude pre-training. To use the autoencoders to produce 4×4 reactor core solutions then the training data from 3×3 reactor core dataset needs to be scaled to to the 4×4 reactor core magnitude.

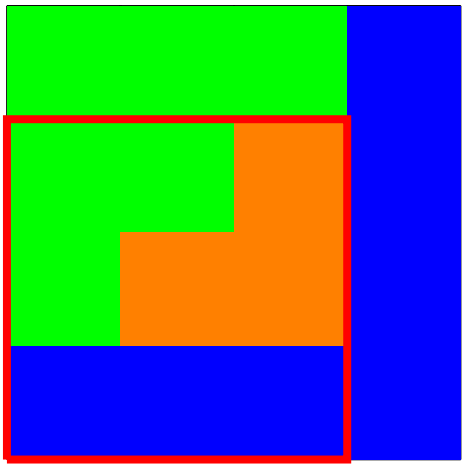


Figure 7.25: 4×4 grid of possible configurations for the reactor with a red square showing how the alternative data-set is produce from a part of the full reactor.

Figure 7.25 shows how a 3×3 grid might be overlayed over a 4×4 grid. Two solutions can be generated, one consisting of the material properties within the full 4×4 grid and one only consisting of material properties within the 3×3 grid for a single sample i. A number of samples containing two solutions must be generated, in this case 50 samples were chosen. The system is then split into four quadrants, given by figure 7.26. The data from the 4×4 solution that is contained within a single quadrant ($\phi_{a,i,true}$) is then compared with the data from 3×3 solution that matches the material parameters and is also contained within the same quadrant ($\phi_{a,i,alt}$). A scaling factor (s) for each quadrant (a) and each energy group (g) is given by dividing the 4×4 data (ϕ_{true}) by the 3×3 data (ϕ_{alt}) for a single sample i and taking the mean across all of them:

$$s_{a,g} = \overline{(\frac{\phi_{a,i,true,g}}{\phi_{a,i,alt,g}})}, \forall g \in \{1, 2, \ldots, N_g\}, \tag{7.12}$$

Data from the 3×3 core solutions can then scaled to be the same magnitude as the 4×4 dataset by multiplying by the scaling factor:

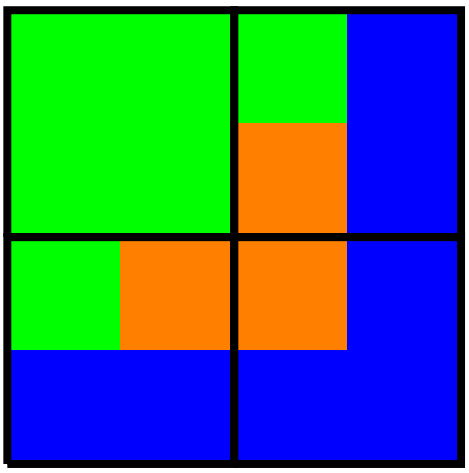
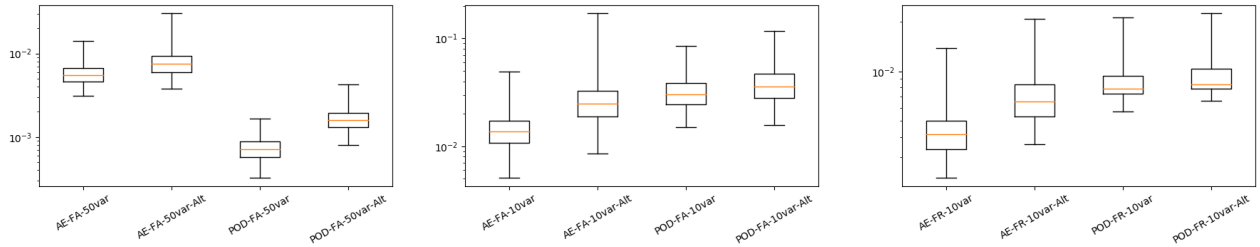


Figure 7.26: 4×4 grid of possible configurations for the reactor with black squares showing the four quadrants.

$$\phi_{a,alt-scaled,g} = s_{a,g}\dot{\phi}_{a,alt,g} \tag{7.13}$$

The dataset containing $\phi_{a,alt-scaled,g}$ contains the 350 other snapshots not used to determine the scaling factor. This is then used to train the autoencoders. These autoencoders are already scaled for the 4×4 solutions, therefore no scaling factor is needed while generating solutions.

7.4.5 Compression



(a) Boxplot of the compression errors for the fuel assembly compression to 50 variables using Autoencoders and POD.

(b) Boxplot of the compression errors for the fuel assembly compression to 10 variables using Autoencoders and POD.

(c) Boxplot of the compression errors for the fuel rod compression to 10 variables using Autoencoders and POD.

Figure 7.27: Boxplot of the compression errors using Autoencoders trained on the standard and alternative dataset and POD basis functions generated from the standard and alternative dataset. The median is given by the orange line, the interquartile ranges by the box and the minimum and maximum values by whiskers.

Figure 7.27 show the boxplots of the compression errors for all methods. It can be observed that

for the 50 variables the POD methods outperforms the AE methods. It can also be seen that the methods utilising the Alternative dataset all perform marginally worse than the standard dataset.

FA-50Var	**AE-FA-50Var**	**AE-FA-50Var-Alt**	**POD-FA-50Var**	**POD-FA-50Var-Alt**
Compression Error	5.821×10^{-3}	8.149×10^{-3}	7.549×10^{-4}	1.682×10^{-3}
FA-10Var	**AE-FA-10Var**	**AE-FA-10Var-Alt**	**POD-FA-10Var**	**POD-FA-10Var-Alt**
Compression Error	1.484×10^{-2}	2.906×10^{-2}	3.349×10^{-2}	3.941×10^{-2}
FR-10Var	**AE-FR-10Var**	**AE-FR-10Var-Alt**	**POD-FR-10Var**	**POD-FR-10Var-Alt**
Compression Error	4.484×10^{-3}	7.171×10^{-3}	8.89×10^{-3}	9.646×10^{-3}

Table 7.6: Mean compression errors in the flux profile for dimensionality reduction methods using AEs and POD.

Table 7.6 shows the mean compression errors across all samples. The 50 variable fuel assembly compression shows that POD is much more effective, being almost an entire order of magnitude better than the AE methods. For the lower variable compression, all compression errors are within the same order of magnitude, with the AE methods being slightly better than POD.

7.4.6 Results

Figure 7.28 shows the test configuration used for the results in this section. Figure 7.28a shows the 9 parameter values, surrounded by the reflector sub-domains. Figure 7.28b shows the actual sub-domain material properties. This solution required the UOX, MOX and Reflector ROMs. The UOX ROM is used 5 times, the MOX ROM is used 4 times and the Reflector ROM is used 7 times.

Figure 7.29 shows the Scalar flux profiles for the POD-FA-50var and POD-FA-50var-Alt ROMs and the pointwise comparison with the HFM. The highest errors appear to occur at the sub-domain interfaces, apparent around the fuel assemblies.

Figure 7.30 shows the Scalar flux profiles for the AE-FA-50var and AE-FA-50var-Alt ROMs and the pointwise comparison with the HFM. Again, The highest errors appear to occur at the sub-domain interfaces, apparent around the fuel assemblies. These errors are an order of magnitude worse than the POD ones in figure 7.29.

Figure 7.31 shows the k_{eff} convergence for each method compressing the fuel assemblies to 50

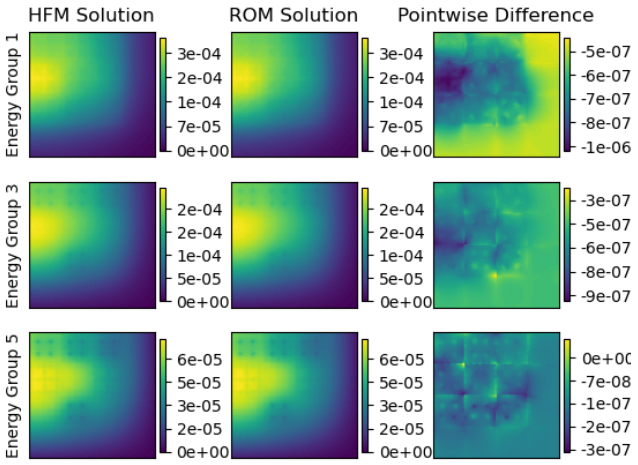
$D_{1,1}$ =1	$D_{1,2}$ =3	$D_{1,3}$ =3	4
$D_{2,1}$ =2	$D_{2,2}$ =0	$D_{2,3}$ =2	4
$D_{3,1}$ =0	$D_{3,2}$ =1	$D_{3,3}$ =0	4
4	4	4	4

(a) Reactor test configuration showing values of the 9 parameters and surrounding reflector domains.

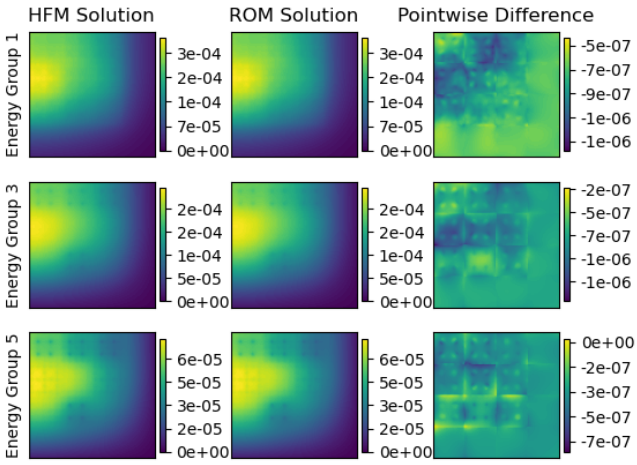
UOX-CR	MOX-CR	MOX-CR	Reflector
MOX-GT	UOX-GT	MOX-GT	Reflector
UOX-GT	UOX-CR	UOX-GT	Reflector
Reflector	Reflector	Reflector	Reflector

(b) Reactor test configuration showing actual sub-domain regions. Each sub-domains contains either UOX fuel assembly, MOX fuel assembly or Reflector. Each fuel assembly contains either guide tubes (GT) or control rods (CR).

Figure 7.28: Reactor test configurations for both parameters and the corresponding sub-domain material properties.



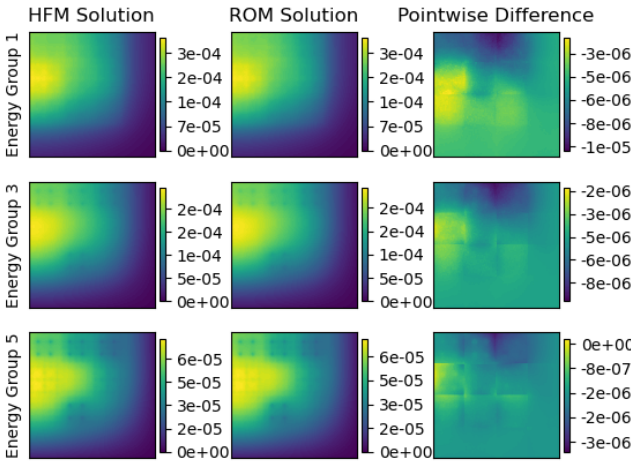
(a) Scalar flux (neutrons $\mathrm{cm}^{-2}\,\mathrm{s}^{-1}$) across the reactor core for three energy groups for HFM and the POD-FA-50var ROM and a pointwise difference between the two.



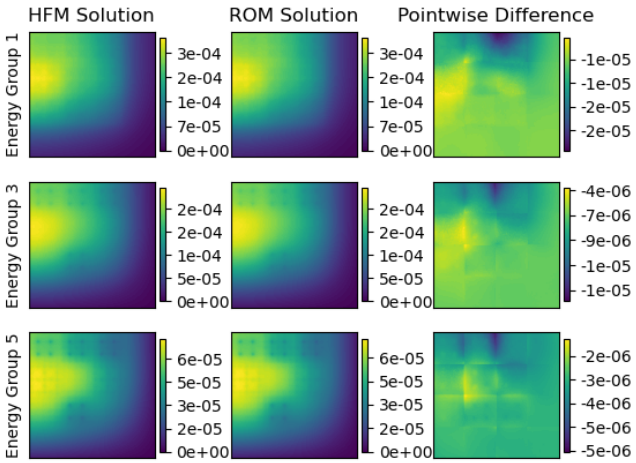
(b) Scalar flux (neutrons $\mathrm{cm}^{-2}\,\mathrm{s}^{-1}$) across the reactor core for three energy groups for HFM and POD-FA-50var-Alt ROM and a pointwise difference between the two.

Figure 7.29: Fuel assembly flux and pointwise comparison for a reactor core solution using POD-FA-50var and POD-FA-50var-Alt with the test configuration.

variables. It can be observed that the POD methods converge to a k_{eff} that is closer to the HFM than the AE methods. The AE methods also show slower convergence than the POD



(a) Scalar flux (neutrons $\mathrm{cm}^{-2}\,\mathrm{s}^{-1}$) across the reactor core for three energy groups for HFM and the AE-FA-50var ROM and a pointwise difference between the two.



(b) Scalar flux (neutrons $\mathrm{cm}^{-2}\,\mathrm{s}^{-1}$) across the reactor core for three energy groups for HFM and AE-FA-50var-Alt ROM and a pointwise difference between the two.

Figure 7.30: Fuel assembly flux and pointwise comparison for a reactor core solution using AE-FA-50var and AE-FA-50var-Alt with the test configuration.

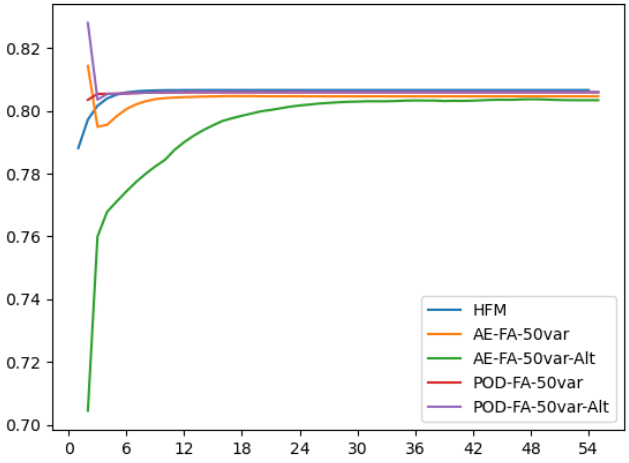
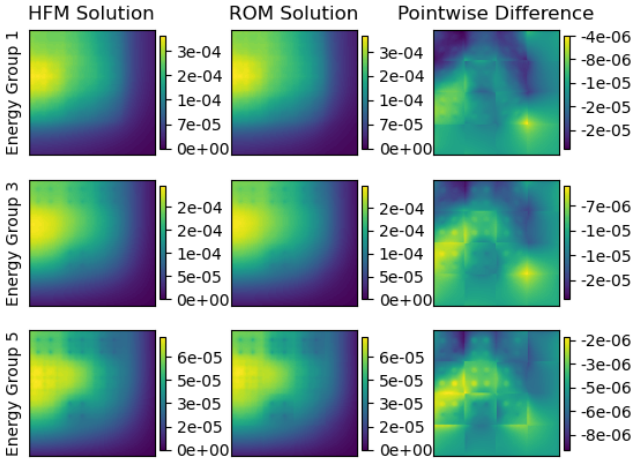


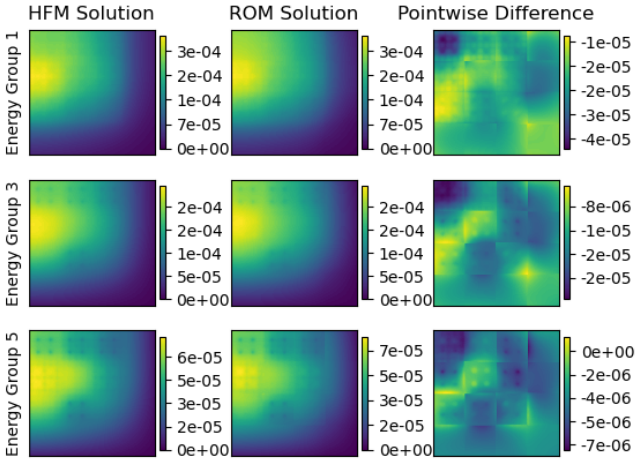
Figure 7.31: k_{eff} vs power iteration for the test configuration. The converged k_{eff} for each method is as follows: HFM - **0.80659**, POD-FA-50var - **0.80588**, POD-FA-50var-Alt - **0.80597**, AE-FA-50var - **0.80460**, AE-FA-50var-Alt -**0.80338**.

methods.

Figure 7.32 shows the Scalar flux profiles for the POD-FA-10var and POD-FA-10var-Alt ROMs and the pointwise comparison with the HFM. The highest errors are at the sub-domain interfaces, apparent between fuel assemblies.

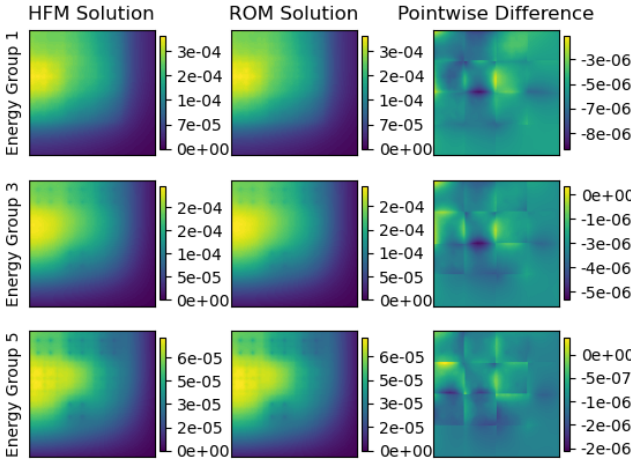


(a) Scalar flux (neutrons $\mathrm{cm}^{-2}\,\mathrm{s}^{-1}$) across the reactor core for three energy groups for HFM and the POD-FA-10var ROM and a pointwise difference between the two.

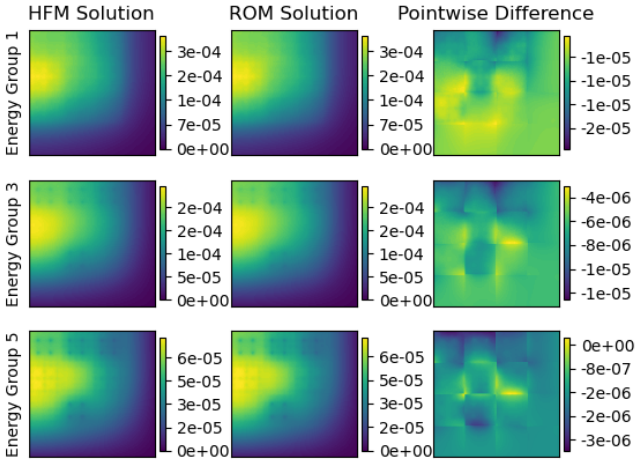


(b) Scalar flux (neutrons $\mathrm{cm}^{-2}\,\mathrm{s}^{-1}$) across the reactor core for three energy groups for HFM and POD-FA-10var-Alt ROM and a pointwise difference between the two.

Figure 7.32: Fuel assembly flux and pointwise comparison for a reactor core solution using POD-FA-10var and POD-FA-10var-Alt with the test configuration.



(a) Scalar flux (neutrons $\mathrm{cm}^{-2}\,\mathrm{s}^{-1}$) across the reactor core for three energy groups for HFM and the AE-FA-10var ROM and a pointwise difference between the two.



(b) Scalar flux (neutrons $\mathrm{cm}^{-2}\,\mathrm{s}^{-1}$) across the reactor core for three energy groups for HFM and AE-FA-10var-Alt ROM and a pointwise difference between the two.

Figure 7.33: Fuel assembly flux and pointwise comparison for a reactor core solution using AE-FA-10var and AE-FA-10var-Alt with the test configuration.

Figure 7.33 shows the Scalar flux profiles for the AE-FA-10var and AE-FA-10var-Alt ROMs and the pointwise comparison with the HFM. Again the highest errors can be seen at the sub-domain interfaces, although they are slightly lower than the POD solution in figure 7.32.

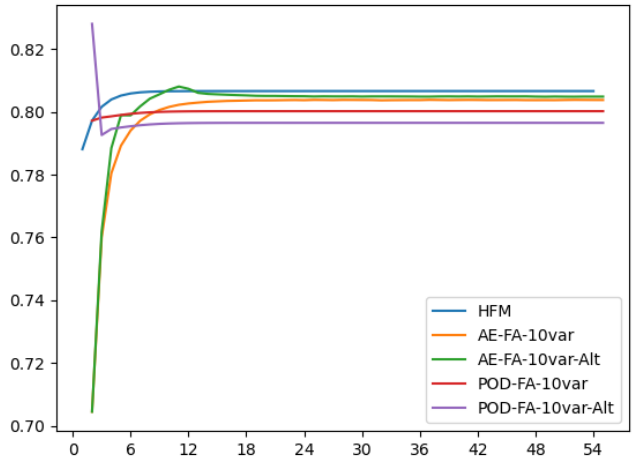
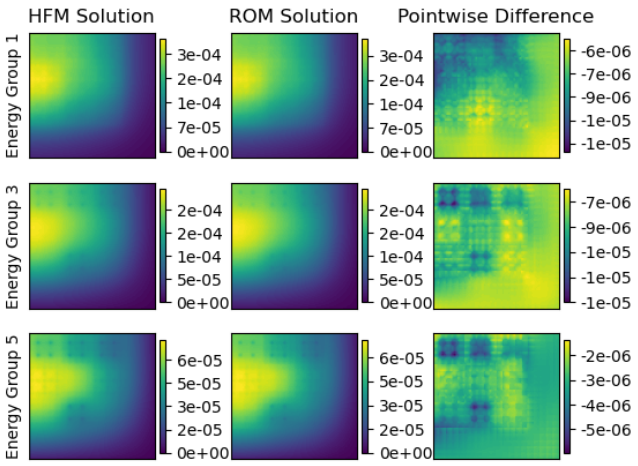
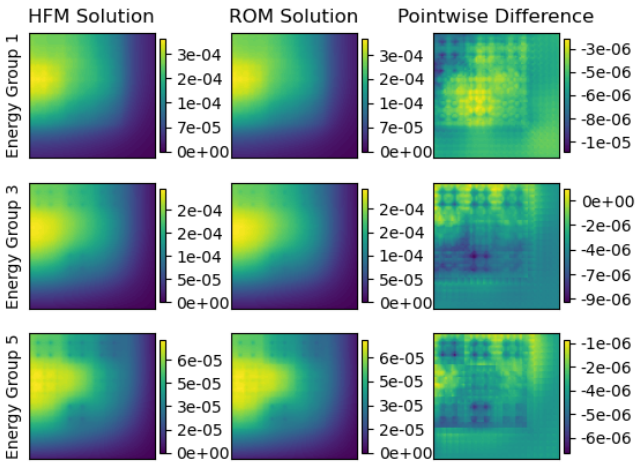


Figure 7.34: k_{eff} vs power iteration for the test configuration. The converged k_{eff} for each method is as follows: HFM - **0.80659**, POD-FA-10var - **0.80019**, POD-FA-10var-Alt - **0.79648**, AE-FA-10var - **0.80371**, AE-FA-10var-Alt -**0.80479**.

Figure 7.34 shows the k_{eff} for all methods compressing the fuel assemblies to 10 variables. It can be seen that the POD methods converge to a k_{eff} that is lower than the HFM, with the AE methods being closer.



(a) Scalar flux (neutrons $\text{cm}^{-2}\,\text{s}^{-1}$) across the reactor core for three energy groups for HFM and the POD-FR-10var ROM and a pointwise difference between the two.

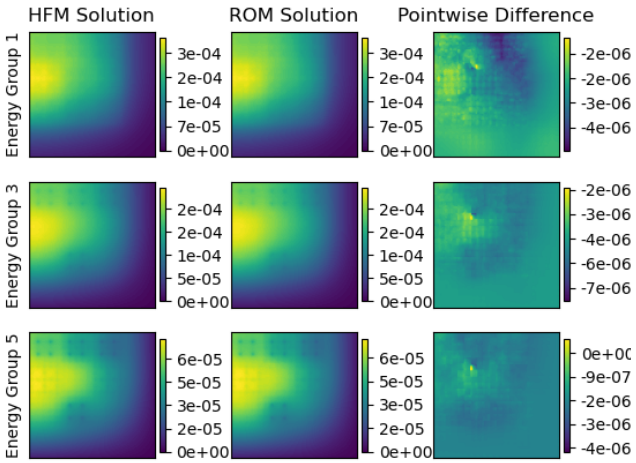


(b) Scalar flux (neutrons $\text{cm}^{-2}\,\text{s}^{-1}$) across the reactor core for three energy groups for HFM and POD-FR-10var-Alt ROM and a pointwise difference between the two.

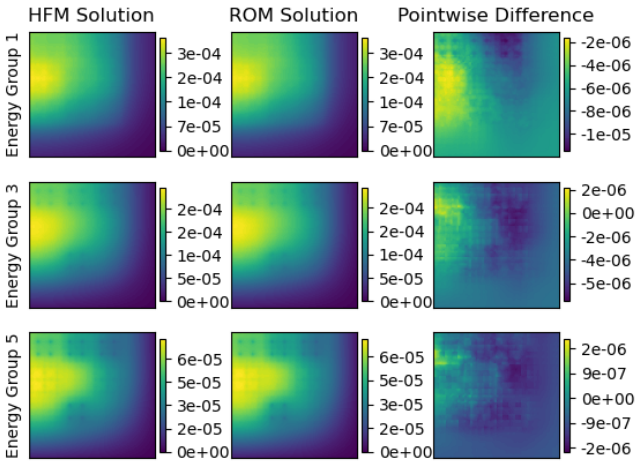
Figure 7.35: Fuel assembly flux and pointwise comparison for a reactor core solution using POD-FR-10var and POD-FR-10var-Alt with the test configuration.

Figure 7.35 shows the Scalar flux profiles for the POD-FR-10var and POD-FR-10var-Alt ROMs

and the pointwise comparison with the HFM. The errors can be seen to be largest in the control rod and guide-tube region, apparent at the four locations in each fuel assembly.



(a) Scalar flux (neutrons $\mathrm{cm}^{-2}\,\mathrm{s}^{-1}$) across the reactor core for three energy groups for HFM and the AE-FR-10var ROM and a pointwise difference between the two.



(b) Scalar flux (neutrons $\mathrm{cm}^{-2}\,\mathrm{s}^{-1}$) across the reactor core for three energy groups for HFM and AE-FR-10var-Alt ROM and a pointwise difference between the two.

Figure 7.36: Fuel assembly flux and pointwise comparison for a reactor core solution using AE-FR-10var and AE-FR-10var-Alt with the test configuration.

Figure 7.36 shows the Scalar flux profiles for the AE-FR-10var and AE-FR-10var-Alt ROMs and the pointwise comparison with the HFM. For both these methods the highest error occurs on one specific sub-domain, at the top and the second from the right. This sub-domain contains UOX-CR material properties.

Figure 7.37 contains the k_{eff} for each method compressing the fuel rods to 10 variables. All methods show slower convergence than the fuel assembly solutions. The AE methods converge to a k_{eff} that is closer to the HFM than the POD methods.

Figure 7.38 shows the boxplots for the maximum errors in the flux profile for all methods. It can be observed that typically the AE methods have a higher range than the POD methods. For the fuel assembly compression to 50 variables, the POD methods outperform the AE methods, achieving a lower median. For the fuel assembly compression to 10 variables and the fuel rod compression to 10 variables, the AE methods achieve a slightly lower median than the POD methods.

Figure 7.39 shows the boxplots for the errors in k_{eff} for all methods. Again, it can be observed

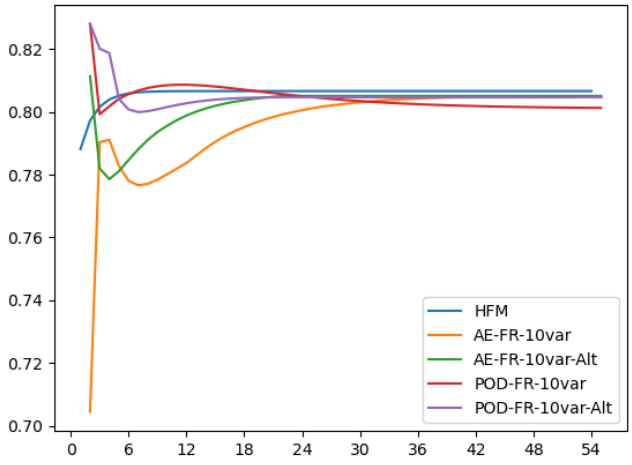
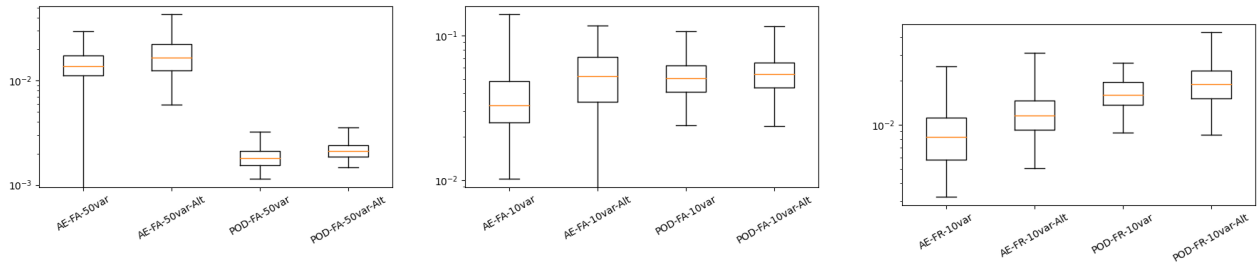


Figure 7.37: k_{eff} vs power iteration for the test configuration. The converged k_{eff} for each method is as follows: HFM - **0.80659**, POD-FR-10var - **0.80123**, POD-FR-10var-Alt - **0.80466**, AE-FR-10var - **0.80472**, AE-FR-10var-Alt -**0.80503**.



(a) Boxplot of the error in the flux profile using the ROMs that compress the fuel assembly to 50 variables compared with the HFM.

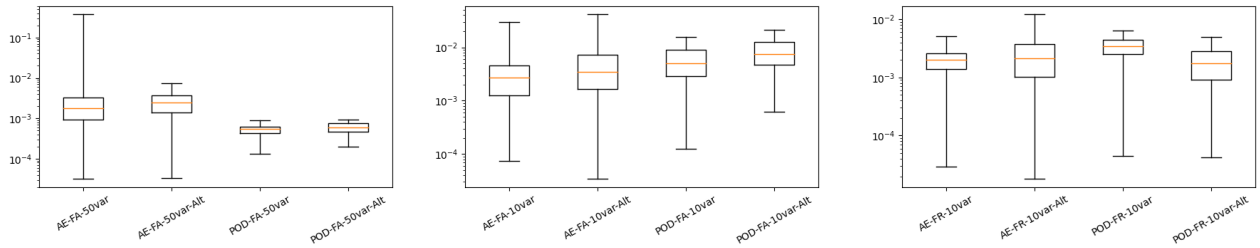
(b) Boxplot of the error in the flux profile using the ROMs that compress the fuel assembly to 10 variables compared with the HFM.

(c) Boxplot of the error in the flux profile using the ROMs that compress the fuel rods to 10 variables compared with the HFM.

Figure 7.38: Boxplot of the error in the flux profile solution generated using Autoencoders trained on the standard and alternative datasets and POD basis functions generated from the standard and alternative datasets compared with the HFM. The median is given by the orange line, the interquartile ranges by the box and the minimum and maximum values by whiskers.

that typically the AE methods have a higher range than the POD methods. For the fuel assembly compression to 50 variables, the POD methods outperform the AE methods, achieving a lower median. For the fuel assembly compression to 10 variables and the fuel rod compression to 10 variables, the AE methods achieve a slightly lower median than the POD methods.

Table 7.7 contains the mean maximum flux errors and k_{eff} errors for all methods. It can be observed that the errors for the POD methods compressing the fuel assembly to 50 variables are



(a) Boxplot of the error in k_{eff} using the ROMs that compress the fuel assembly to 50 variables compared with the HFM.

(b) Boxplot of the error in k_{eff} using the ROMs that compress the fuel assembly to 10 variables compared with the HFM.

(c) Boxplot of the error in k_{eff} using the ROMs that compress the fuel rods to 10 variables compared with the HFM.

Figure 7.39: Boxplot of the error in the flux profile solution generated using Autoencoders trained on the standard and alternative datasets and POD basis functions generated from the standard and alternative datasets compared with the HFM. The median is given by the orange line, the interquartile ranges by the box and the minimum and maximum values by whiskers.

FA-50Var	**AE-FA-50Var**	**AE-FA-50Var-Alt**	**POD-FA-50Var**	**POD-FA-50Var-Alt**
Flux Error	1.457×10^{-2}	1.794×10^{-2}	1.842×10^{-3}	2.175×10^{-3}
k_{eff} Error	5.764×10^{-3}	2.734×10^{-3}	5.432×10^{-4}	6.012×10^{-4}
FA-10Var	**AE-FA-10Var**	**AE-FA-10Var-Alt**	**POD-FA-10Var**	**POD-FA-10Var-Alt**
Flux Error	4.004×10^{-2}	5.342×10^{-2}	5.312×10^{-2}	5.624×10^{-2}
k_{eff} Error	3.692×10^{-3}	5.471×10^{-3}	6.029×10^{-3}	8.551×10^{-3}
FR-10Var	**AE-FR-10Var**	**AE-FR-10Var-Alt**	**POD-FR-10Var**	**POD-FR-10Var-Alt**
Flux Error	9.211×10^{-3}	1.251×10^{-2}	1.666×10^{-2}	1.982×10^{-2}
k_{eff} Error	2.079×10^{-3}	2.603×10^{-3}	3.341×10^{-3}	1.898×10^{-3}

Table 7.7: Mean maximum errors in the flux profile and k_{eff} errors for solutions generated using AEs and POD ROMs, compared with the HFM.

about an order of magnitude better than the AE methods. For the methods compressing the fuel assembly to 10 variables and the fuel rods to 10 variables, all methods perform similarly, within the same order of magnitude. The alternative dataset has similar errors to the corresponding method using a standard dataset for all methods. This indicates that an alternative dataset could be used to reduce the offline computational cost.

7.5 Summary

For the 1D problem, the accuracy of the solution was in the same order of magnitude when a total of 250 basis functions were used, irrespective of the domain decomposition methods used. This means that domain decomposition methods and locally formed solutions can be used with POD without a significant loss in accuracy. This is important, as it avoids using high-fidelity models for full reactor simulations to form the snapshots. These high-fidelity simulations may be computationally expensive or even intractable.

It was found that when the basis functions were constructed from the same snapshots as the global system, the accuracy was similar whether they were constructed based on the type, being UOX, MOX or reflector sub-domains, or location of the data, being the position of the sub-domain within the system. This has implications when considering large-scale problems where sub-domains with the same material properties and geometry repeat themselves. When the basis functions are constructed from the locations of sub-domains, the number of sets of basis functions required increases if the number of sub-domains increases. However, when basis functions are constructed based on the material type of sub-domains, assuming that new sub-domains have the same material types as existing ones, the required number of basis functions does not increase. This can be beneficial if domains have many degrees of freedom, singular-value decomposition becomes computationally expensive, and similar sub-domains are repeated within the system.

It was also found that a much smaller data set, constructed using a set of snapshots that were significantly less expensive to generate, could be used to generate basis functions where the solutions were only slightly less accurate than using the snapshots generated over the full solution. This means that the offline computational cost could be significantly decreased by being able to generate basis functions from smaller snapshots. This also means that the problem could be scaled up or modifications made to the geometry without requiring more offline computational work.

It was also found that when the number of basis functions retained in the global method was 50, it performed significantly worse than when domain decomposition methods were used. Although the number of POD coefficients used globally is less than the domain decomposition

methods, the information retained by the global method is the same as the five sets of basis functions because the basis functions for the global method are a matrix of 9350×50. For the domain decomposition methods, it is five matrices of size 1870×50. The total information in these matrices is the same. For the domain decomposition method, the global basis functions is a matrix of size 9350×250, but a significant portion of this matrix is zeros and, as such, can be ignored in computational costs.

Three of the methods presented were selected for additional testing. These three methods were G-POD, DDL-POD and DDT-POD. The first of these tests was to limit the number of snapshots available to the methods. All three methods had lower accuracy than when the full range of snapshots was utilised. The global method was greatly affected by the number of basis functions it could retain and had significantly worse accuracy. The domain decomposition methods were less limited by this and could use more basis functions than the global method, but still had a slight loss in accuracy compared to when all snapshots were used. It was found that the methods utilising domain decomposition performed better than the global method for a limited number of snapshots. This is beneficial if it is either difficult or time-consuming to produce a large number of snapshots.

The second test performed on the three methods, G-POD, DDL-POD and DDT-POD, was the smart selection of the number of basis functions. This was done by determining the number of basis functions based on the energy retained rather than a fixed number. It was found that all methods performed similarly but had different numbers of basis functions to reach the same fraction of energy retained. The global method required the smallest total number of basis functions, followed by the method based on the type of domain, and finally, the method based on location required the most basis functions. POD methods combined with domain decomposition produced more accurate solutions with fewer basis functions when the sets of basis functions were constructed based on the material type of domain rather than the location. Although the global method required the smallest total number of basis functions, each of these basis functions spans the whole computational domain. The domain decomposition basis functions only span a single sub-domain, thereby being one-fifth the size of the ones used in the global method.

The accuracy of these methods was largely independent of whether sets of basis functions were constructed based on the material type, location or both of the sub-domains. This means that

sets of basis functions could be repeated in the global system for sub-domains with similar material properties. For a reactor, where sub-domains with the same material parameters appear several times, this could significantly improve optimisation by enabling rapid solutions to changes in the geometry, such as the layout or amount of fuel assemblies, without an increase in the offline computational cost. It was found that basis functions could be generated from a set of snapshots that were not generated from the full HFM without a significant loss in accuracy. This can significantly reduce the offline computational cost to create the basis functions, one of the current limitations of ROM.

Additional testing of the methods found that when the number of snapshots used to construct the ROMs was limited, all three methods suffered a loss in accuracy, but the global method had a more significant reduction in accuracy than either domain decomposition method used. If the number of initial snapshots is limited, it would be beneficial to use domain decomposition methods instead. An additional test was performed by a smart selection of a number of basis functions, where this was chosen based on the fraction of energy retained. It was found that all three methods performed with similar levels of accuracy, despite the different number of basis functions used with each method. For the domain decomposition methods, constructing the sets of basis functions from the material type of domain produced slightly more accurate results with fewer basis functions than when they were constructed from the location of the domain.

These methods were extended to a 2D problem and compared with using autoencoders. It was observed that the POD methods outperform the AE methods when a higher number of variables are used. However, when this is reduced, the AE methods perform similarly to the POD methods. An alternative dataset is also presented and constructed using smaller snapshots in the offline phase. This alternative dataset can be utilised with POD without any changes. The AE method requires a scaling factor to be determined, generated by comparing a subset of the alternative dataset with the full snapshots. This scaling factor brings the AE method errors to a similar value as the standard data. This allows a partially reduced offline cost for the AE methods.

For modelling nuclear reactors, combining both ROM methods with domain decomposition methods can have considerable benefits. The potential to enable parallel computing while retaining the same level of accuracy means that the time taken to produce solutions could be significantly reduced, which can help real-time analysis be performed. A realistic reactor core

typically contains many fuel assemblies, such as the 193 in Sizewell B, and by utilising domain decomposition all of these fuel assemblies could be resolved simultaneously, potentially allowing real-time safety analysis.

A drawback to ROM is the offline computational cost of producing snapshots, and this could be offset by producing smaller snapshots (with fewer degrees of freedom within each snapshot) during the offline phase to be used in the online phase. It is seen here that there is a slight loss in accuracy from using smaller snapshots, but the snapshots generated during the offline phase are significantly smaller than using the global system. In addition to this, the neural networks trained for dimensionality reduction on the sub-domain data have significantly fewer trainable parameters than an equivalent network trained on the global system would have.

Optimising the design of reactors can also benefit from ROM combined with domain decomposition. By constructing basis functions for each material type of sub-domain, where similar sub-domains repeat within the global system, then an optimum geometry for the arrangement of those sub-domains can be determined quickly and with no increase in the offline computational cost. Typical ROM methods require a fixed geometry that cannot be varied or incur further offline costs, but using domain decomposition means this can be avoided. Therefore, the number of design parameters that can be varied for ROM methods is increased when utilising multiple ROMs in sub-domains.

Future work will apply the ROMs based on the domain decomposition proposed here to progressively more realistic and challenging problems. This includes increasing the spatial dimension to 3D, using transport theory instead of diffusion theory and using many more energy groups than the 22 used here. The behaviour seen in these problems will be closer to that of real reactors, and their modelling will benefit more from reducing the computational cost that the domain decomposition ROMs can provide.

Chapter 8

Hiearchical ROM

Chapter 2 contains the methods used to generate the high-fidelity snapshots used in this chapter. Proper-Orthogonal decomposition methods are described in chapter 5.

8.1 Introduction

Chapter 7 investigates how the internal structure of a reactor can be broken down into several levels, consisting of fuel assemblies and fuel rods. It primarily assesses how domain decomposition can be utilised with ROMs, assessing how different subsets of snapshots can be utilised. One of the essential conclusions of this chapter is how a single ROM can be used to resolve multiple sub-domains of the global problem. Utilising ROMs in this way benefits the offline phase, reducing the computational cost required to produce all of the necessary ROMs.

Proper-Orthogonal Decomposition (POD) has seen great success utilising interpolation methods, which have been generally utilised to form Non-intrusive ROMs (NIROMs). In [50] and [73], the basis functions generated from POD are interpolated using linear interpolation of the initial conditions. As demonstrated by [190], Proper-Orthogonal Decomposition can be used in an interpolation method where the POD coefficients from the preceding time-step are used to interpolate those of the next time step. A non-linear interpolation method is applied to a reactor problem in [86], which utilised different basis function sets for different control rod heights, interpolating between the sets for unseen control rod heights.

Chapter 7 utilises the same type of reduce-order models for every sub-domain in each method. However, separating a global domain into sub-domains, and applying a ROM to each sub-domain, opens up several possibilities. Because each solution for a sub-domain is resolved independently, different ROMs, such as POD and AE-based, could be combined within a single solution. This may be useful when an AE can capture the behaviour in specific sub-domains more effectively than POD.

Another possible method is implementing non-intrusive and projection-based methods for different sub-domains within a single solution. This may be performed by interpolating between sub-domains, using the ROMs in neighbouring sub-domains as the base for interpolation. Combining methods like this has the potential for retaining the higher accuracy of the projection-based methods while incorporating some of the computational cost reduction of the non-intrusive methods. As previously stated, an actual reactor can have hundreds of fuel assemblies and thousands of fuel rods, which means that ROMs modelling one of these could be used hundreds or thousands of times within a solution. Reducing the number of these that need to be resolved with projection-based methods through interpolation would significantly decrease the computational cost.

This chapter investigates a potential method for forming a hierarchy of reduced-order models using interpolation. An interpolation method is tested on a fuel assembly, separating the domain into several sub-domains with the same material parameters, meaning the same ROM can be used for all sub-domains. These methods are then extended to a reactor core, where a multi-level interpolation forms a hierarchy of ROMs. In addition to containing all the typical benefits of reduced-order models, the additional interpolation results in less than half the latent variables as a standard domain decomposition POD-based model.

The chapter is organised as follows: Section 8.2 contains the methodology for forming hierarchical methods with POD, section 8.3 contains an overview of the results and finally, a summary of the chapter is given in section 8.4.

8.2 Methodology

8.2.1 Determining matrices for sub-domains

A 1D problem with a global domain Ω is split into K sub-domains where:

$$\Omega = \Omega_1 \cup \Omega_2 \ldots \cup \Omega_K, \tag{8.1}$$

where each domain Ω_k is neighboured by domains Ω_{k-1} and Ω_{k+1}. Each domain Ω_k has associated matrices, $\boldsymbol{A_k}$ and $\boldsymbol{B_k}$, basis functions $\boldsymbol{R_k}$ and flux $\boldsymbol{\phi_k}$ associated with it. Assuming $K = 3$ then matrix $\boldsymbol{A}$ for domain Ω can be constructed from the matrices for individual domains by:

$$\boldsymbol{A_p} = \begin{bmatrix} \boldsymbol{A_{1,1}} & \boldsymbol{A_{1,2}} & 0 \\ \boldsymbol{A_{2,1}} & \boldsymbol{A_{2,2}} & \boldsymbol{A_{2,3}} \\ 0 & \boldsymbol{A_{3,2}} & \boldsymbol{A_{3,3}} \end{bmatrix}, \tag{8.2}$$

where the off-diagonal matrices represent the transport terms that cause interactions across domain interfaces. Matrix $\boldsymbol{B}$ and the basis functions $\boldsymbol{R}$ can be constructed by:

$$\boldsymbol{B} = \begin{bmatrix} \boldsymbol{B_1} & 0 & 0 \\ 0 & \boldsymbol{B_2} & 0 \\ 0 & 0 & \boldsymbol{B_3} \end{bmatrix}, \quad \boldsymbol{R} = \begin{bmatrix} \boldsymbol{R_1} & 0 & 0 \\ 0 & \boldsymbol{R_2} & 0 \\ 0 & 0 & \boldsymbol{R_3} \end{bmatrix}. \tag{8.3}$$

The flux in domain $\boldsymbol{\phi}$ and $\boldsymbol{\alpha}$ can be determined by concatenating the flux and POD coefficients respectively:

$$\boldsymbol{\phi} = \begin{bmatrix} \boldsymbol{\phi_1} \\ \boldsymbol{\phi_2} \\ \boldsymbol{\phi_3} \end{bmatrix}, \quad \boldsymbol{\alpha} = \begin{bmatrix} \boldsymbol{\alpha_1} \\ \boldsymbol{\alpha_2} \\ \boldsymbol{\alpha_3} \end{bmatrix}. \tag{8.4}$$

8.2.2 Determining the Interpolation Matrix

If Ω_1, Ω_2 and Ω_3 have similar material properties and the same basis functions are used in their domains, i.e $\boldsymbol{R_1} = \boldsymbol{R_2} = \boldsymbol{R_3}$, we can assume a linear interpolation between the POD coefficients in each domain:

$$\alpha_2 \approx \tilde{\alpha_2} = \frac{\alpha_1 + \alpha_3}{2}, \tag{8.5}$$

which can be used to reduce the number of POD coefficients solved in the system. Reducing the coefficients and recovering them can be determined by $\boldsymbol{M}$ and $\boldsymbol{M_I}$ respectively:

$$\boldsymbol{M} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad \boldsymbol{M_I} = \begin{bmatrix} 1 & 0 \\ \frac{1}{2} & \frac{1}{2} \\ 0 & 1 \end{bmatrix}, \tag{8.6}$$

such that:

$$\tilde{\boldsymbol{\alpha}} = \begin{bmatrix} \boldsymbol{\alpha_1} \\ \tilde{\boldsymbol{\alpha_2}} \\ \boldsymbol{\alpha_3} \end{bmatrix} = \boldsymbol{M_I} \begin{bmatrix} \boldsymbol{\alpha_1} \\ \boldsymbol{\alpha_3} \end{bmatrix} \tag{8.7}$$

.

8.2.3 Hierarchical Reduced-Order Model for Criticality with POD

Having established the Reduced-Order model using POD basis functions, the interpolation can be performed by inserting equation (8.7) into equation (5.10) and pre-multiplying by $\boldsymbol{M}$:

$$\boldsymbol{M R^T A R M_I \tilde{\alpha}} = \boldsymbol{\lambda M R^T B R M_I \tilde{\alpha}} \tag{8.8}$$

Assuming all K sub-domains have P latent variables and the number of sub-domains that can be interpolated is given by I the new system has size $(K-I)P \times (K-I)P$ matrices as opposed to the $KP \times KP$ of the standard POD model. This turns algorithm 3 into:

This method is referred to as Hierarchical POD or HPOD. Additional levels in the hierarchy can be established by interpolating on multiple levels by including multiple interpolation matrices,

Algorithm 6 POD-based hierarchical reduced-order model: inner iterations.

function HPOD_INNER_ITERATIONS($\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{\phi}$, λ, $\boldsymbol{R}$, $\boldsymbol{M}$, $\boldsymbol{M_I}$)
  $\boldsymbol{s} = \lambda \left(\boldsymbol{M R}^T \boldsymbol{B}\right) \boldsymbol{\phi}$! *set a source with values from the outer iterations*
  solve the reduced-order model for $\boldsymbol{\alpha}$:
  $\left(\boldsymbol{M R}^T \boldsymbol{A R M_I}\right) \boldsymbol{\alpha} = \boldsymbol{s}$
  $\boldsymbol{\phi} = \boldsymbol{R M_I \alpha}$! *find the updated scalar flux from the reduced variables*
  return $\boldsymbol{\phi}$

an example Hierarchical Reduced-Order model where fuel rods are interpolated on one level and fuel assemblies on another level:

$$\boldsymbol{M_2 M_1 R^T A R M_{I1} M_{I2} \tilde{\alpha}} = \boldsymbol{\lambda M_2 M_1 R^T B R M_{I1} M_{I2} \tilde{\alpha}}, \tag{8.9}$$

where $\boldsymbol{M_1}$ and $\boldsymbol{M_{I1}}$ are the interpolation matrices associated with fuel rods and $\boldsymbol{M_2}$ and $\boldsymbol{M_{I2}}$ are the interpolation matrices associated with the fuel assemblies.

8.2.4 Adjusted Hierarchical ROM

The HROM established in the previous section may be utilised in conjunction with the standard ROM to produce a more accurate solution. Algorithm 7 shows how algorithm 2 can be modified to start with the interpolated basis functions, using algorithm 6, and moving onto the standard basis functions, algorithm 3, after a certain k_{eff} tolerance has been achieved. This is referred to as the Adjusted Hierarchical POD or AHPOD.

8.2.5 Fuel Assembly Problem and Construction of Matrices

Figure 8.1 contains the geometry for a UOX fuel assembly. This consists of 15×15 lattice containing either UOX fuels rods, guide tubes or control rods. Each grid space contains 20×20 cells, forming a total of $90,000$ cells along with 1200 cells to enforce the boundary conditions. The energy was discretised into seven groups, meaning the fuel assembly has $631,200$ degrees of freedom. Each side of the fuel assembly is length 21.42cm meaning each cell is 0.0714cm $\times$ 0.0714cm. Each side also has Vacuum boundary conditions applied to it. The material cross-sections for this problem are the same as from the KAIST benchmark [149].

The cells containing guide-tubes or control rods can instead by a combination of the two. A mixing coefficient r can be used to determine the amount of control rod inserted, with the cor-

Algorithm 7 Adjusted Power method: outer iterations.

1: **function** OUTER_ITERATIONS($\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{\phi}^{\text{guess}}$, λ^{guess})
2: $\boldsymbol{\phi}^{(0)} = \boldsymbol{\phi}^{\text{guess}}$, $\lambda^{(0)} = \lambda^{\text{guess}}$
3: $i\_max = 200$
4: $k\_tol = 10^{-6}$
5: $k\_adj = 10^{-3}$
6: $i = 0$
7: `not_converged = True`
8: **while** `not_converged` **do**
9: **if** $|k_{\text{eff}}^{(i)} - k_{\text{eff}}^{(i-1)}| > k\_adj$) **then**
10: $\boldsymbol{\phi}^{(i+1)}$ = HPOD_INNER_ITERATIONS($\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{\phi}^{(i)}, \lambda^{(i)}, \boldsymbol{RM}, \boldsymbol{M_I}$)
! *to solve Equation (*(8.8)*)*
11: **else**
12: $\boldsymbol{\phi}^{(i+1)}$ = POD_INNER_ITERATIONS($\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{\phi}^{(i}, \lambda^{(i)}, \boldsymbol{R}$)
! *to solve Equation (*(5.10)*)*
13: $\boldsymbol{\phi}^{(i+1)} \leftarrow \dfrac{\boldsymbol{\phi}^{(i+1)}}{\boldsymbol{b}^T \boldsymbol{B} \boldsymbol{\phi}^{(i+1)}}$! *normalising the flux*
14: $\lambda^{(i+1)} = \dfrac{\boldsymbol{b}^T \boldsymbol{A} \boldsymbol{\phi}^{(i+1)}}{\boldsymbol{b}^T \boldsymbol{B} \boldsymbol{\phi}^{(i+1)}}$
15: **if** ($i = i\_max$ **or** $|k_{\text{eff}}^{(i+1)} - k_{\text{eff}}^{(i)}| < k\_tol$) **then**
16: `not_converged = False`
17: $\boldsymbol{\phi} \leftarrow \boldsymbol{\phi}^{(i+1)}$
18: $\lambda \leftarrow \lambda^{(i+1)}$
19: **else**
20: $i \leftarrow i + 1$
21: **return** $\boldsymbol{\phi}$, λ

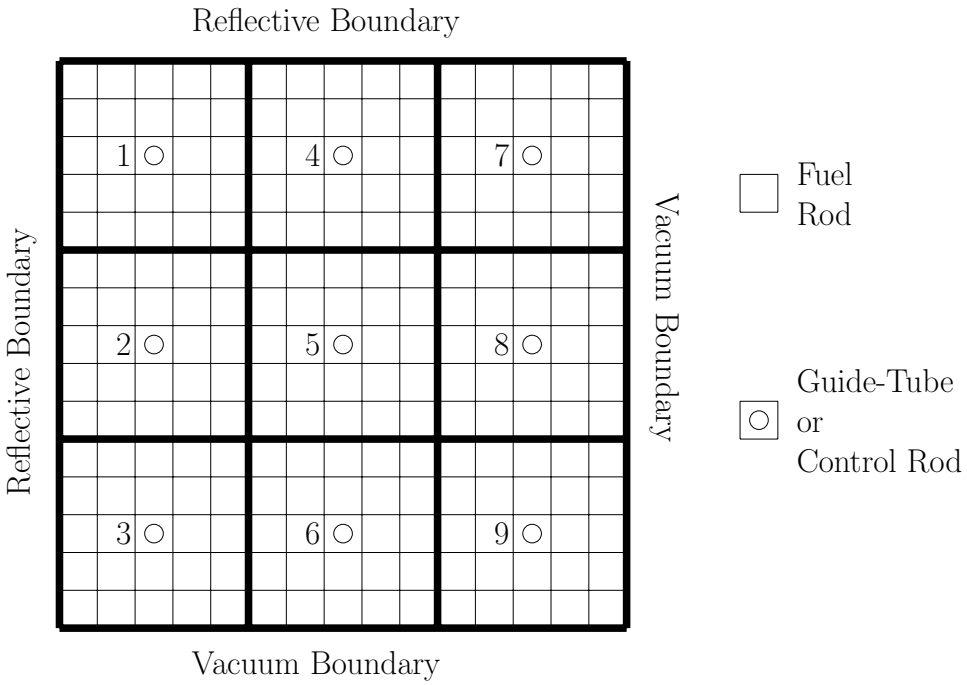


Figure 8.1: Geometry of UOX fuel assembly.

responding scattering cross sections Σ^s_{hom} and absorption cross sections Σ^a_{hom} being determined by a mixture of the two:

$$\Sigma^a_{\text{hom}} = r\Sigma^a_{\text{gt}} + (1-r)\Sigma^a_{\text{cr}}, \tag{8.10}$$

$$\Sigma^s_{\text{hom}} = r\Sigma^s_{\text{gt}} + (1-r)\Sigma^s_{\text{cr}}, \tag{8.11}$$

where gt represents the guide-tube cross sections and cr represents the control rod cross sections. Twenty HFM snapshots are generated with different r values, with a single r value dictating the cross-sections for every control rod for a single snapshot.

In figure 8.1, Nine separate sub-domains are indicated by the bold outlines with a number inside it. Each sub-domains contains the same material parameters inside, a 5 grid of fuel rods with a control rod inside. Each HFM solutions sub-domains forms the snapshots, 180 from the 20 solutions, used to construct the POD basis functions $\boldsymbol{R_{UOX}}$. Assuming 20 basis functions are retained, resulting in 20 POD coefficients for each sub-domains or 180 total for the whole fuel assembly. These POD coefficients are concatenated in order, and basis functions are repeated such that:

$$\boldsymbol{\alpha} = \begin{bmatrix} \boldsymbol{\alpha_1} \\ \boldsymbol{\alpha_2} \\ \cdots \\ \boldsymbol{\alpha_9} \end{bmatrix}, \quad \boldsymbol{R} = \begin{bmatrix} \boldsymbol{R_{UOX}} & 0 & \cdots & 0 \\ 0 & \boldsymbol{R_{UOX}} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & \boldsymbol{R_{UOX}} \end{bmatrix}, \tag{8.12}$$

where the underscore corresponds with the sub-domain number. The sub-domains can be interpolated as follows:

$$\tilde{\Omega_2} = \frac{\Omega_1 + \Omega_3}{2} \tag{8.13}$$

$$\tilde{\Omega_4} = \frac{\Omega_1 + \Omega_7}{2} \tag{8.14}$$

$$\tilde{\Omega_6} = \frac{\Omega_3 + \Omega_9}{2} \tag{8.15}$$

$$\tilde{\Omega_8} = \frac{\Omega_7 + \Omega_9}{2} \tag{8.16}$$

$$\tilde{\Omega_5} = \frac{\Omega_1 + \Omega_3 + \Omega_7 + \Omega_9}{4} \tag{8.17}$$

Resulting in the following interpolation matrices:

$$\boldsymbol{M} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \quad \boldsymbol{M_I} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 & 0 \\ 0 & 1 & 0 & 0 \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 \\ \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} \\ 0 & \frac{1}{2} & 0 & \frac{1}{2} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{8.18}$$

reducing the number of total POD coefficients in the system from 90 to 40.

8.2.6 Reactor Core problem

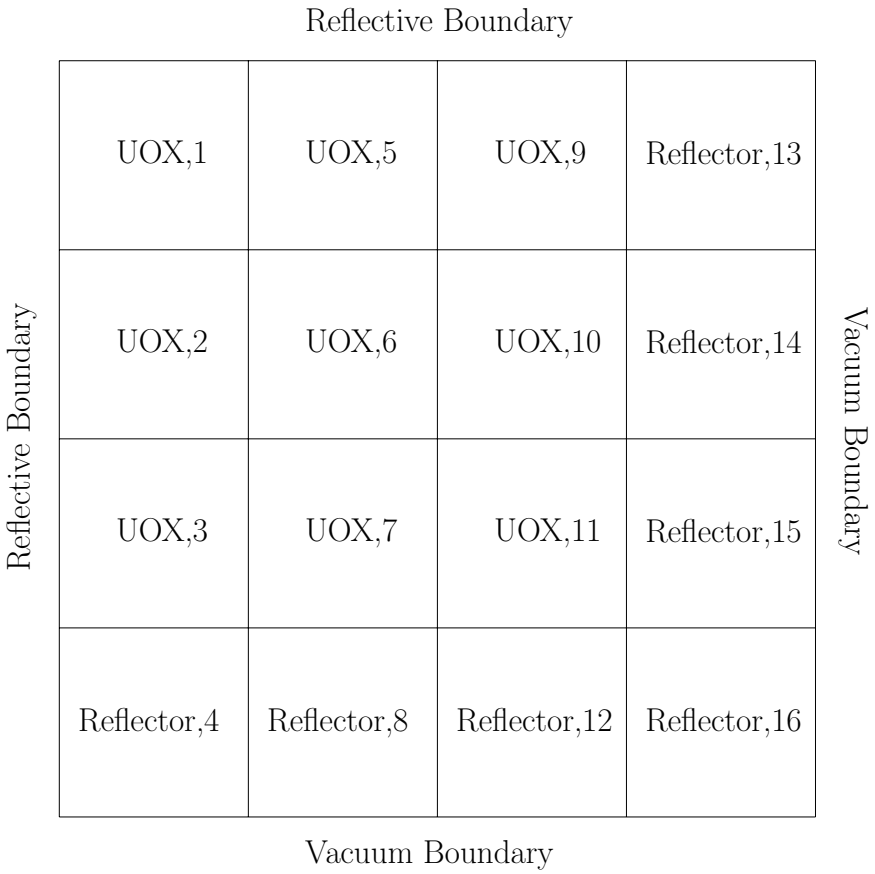


Figure 8.2: Mesh of Reactor Core.

Figure 8.2 contains the geometry for a quarter of a reactor core. This contains a 3×3 grid of fuel assemblies. Each of these are a UOX fuel assembly, with the same configuration and information shown in section 8.2.5. The upper and left sides have reflective boundary conditions applied and the right and bottom sides have vacuum boundary conditions applied, meaning the whole geometry represents the bottom right quarter of a reactor. The width of the reflector is also 21.42cm so the each side of the geometry is 85.68cm. The mesh remains uniform in the reflector, meaning the number of cells along the width of the reflector is the same as the number of cells along the width of a fuel assembly. The reflector, therefore, contains $630,000$ cells, all nine fuel assemblies contain a total of $810,000$ cells and a total of 4804 cells are used for boundary conditions. The energy was again discretised into seven groups resulting in $10,113,628$ degrees of freedom. These sub-domains can be interpolated in the following ways:

$$\tilde{\Omega_2} = \frac{\Omega_1 + \Omega_3}{2}, \tag{8.19}$$

$$\tilde{\Omega_5} = \frac{\Omega_1 + \Omega_9}{2}, \tag{8.20}$$

$$\tilde{\Omega_7} = \frac{\Omega_3 + \Omega_{11}}{2}, \tag{8.21}$$

$$\tilde{\Omega_{10}} = \frac{\Omega_9 + \Omega_{11}}{2}, \tag{8.22}$$

$$\tilde{\Omega_6} = \frac{\Omega_1 + \Omega_3 + \Omega_9 + \Omega_{11}}{4}, \tag{8.23}$$

$$\tilde{\Omega_8} = \frac{2\Omega_4 + \Omega_{16}}{3}, \tag{8.24}$$

$$\tilde{\Omega_{12}} = \frac{\Omega_4 + 2\Omega_{16}}{3}, \tag{8.25}$$

$$\tilde{\Omega_{14}} = \frac{2\Omega_{13} + \Omega_{16}}{3}, \tag{8.26}$$

$$\tilde{\Omega_{15}} = \frac{\Omega_{13} + 2\Omega_{16}}{3}, \tag{8.27}$$

Reducing the number of sub-domains being solved from 16 to 7. Each UOX fuel assembly can be decomposed the same way as given by figure 8.1, utilising the basis functions (R_{UOX}) previously used. The reflector sub-domains can be decomposed in a similar manner, resulting in another set of basis functions for the reflector region (R_{REF}).

8.3 Results

In this section, a brief description of the error measures used is given, followed by a look at the results generated from the fuel assembly and reactor core unseen cases.

8.3.1 Error Measures

Multiple error measures are used to assess the accuracy of the results. The first is the largest normalised maximum error in the flux profile and is determined by:

$$e_{\max}(\boldsymbol{\phi}^{\mathrm{POD}}) = \frac{\phi_j^{\mathrm{HFM}} - \phi_j^{\mathrm{POD}}}{\|\boldsymbol{\phi}^{\mathrm{HFM}}\|_\infty} \quad \text{where } j = \underset{i \in \text{cells}}{\operatorname{argmax}} \mid \phi_i^{\mathrm{HFM}} - \phi_i^{\mathrm{POD}} \mid , \tag{8.28}$$

in which the superscript "POD" indicates the solution was generated by POD and i and j represent cell indices.

The second is the error in k_{eff}, which is just the difference between the HFM and POD solutions:

$$e(k_{\text{eff}}^{\text{POD}}) = k_{\text{eff}}^{\text{HFM}} - k_{\text{eff}}^{\text{POD}}. \tag{8.29}$$

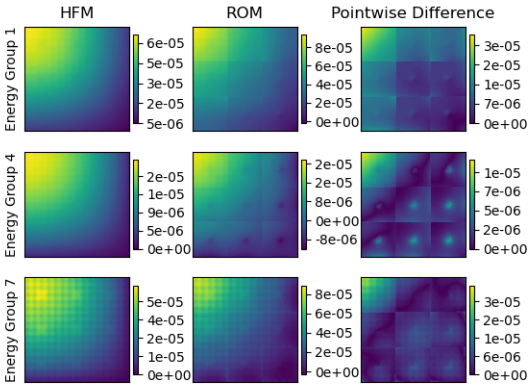
An average maximum error is also used, which for the error in the flux profile is given by:

$$\bar{e}_{\max}(\boldsymbol{\phi}^{\text{POD}}) = \frac{1}{N}\sum_{l=1}^{N} \left|\, e_{\max}(\boldsymbol{\phi}_l^{\text{POD}})\right|, \tag{8.30}$$

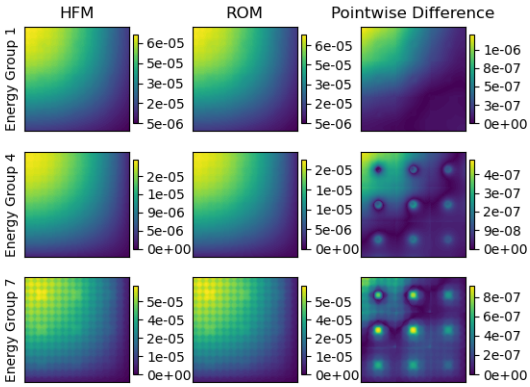
where N is the total number of snapshots.

8.3.2 Fuel Assembly Results

The fuel assembly is decomposed the way figure 8.1 describes, resulting in nine sub-domains. R_{UOX} has 20 basis functions retained, resulting in 180 POD coefficients for the POD method. Four sub-domains are resolved for the HPOD method, resulting in 80 POD coefficients. To test the method, 20 solutions are generated for the dataset used to construct the basis functions.



(a) Scalar flux (neutrons $\text{cm}^{-2}\,\text{s}^{-1}$) across the fuel assembly for three energy groups for HFM and HPOD and a pointwise difference between the two.



(b) Scalar flux (neutrons $\text{cm}^{-2}\,\text{s}^{-1}$) across the fuel assembly for three energy groups for HFM and AHPOD and a pointwise difference between the two.

Figure 8.3: Fuel assembly flux and pointwise comparison for a fuel assembly solution using HPOD and AHPOD with $r = 0.841$.

Figure 8.3 shows the flux produced by the HPOD and AHPOD methods compared with the HFM. It can be observed that the HPOD method shows larger errors in the upper left region,

where the larger flux is. The AHPOD method shows more accurate solutions when corrected using the full POD system, with most errors being at the control rod regions.

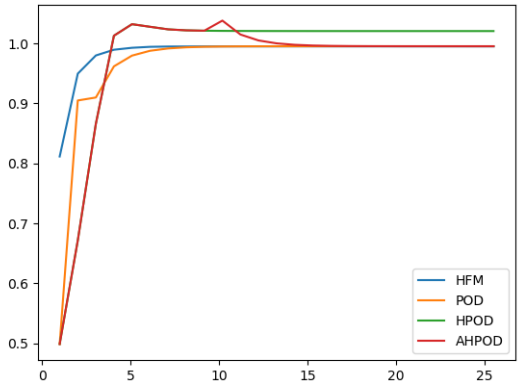


Figure 8.4: k_{eff} vs power iteration for a fuel assembly solution with $r = 0.841$. The converged k_{eff} for each method is as follows: HFM - **0.99492**, POD - **0.99517**, HPOD - **1.02043**, AHPOD - **0.99518**.

Figure 8.4 shows the k_{eff} for the HFM, POD, HPOD and AHPOD solutions. It can be observed that the POD and HPOD solutions have converged after 9 power iterations, at which point the AHPOD adjusts and converged after another 5 power iterations. Table 8.1 contains the

		POD	HPOD	AHPOD
Flux Error	$e_{\max}(\boldsymbol{\phi}^{\text{ROM}})$	8.261×10^{-3}	2.310×10^{-1}	8.312×10^{-3}
k_{eff} Error	k_{eff}	9.232×10^{-4}	2.564×10^{-2}	9.270×10^{-4}

Table 8.1: Average maximum errors for the seen dataset using POD, HPOD and AHPOD ROMs.

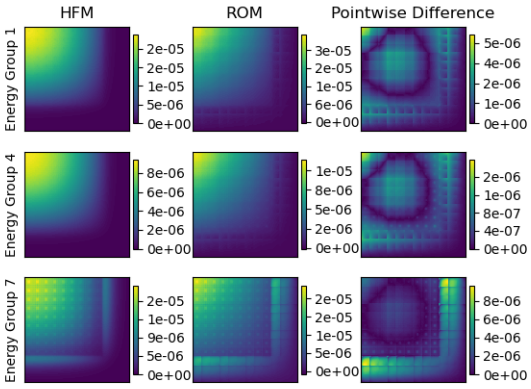
flux and k_{eff} errors for the POD, HPOD and AHPOD methods. It can be seen that the HPOD solutions have errors almost two orders of magnitude worse but when corrected in the AHPOD method, these errors reduce to the same magnitude as POD, as would be expected.

8.3.3 Reactor core Results

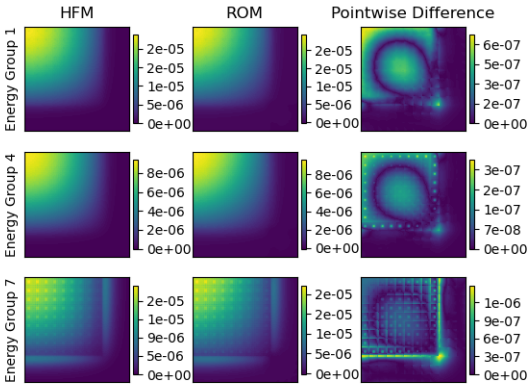
The reactor core consists of nine fuel assemblies, with a reflector region surrounding it. As indicated by figure 8.2 the total domain can be separated into a 4×4 grid of sub-domains. Each of these sub-domains can be further decomposed into the 3×3 grid of sub-domains indicated by 8.1. The reflector sub-domains are decomposed in the same manner. R_{UOX} and R_{REF} both

have 5 basis functions retained. There are 144 sub-domains in the system, resulting in 720 POD coefficients for the POD method. The HPOD method has two levels of interpolation, the first applies to each fuel assembly and the second applies to the global domain. This results in 28 sub-domains being resolved for the HPOD method, containing a total of 140 POD coefficients.

A total of 40 solutions are generated with random r values. These solutions are separated into two datasets: seen and unseen, with 20 solutions within each. Only solutions from the seen dataset are used to construct the basis functions.



(a) Scalar flux (neutrons $\mathrm{cm}^{-2}\,\mathrm{s}^{-1}$) across the reactor core for three energy groups for HFM and HPOD and a pointwise difference between the two.



(b) Scalar flux (neutrons $\mathrm{cm}^{-2}\,\mathrm{s}^{-1}$) across the reactor core for three energy groups for HFM and AHPOD and a pointwise difference between the two.

Figure 8.5: Fuel assembly flux and pointwise comparison for a reactor core solution using HPOD and AHPOD with seen case $r = 0.891$.

Figure 8.5 shows the flux produced by the HPOD and AHPOD methods compared with the HFM. It can be observed that the HPOD method shows larger errors in the reflector region, implying that the interpolation should not be used in this area. The AHPOD method shows more accurate solutions when it is corrected using the full POD system, with the errors largely being at the interface between fuel assemblies and reflector.

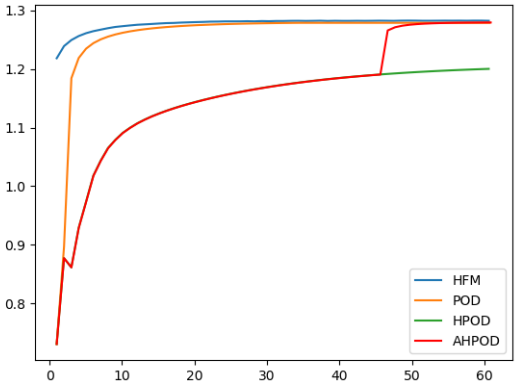
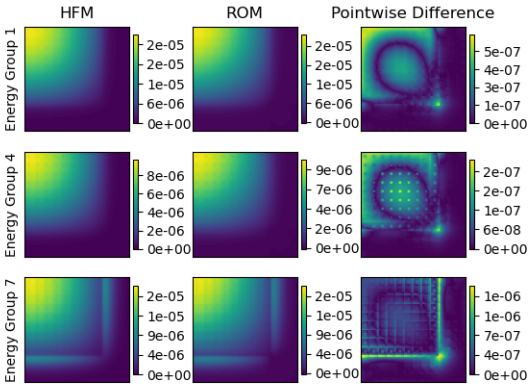
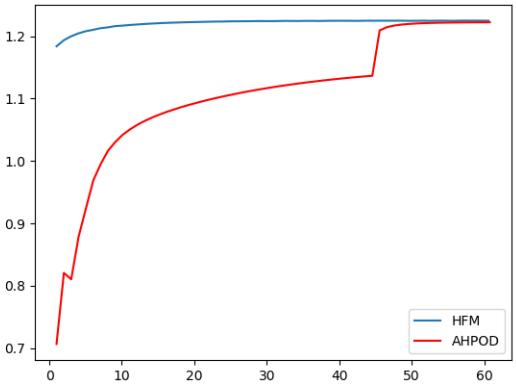


Figure 8.6: k_{eff} vs power iteration for a reactor core solution with seen case $r = 0.891$.The converged k_{eff} for each method is as follows: HFM - **1.28259**, POD - **1.27885**, HPOD - **1.20721**, AHPOD - **1.27975**.

Figure 8.6 shows the k_{eff} for the HFM, POD, HPOD and AHPOD solutions. It can be observed that the POD and HPOD solutions have converged after 45 power iterations, at which point the AHPOD adjusts and converges after another 15 power iterations.



(a) Scalar flux (neutrons $\text{cm}^{-2}\,\text{s}^{-1}$) across the reactor core for three energy groups for HFM and AHPOD and a pointwise difference between the two.

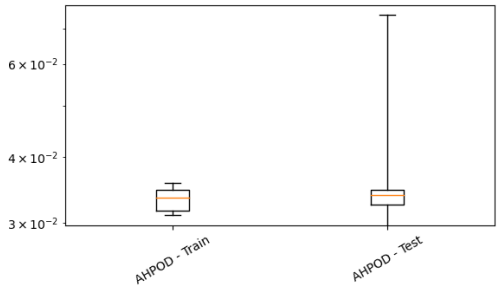


(b) k_{eff} vs power iteration for a reactor core solution with unseen case $r = 0.565$. The converged k_{eff} for each method is as follows: HFM - **1.22485**, AHPOD - **1.22267**.

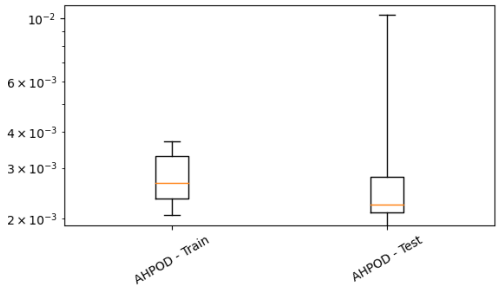
Figure 8.7: Fuel assembly flux and k_{eff} convergence for a reactor core solution using AHPOD with unseen case $r = 0.565$.

Figure 8.7 shows the flux profile and k_{eff} convergence for a unseen case with $r = 0.565$. Figure 8.7a shows similar errors as figures 8.5 and figure 8.7 shows similar convergence to figure 8.6, showing similar results between the seen and unseen cases, despite the difference in r value

(0.891 in the seen, 0.565 in the unseen).



(a) Boxplot of the error in the flux profile for the AHPOD method applied to the reactor core, for seen and unseen datasets.



(b) Boxplot of the error in k_{eff} for the AHPOD method applied to the reactor core, for seen and unseen datasets.

Figure 8.8: Boxplots showing the errors for the AHPOD method applied to the reactor core, for seen and unseen datasets. The median is given by the orange line, the interquartile ranges by the box and the minimum and maximum values by whiskers.

Figure 8.8 contains the boxplots for the errors in the flux profile and k_{eff} for the AHPOD method applied to the seen and unseen data. It can be observed that the seen and unseen datasets have similar medians and interquartile ranges, with the test cases having a larger maximum. Similar performance on both datasets indicate good generalisation.

		POD	HPOD
Flux Error	$e_{\max}(\boldsymbol{\phi}^{\text{ROM}})$	3.091×10^{-2}	2.026×10^{-1}
k_{eff} Error	k_{eff}	3.059×10^{-3}	6.772×10^{-2}

Table 8.2: Average maximum errors for the seen dataset using POD and HPOD ROM.

Table 8.2 contains the average maximum errors for the POD and HPOD methods. Again, it can be noticed that the HPOD method performs an order of magnitude worse than the POD method.

		Seen	Unseen
Flux Error	$e_{\max}(\boldsymbol{\phi}^{\text{AHPOD}})$	3.332×10^{-2}	3.400×10^{-2}
k_{eff} Error	k_{eff}	2.753×10^{-3}	2.866×10^{-3}

Table 8.3: Average maximum errors for the seen and unseen data sets using AHPOD ROM.

Table 8.3 contains the average maximum errors for the AHPOD method applied to the seen and unseen datasets. It can be observed that after adjustment, the AHPOD method has similar errors to the POD ones in table 8.2 for the seen dataset. AHPOD performs similarly on the seen data and the unseen data.

8.4 Summary

This chapter assesses how POD can be combined with interpolation matrices to form a Hierarchy of ROMs, with two methods being developed. The first method involves just interpolating between similar regions called Hierarchical-POD (HPOD). The second involves an adjustment phase, where the interpolation matrices are removed, called Adjusted Hierarchical-POD (AHPOD). These methods are tested on a fuel assembly problem and a reactor core problem.

For the fuel assembly test case, the same POD basis functions are used for all methods. 20 solutions of the global domain are generated, separated into nine sections, resulting in 180 total snapshots for constructing the basis functions. Twenty basis functions are retained, resulting in a matrix of $70,000 \times 20$. These are used nine times in the POD method, one for each sub-domain, forming a R matrix that is $630,000 \times 180$. The HPOD method only requires using the basis functions four times, interpolating the remaining nine domains, forming a RM_I matrix of $630,000 \times 80$, less than half the size of the R matrix alone. Both methods show a similar rate of convergence, having converged in under 10 iterations. The AHPOD method shows an adjustment, starting with the RM_I matrix and changing to just the R after a tolerance has been achieved. Unsurprisingly this has the same order of magnitude of errors as the POD method. The correction appears to converge in under five power iterations. Without this adjustment, the HPOD method has Flux and k_{eff} errors that are almost two orders of magnitude worse than the POD method.

The second test case involved a reactor core. This reactor core consisted of nine fuel assemblies, the geometry as the previous test case, surrounded by seven reflector sub-domains of the same size as the fuel assemblies. The fuel assemblies and reflector regions are decomposed as in the previous test case, with each domain split into nine further sub-domains. Five basis functions are retained for both the fuel assembly and reflector regions, resulting in a matrix of 70000×5 for both sets. These are used in all 144 sub-domains for the POD method, resulting in an R matrix

of size $10,080,000 \times 720$. The HPOD method utilised the same interpolation matrices used in the previous test case to reduce the sub-domains resolved in each fuel assembly and reflector. It also utilised an additional interpolation matrix, reducing the number of fuel assemblies resolved from nine to four and reducing the number of reflector regions resolved from seven to three. This results in a $RM_{I2}M_{I1}$ matrix of size $10,080,000 \times 140$, less than one-fifth the size of the POD method. The HPOD method converges slightly slower than the POD method. Again a correction is performed using the AHPOD method, which converges slightly quicker from adjustment. The errors for the HPOD method without this adjustment are about an order of magnitude worse. The method is also tested with unseen data, where similar errors are shown for both unseen and seen data.

In both test cases, the HPOD method produces a solution that is worse than just the POD method. The flux profiles produced by HPOD show a similar distribution to the HFM, potentially being used as a pre-conditioner. This was demonstrated using the POD method, which appears to converge slightly faster from the adjustment point than the POD method. The computational cost reduction for this may be more effective when the solver used after adjustment is vastly more computationally demanding than the HPOD method, such as using POD with a larger number of retained basis functions.

The methods utilised here assume that the POD coefficients in neighbouring regions could be linearly interpolated. This may not be the case, in which an alternative interpolating scheme may be more appropriate and result in more accurate solutions. Additionally, the interpolating scheme may be more effective with other forms of dimensionality reduction. Autoencoders with a distribution forced on the latent space, such as Variational or Adversarial autoencoders, may allow for the interpolation methods presented here to be used more effectively than POD, due to the non-linear dimensionality reduction.

Finally, the methods here could also be utilised with the AE-based methods established in section 6. In this case, interpolation would be done to the latent variables in the centre of the autoencoder. Using autoencoders requires more consideration, as the latent space that a typical autoencoder produces may not be appropriate for interpolation. In this case, a variational autoencoder or adversarial autoencoder would be more appropriate, providing the latent space with a more suitable distribution for interpolation.

Chapter 9

Conclusion

9.1 Summary of Thesis Achievements

Chapter 3 presents a new method for solving the diffusion approximation to the Boltzmann transport equations. This method utilises the tools within machine learning libraries to replicate the process of solving partial differential equations using a numerical scheme, such as the Jacobi method. The approach here defines the weights of convolutional layers to reproduce the operations of iterative solvers, forming a neural network that acts as a Jacobi iteration. This concept is then extended to develop a sawtooth multigrid solver, forming a neural network that acts as a multigrid iterative solution method, incorporating the Jacobi neural network internally. This multigrid network is used with another network that acts as a multi-group solver. A fuel assembly test case demonstrates that the approach produces the same (to within solver tolerance) solution as that obtained using a Gauss-Seidel solver written in Fortran, producing the same k_{eff} convergence. This test case is also used to demonstrate how higher-order filters may be substituted into the convolutional layers for a more accurate stencil for the diffusion operator. The approach is also extended to a more computationally demanding problem in the form of a reactor core. It is demonstrated that this method used in conjunction with a GPU is one-third the computational time as utilising a serial Fortran code. This method provides an easily accessible way for numerical schemes to capitalise on hardware developments without requiring significant modifications. Existing machine learning libraries are optimised for GPU and new AI computers, allowing neural networks to utilise this hardware. By forming the entire

solver as a neural network, it becomes optimised for this hardware and remains optimised with new developments as long as the libraries are maintained. This is the first implementation of a physics-based solver as a single neural network, utilising a multigrid method, providing the groundwork for future physics-based neural network solvers.

Chapter 4 extends the work of Chapter 3 to solve the Boltzmann transport equation. This work extends the neural network multigrid model to solve the discrete equations in the angle of particle travel as well as Cartesian space, enabling the level of parallelism necessary to run efficiently on GPU or AI computers. Additionally, a new Convolutional Finite Element method (ConvFEM) is presented, replacing the convolutional layers filters with higher-order stencils, enabling a more accurate solution. Finally, a non-linear Petrov-Galerkin method is presented. This method adds diffusion to the residual, increasing the stability of the higher-order filters presented. These methods are tested on a simple single-duct problem. The filters are demonstrated, showing that solutions converge and the Petrov-Galerkin diffusion helps keep them stable. These methods are extended to a multi-energy group fuel assembly problem, demonstrating that the method can be applied to more complex problems. This work extends the neural network solver to a space-angle multigrid method, develops the new ConvFEM and introduces a non-linear Petrov-Galerkin method for stability. This framework provides a physics-based neural network for the full Boltzmann transport equation that can be utilised for high-fidelity solvers.

Chapter 6 presents a new method for an autoencoder-based reduced order model. This utilises autoencoders (AE) for dimensionality reduction, producing a novel projection-based reduced-order model. Proper Orthogonal Decomposition (POD) and singular value decomposition (SVD) are used for dimensionality reduction, producing a linear embedding of the high-dimensional space into the low-dimensional space. The embedding found by an autoencoder, dependent on the activation functions, is generally non-linear. This non-linearity has the potential to provide more accurate representations of data. These methods are demonstrated on a simple 1D test case, where at higher basis functions, POD outperforms the networks. However, a test to determine the number of latent variables required is done using a variational autoencoder (VAE). The VAE provides a unique look at the problem, utilising only two variables out of ten possible. Based on this, the AE-based ROM is shown to outperform POD when only two variables are used. These methods are extended to a 2D test case, where a novel hybrid SVD-AE

is presented. This hybrid forgoes the non-linearity of a pure AE to utilise the basis functions typically used by POD as the first form of dimensionality reduction. These coefficients are then used within an autoencoder to compress further. It is shown that both autoencoder methods outperform POD, with the hybrid SVD-AE outperforming the method using just an AE. When AEs are used with ROM, they are typically used with a non-intrusive method, and this is the first one that is projection-based. The method can also work with Variational, Convolutional and Adversarial Autoencoders and provides a way to utilise neural networks for dimensionality reduction, which are shown to be more effective than POD, while utilising the governing equations.

Chapter 7 presents an approach to combine domain decomposition with reduced-order models for reactor physics problems. This method breaks the global problem into sub-domains, assigning a ROM to each sub-domain. The solution is then solved by iterating across sub-domains, enabling sub-domains to be solved in parallel. The method is first tested using POD as dimensionality reduction on a 1D test case. Various methods of snapshot selection are assessed, including a global method utilising all snapshots and local methods utilising specific sub-domains of snapshots. The local methods included datasets based on location, material type, or both. The accuracy is largely independent of this, meaning that basis functions can be utilised multiple times within the global domain. Reactors typically have repeating material parameters in fuel rods and fuel assemblies, so utilising smaller ROMs multiple times is beneficial. The second test involves limiting the number of snapshots, resulting in a loss in accuracy for all methods. However, the global method suffered a more significant loss than the others. One of the greatest drawbacks of ROM is the requirement during the offline phase of producing snapshots, which are still computationally expensive to produce for reactor problems. Improving accuracy by utilising fewer or smaller snapshots is beneficial to reducing this drawback for computationally demanding problems. These methods are then extended to a 2D case, where ROMs are constructed based on material type. A comparison is given between POD and an AE-based ROM, showing that the AE-based ROM slightly outperforms POD with a reduction in the number of compressed variables (latent variables). Additionally, a method for utilising less computationally demanding snapshots is presented by constructing the ROMs for sub-domains from partial snapshots of the system. This method shows that a similar accuracy can be produced by both POD and AE-based ROMs using smaller snapshots generated in the offline phase, reducing the computational cost of this phase. The work here demonstrates

the viability of separating the global domain, the reactor, into smaller sub-domains, the fuel assemblies and fuel rods, allowing a few smaller ROMs to construct the global solution. These methods have the added benefit of potentially reducing the computational cost of the offline phase, a significant benefit when this phase is so demanding. The methods here provide a way to capitalise on multiple pieces of hardware, potentially providing a significant computational speed up.

Chapter 8 presents an interpolation method for forming a hierarchy of ROMs. Like Chapter 7, this method decomposes the global domain into several sub-domains, assigning a ROM to each one based on material parameters. Specific sub-domains, where the neighbouring sub-domains have the same material parameters, are instead interpolated, assuming a linear interpolation between the neighbours. A Hierarchical-POD method is developed, utilising an interpolation matrix to reduce the number of sub-domains being resolved. The first test case is a fuel assembly, decomposed into nine sub-domains, each utilising the same POD basis functions. The HPOD method uses interpolation to resolve four out of the nine sub-domains. The second test case is a reactor core consisting of a 4×4 grid of fuel assembly and reflector sub-domains. The HPOD method utilises the same interpolation matrices for each sub-domain as the first test case, with an additional interpolation between fuel assemblies and reflector regions, forming a two-level hierarchy. The second level means it resolves only seven out of the sixteen sub-domains. In both cases, the HPOD method revolves to a reasonable approximation of the HFM flux profile but performs almost two orders of magnitude worse than the POD method alone. The Adjusted Hierarchical POD method, where an adjustment is made to correct the solution using the standard POD method, is also demonstrated on both test cases. This shows slightly quicker convergence from adjustment than the POD method does from the start. A reactor typically has hundreds of fuel assemblies and thousands of fuel rods, resulting in the same material patterns appearing multiple times in the global system. These repeated material parameters offer an opportunity to reduce the computational cost significantly by interpolating between areas where appropriate. ROMs offer the opportunity to interpolate between the reduced variables, approximating the solution in specific sub-domains. This could be used as a pre-conditioner, reducing the computational cost of generating a solution. The current interpolation scheme does not provide a sufficiently accurate solution for the reduction in computational cost to be utilised without the adjustment, therefore if this method is to be utilised changes would need to be made to the interpolation scheme to provide more accurate solutions.

9.2 Future Work

Although each chapter details how the relevant methods might be used in future work, they are summarised here for a better overview. Although there is some cross-over, the overall future work of chapters 3 and 4 are summarised together in section 9.2.1, and chapters 6, 7 and 8 are summarised in section 9.2.2. The possibility of this future work is given in section 9.2.3.

9.2.1 Neural Network Solvers

The Neural Network solvers produced in the research are a neural network implementation of the physics of neutron transport, producing a physics-driven network. The final stage for neutron transport would be to adapt the network to incorporate the power method, creating a single network that solves eigenvalue problems. A network that solves the complete neutron transport problem is the first stage in producing a quick, accessible, and accurate simulation of a complete reactor. However, a full reactor simulation requires more physics models than just neutron transport. Touched on briefly in the literature review, coupling involves at least two models taking information from one another but significantly increases the computational power required. Thermal hydraulics is an essential part of the physics of an entire reactor, playing a significant role in the moderator and coolant of a reactor. Coupling the thermal hydraulics with neutronics means that the thermal hydraulic code can convey thermodynamics information to the neutron transport code, which can, in turn, convey flux information back. Future work could involve modelling the thermal hydraulics as a neural network, which can then be coupled with the neutron transport network and form a single network containing both physics (neutron transport and thermal hydraulics). Capturing all of the physics of a reactor as a single neural network may provide significant insight into the behaviour of a reactor, including information into the unknown physics.

Utilising the neural network architecture for physics allows an approach to modelling the unknown physics while considering the known physics. AI models will typically be data-driven, where models are trained to approximate the whole system's behaviour. Data-driven methods are helpful when the physics of a system is unknown, but often in physics-based models, there are large parts where the physics is known. For a reactor, one example is subgrid-scale modelling, which is the representation of small-scale physics that cannot be accurately resolved on

a computational mesh and is often key to modelling the turbulent flow and other small-scale physics in a reactor. The physics-driven neural networks offer an opportunity to incorporate additional trainable parameters, representing the sub-grid scale physics or other unknown governing equations. The networks can then be trained on available data, producing a hybrid physics-resolving and sub-grid-scale neural reactor model.

For real-time and safety analysis, a reactor simulation must be as accurate as possible, where uncertainties may be catastrophic. The neural network architecture has benefits, allowing typical machine learning techniques, such as backpropagation, to be used. The ability to use these techniques means that some of the recent advancements in data assimilation, uncertainty quantification and sensitivity analysis using machine learning can be utilised. This allows an insight into the inner workings of a reactor, a significant advantage over existing code.

Finally, it has been mentioned that the solutions generated using the neural networks were run on GPUs. A typical neutron transport code would be significantly harder to implement in a way that utilised a GPU in the same manner, meaning this method improves the accessibility of the running computationally demanding problems in parallel. However, modern machine-learning libraries are well maintained, adapting quickly to hardware developments. New exascale computers, utilising AI computers and >10K GPUs, offer an opportunity for some of the most computationally demanding physics problems to be run. These exascale computers can be used with machine learning libraries, meaning that the physics-driven neural networks can be run effectively on them. The whole reactor can be broken down into sub-domains, assigning a GPU and network to each sub-domain. These sub-domains could be resolved with some of the domain decomposition methods described for ROM. Optimising the neural network architecture for use with multiple GPUs will be an important part of using this method for high-fidelity solvers.

9.2.2 Reduced-Order Modelling

Autoencoders are a significant development in dimensionality reduction. They can provide a more accurate representation of the data at a lower number of latent variables than other methods might provide. As machine learning methods develop, the architectures of autoencoders may change, providing even more accurate representations. The autoencoder-based Reduced-

Order Model (ROM) produced here is a method that provides a way to utilise any autoencoder with the governing equations of a system, typically more accurate than the non-intrusive methods. This method may prove more effective in other areas of physics, such as fluid dynamics, where nonlinearities are more common than neutronics. For complete reactor-core modelling, the individual physics models can each utilise their own ROM, which can then be coupled to produce a full reactor simulation. As mentioned previously, the computational cost of this method can depend on the size of the neural network used. One way to prevent this is to remove the need to perform the forward passes of the neural network by pre-calculating the matrices used to map the latent variables to the full flux, reducing accuracy but improving the computational cost during the online phase.

ROM methods combined with domain decomposition methods would benefit nuclear modelling. The ROMs being constructed from sub-domains allow a level of flexibility that a global ROM does not allow. Geometry can be varied heavily, with ROMs being moved around to determine an optimum geometry. Additionally, optimum ROMs can be utilised within each sub-domain, where POD/AE-based methods can be used when they provide the most accurate solution for that sub-domain. The ability to vary geometry significantly, which is not possible with global methods, provides a way to optimise designs at significantly less computational cost during the offline phase, which typically requires a high computational cost to produce the original snapshots. This method is ideally suited to a reactor, where geometries repeat themselves. Additionally, being able to utilise multiple pieces of hardware is beneficial for extremely computationally demanding problems. Optimising the datasets for use in producing ROMs for sub-domains, such as grouping sections of a domain with similar behaviour, would be important for developing this method further.

Although not focused on in the research presented here, Non-intrusive Reduced-Order Models (NIROM) have seen significant advancement, especially in machine learning. These models can be helpful when the physics is complicated or unknown, as can be the case in reactors. The domain decomposition and hierarchical methods presented here could be utilised with NIROM to model a reactor's known and unknown physics in a more data-driven approach.

The Reduced-Order Models established here could be utilised within the neural network solvers described in the previous section. Physics-driven neural networks can utilise the dimensionality reduction methods established, easily incorporating autoencoders since they are already neural

networks. POD basis functions could also be used within a single fully connected layer, each basis function representing the weights of the connections between the inputs and a single output. These can be used to model specific parts of a reactor. One approach may be to utilise them for sub-grid scale modelling, where NIROM can be used to form a data-driven approximation.

9.2.3 The Transparent Neural Nuclear Reactor

As emphasised during this research, resolving the physics of an entire reactor is a computationally demanding task which has kept the nuclear sector from utilising some critical benefits of computational modelling. The research presented here attempts to assist with this, using computational resources more efficiently through the neural network solvers and reducing the computational cost of producing solutions for unseen parameters through reduced-order modelling and domain decomposition. A concept combining all of these, along with the future work described here, is proposed here, called the Transparent Neural Nuclear Reactor.

The Transparent Neural Nuclear Reactor is a model that addresses current computational problems by capturing the full physics of a reactor as a single neural network. All known physics models can be implemented as described in chapters 3 and 4, coupled together to form a single neural network. The unknown governing equations can then be represented through minimal additional trainable parameters, utilising available data for a hybrid physics-resolving and sub-grid-scale neural reactor model. Particular physics, such as those at a sub-grid scale, may benefit from utilising reduced-order models, significantly reducing the computational cost when an efficient low-dimensional representation can be found.

The result is a single neural network capable of capturing all the physics of a reactor, potentially providing a more accurate representation than a pure data-driven or pure physics-driven method alone. The neural network can be utilised for data assimilation or uncertainty quantification, revealing the inner workings of a reactor (the transparent reactor concept) and the ability to incorporate live data from real reactors for real-time analysis. In addition, utilising the machine learning libraries, the framework would be optimised for use with any hardware, from a researcher with a single GPU to an organisation with access to hundreds of GPUs or TPUs. This allows the accessibility of research for all scales in a way that has not been seen before.

Bibliography

1. Ragusa, J. *Overview of reactor core neutron transport codes* 2006. https://www.mcs.anl.gov/events/workshops/nprcsafc/Group20D/Ragusa_DOE_aug_11_v1.pdf.
2. Meyer, G. & Stokke, E. Description of Sizewell B Nuclear Power Plant (1997).
3. Palmtag, S. *et al.* Coupled neutronics and thermal-hydraulic solution of a full core PWR using VERA-CS. *Proc. Int. Topl. Mtg. Advances in Reactor Physics (PHYSOR)* (eds Chiba, G. & Yamamoto, A.) (2014).
4. Slaybaugh, R., Ramirez-Zweiger, M., Pandya, T., Hamilton, S. & Evans, T. Eigenvalue Solvers for Modeling Nuclear Reactors on Leadership Class Machines. *Nuclear Science and Engineering* **190,** 31–44 (2018).
5. Brunton, S., Noack, B. & Koumoutsakos, P. Machine learning for fluid mechanics. *Annual Review of Fluid Mechanics* **52,** 477–508 (2020).
6. Schilders, W., Van der Vorst, H. & Rommes, J. *Model order reduction: theory, research aspects and applications* (eds Schilders, W., van der Vorst, H. & Rommes, J.) (Springer, 2008).
7. Toselli, A. & Widlund, O. *Domain decomposition methods-algorithms and theory* (Springer Science & Business Media, 2006).
8. Phillips, T. R., Heaney, C. E., Chen, B., Buchan, A. G. & Pain, C. C. Solving the discretised neutron diffusion equations using neural networks. *International Journal for Numerical Methods in Engineering* (2023).
9. Phillips, T. R., Heaney, C. E., Chen, B., Buchan, A. G. & Pain, C. C. *Solving the discretised Boltzmann transport equations using neural networks: Applications in neutron transport* Unpublished Link. Unpublished.

10. Phillips, T. R., Heaney, C. E., Smith, P. N. & Pain, C. C. An autoencoder-based reduced-order model for eigenvalue problems with application to neutron diffusion. *International Journal for Numerical Methods in Engineering* **122,** 3780–3811 (2021).

11. Phillips, T. R., Heaney, C. E., Tollit, B. S., Smith, P. N. & Pain, C. C. Reduced-Order Modelling with Domain Decomposition Applied to Multi-Group Neutron Transport. *Energies* **14,** 1369 (2021).

12. Phillips, T. R. *et al.* Multi-Output Regression with Generative Adversarial Networks (MOR-GANs). *Applied Sciences* **12,** 9209 (2022).

13. Phillips, T. R. *et al. Data-driven modelling and analysis of pollution at a point in an underground station* Unpublished Link. Unpublished.

14. Lewis, E. E. & Miller, W. F. Computational methods of neutron transport (1984).

15. Patankar, S. *Numerical heat transfer and fluid flow* (CRC press, 1980).

16. Bernard, L. & Bonnaud, E. Finite volume method for fission gas release modeling. *Journal of nuclear materials* **244,** 75–84 (1997).

17. Bathe, K.-J. *Finite element procedures* (2006).

18. Zienkiewicz, O. & Taylor, R. *The finite element method: solid mechanics* (Butterworth-Heinemann, 2000).

19. Lathouwers, D. Goal-oriented spatial adaptivity for the S_N equations on unstructured triangular meshes. *Annals of Nuclear Energy* **38,** 1373–1381 (2011).

20. Goffin, M. *et al.* Minimising the error in eigenvalue calculations involving the Boltzmann transport equation using goal-based adaptivity on unstructured meshes. *Journal of Computational Physics* **242,** 726–752 (2013).

21. Baker, G. Finite element methods for elliptic equations using nonconforming elements. *Mathematics of Computation* **31,** 45–59 (1977).

22. Lathrop, K. & Carlson, B. *Discrete ordinates angular quadrature of the neutron transport equation* tech. rep. (Los Alamos Scientific Lab., N. Mex., 1964).

23. Chai, J., Lee, H. & Patankar, S. Ray effect and false scattering in the discrete ordinates method. *Numerical Heat Transfer, Part B Fundamentals* **24,** 373–389 (1993).

24. Pomraning, G. Variational boundary conditions for the spherical harmonics approximation to the neutron transport equation. *Annals of Physics* **27,** 193–215 (1964).

25. Daubechies, I. Orthonormal bases of compactly supported wavelets. *Communications on Pure and Applied Mathematics* **41,** 909–996 (1988).

26. Askew, J., Fayers, F. & Kemshell, P. *GENERAL DESCRIPTION OF THE LATTICE CODE WIMS.* tech. rep. (Atomic Energy Establishment, Winfrith, Eng., 1966).

27. Pain, C., de Oliveira, C., Goddard, A. & Umpleby, A. Non-linear space-dependent kinetics for the criticality assessment of fissile solutions. *Progress in Nuclear Energy* **39,** 53–114 (2001).

28. Buchan, A. *et al.* Simulated transient dynamics and heat transfer characteristics of the water boiler nuclear reactor SUPO with cooling coil heat extraction. *Annals of Nuclear Energy* **48,** 68–83 (2012).

29. Jewer, S., Buchan, A., Pain, C. & Cacuci, D. An immersed body method for coupled neutron transport and thermal hydraulic simulations of PWR assemblies. *Annals of Nuclear Energy* **68,** 124–135 (2014).

30. Smith, P., Lillington, J., Pain, C., Buchan, A. & Dargaville, S. Directions in Radiation Transport. *International Journal of Multiphysics* **10** (2016).

31. Golub, G. & van Loan, C. *Matrix Computations* (John Hopkins University Press, 1996).

32. Brandt, A. Multi-level adaptive solutions to boundary-value problems. *Mathematics of computation* **31,** 333–390 (1977).

33. Wesseling, P. *Introduction to multigrid methods* tech. rep. (1995).

34. Brandt, A. Advanced Multigrid Solvers for Fluid Dynamics (1999).

35. Stüben, K. An introduction to algebraic multigrid. *Multigrid,* 413–532 (2001).

36. Larsen, E. W. in *Modern mathematical methods in transport theory* 34–47 (Springer, 1991).

37. Buchan, A., Pain, C., Umpleby, A. & Smedley-Stevenson, R. A sub-grid scale finite element agglomeration multigrid method with application to the Boltzmann transport equation. *International journal for numerical methods in engineering* **92,** 318–342 (2012).

38. Dargaville, S. *et al.* Solving the Boltzmann transport equation with multigrid and adaptive space/angle discretisations. *Annals of Nuclear Energy* **86,** 99–107 (2015).

39. Zahr, M. & Farhat, C. Progressive construction of a parametric reduced-order model for PDE-constrained optimization. *International Journal for Numerical Methods in Engineering* **102,** 1111–1135 (2015).

40. Ballarin, F. *et al.* Fast simulations of patient-specific haemodynamics of coronary artery bypass grafts based on a POD-Galerkin method and a vascular shape parametrization. *Journal of Computational Physics* **315,** 609–628 (2016).

41. Kerfriden, P., Passieux, J. & Bordas, S. Local/global model order reduction strategy for the simulation of quasi-brittle fracture. *International Journal for Numerical Methods in Engineering* **89,** 154–179 (2012).

42. Agathos, K., Bordas, S. & Chatzi, E. Parametrized reduced order modeling for cracked solids. *International Journal for Numerical Methods in Engineering* **accepted,** in press. https://onlinelibrary.wiley.com/doi/abs/10.1002/nme.6447 (2020).

43. Jansen, J. & Durlofsky, L. Use of reduced-order models in well control optimization. *Optimization and Engineering* **18,** 105–132 (2017).

44. Xiao, D. *et al.* Non-intrusive model reduction for a 3D unstructured mesh control volume finite element reservoir model and its application to fluvial channels. *International Journal of Oil, Gas and Coal Technology* **19,** 316–338 (2018).

45. Hoang, H., Fu, Y. & Song, J. An hp-proper orthogonal decomposition–moving least squares approach for molecular dynamics simulation. *Computer Methods in Applied Mechanics and Engineering* **298,** 548–575 (2016).

46. Haasdonk, B. & Ohlberger, M. Efficient reduced models and a posteriori error estimation for parametrized dynamical systems by offline/online decomposition. *Mathematical and Computer Modelling of Dynamical Systems* **17,** 145–161 (2011).

47. Peherstorfer, B. & Willcox, K. Dynamic data-driven reduced-order models. *Computer Methods in Applied Mechanics and Engineering* **291,** 21–41 (2015).

48. Everson, R. & Sirovich, L. Karhunen–Loeve procedure for gappy data. *JOSA A* **12,** 1657–1664 (1995).

49. Peherstorfer, B. & Willcox, K. Dynamic data-driven model reduction: adapting reduced models from incomplete data. *Advanced Modeling and Simulation in Engineering Sciences* **3,** 11 (2016).

50. Bui-Thanh, T., Damodaran, M. & Willcox, K. *Proper orthogonal decomposition extensions for parametric applications in compressible aerodynamics* in *21st AIAA Applied Aerodynamics Conference* (2003).

51. Lumley, J. The structure of inhomogeneous turbulent flows. *Atmospheric turbulence and radio wave propagation* (1967).

52. Moore, B. Principal component analysis in linear systems: Controllability, observability, and model reduction. *IEEE transactions on automatic control* **26,** 17–32 (1981).

53. Glover, K. All optimal Hankel-norm approximations of linear multivariable systems and their L_∞-error bounds. *International Journal of Control* **39,** 1115–1193 (1984).

54. Sirovich, L. Turbulence and the dynamics of coherent structures Part I: Coherent structures. *Quarterly of Applied Mathematics* **45,** 561–571 (1987).

55. Pinnau, R. in *Model Order Reduction: Theory, Research Aspects and Applications* 95–109 (Springer, 2008).

56. Kunisch, K. & Volkwein, S. Galerkin proper orthogonal decomposition methods for parabolic problems. *Numerische mathematik* **90,** 117–148 (2001).

57. Holmes, P., Lumley, J., Berkooz, G. & Rowley, C. *Turbulence, coherent structures, dynamical systems and symmetry* (Cambridge University Press, 2012).

58. Hotelling, H. Simplified calculation of principal components. *Psychometrika* **1,** 27–35 (1936).

59. Karhunen, K. Zur spektraltheorie stochastischer prozesse. *Ann. Acad. Sci. Fennicae, AI* (1946).

60. Loève, M. Functions aleatoire de second ordre. *Revue science* **84,** 195–206 (1946).

61. Chaturantabut, S. & Sorensen, D. C. Nonlinear model reduction via discrete empirical interpolation. *SIAM Journal on Scientific Computing* **32,** 2737–2764 (2010).

62. Chaturantabut, S. & Sorensen, D. C. Application of POD and DEIM on dimension reduction of non-linear miscible viscous fingering in porous media. *Mathematical and Computer Modelling of Dynamical Systems* **17,** 337–353 (2011).

63. Du, J. *et al.* POD reduced-order unstructured mesh modeling applied to 2D and 3D fluid flow. *Computers & Mathematics with Applications* **65,** 362–379 (2013).

64. Xiao, D. *et al.* Non-linear model reduction for the Navier–Stokes equations using residual DEIM method. *Journal of Computational Physics* **263,** 1–18 (2014).

65. Benner, P., Gugercin, S. & Willcox, K. A survey of projection-based model reduction methods for parametric dynamical systems. *SIAM review* **57,** 483–531 (2015).

66. Breitkopf, P., Lepot, I., Sainvitu, C. & Villon, P. Multi-fidelity POD surrogate-assisted optimization: Concept and aero-design study. *Struct Multidisc Optim* **56,** 1387–1412 (2017).

67. Kaiser, E. *et al.* Cluster-based reduced-order modelling of a mixing layer. *Journal of Fluid Mechanics* **754,** 365–414 (2014).

68. Guo, M. & Hesthaven, J. Data-driven reduced order modeling for time-dependent problems. *Computer Methods in Applied Mechanics and Engineering* **345,** 75–99 (2019).

69. Swischuk, R., Mainini, L., Peherstorfer, B. & Willcox, K. Projection-based model reduction: Formulations for physics-based machine learning. *Computers & Fluids* **179,** 704–717 (2019).

70. Polifke, W. Black-box system identification for reduced order model construction. *Annals of Nuclear Energy* **67,** 109–128 (2014).

71. Wang, Z. *et al.* Model identification of reduced order fluid dynamics systems using deep learning. *International Journal for Numerical Methods in Fluids* **86,** 255–268 (2018).

72. Bui, D., Hamdaoui, M. & De Vuyst, F. POD–ISAT: An efficient POD-based surrogate approach with adaptive tabulation and fidelity regions for parametrized steady-state PDE discrete solutions. *International Journal for Numerical Methods in Engineering* **94,** 648–671 (2013).

73. Shinde, V., Longatte, E., Baj, F., Hoarau, Y. & Braza, M. A Galerkin-free model reduction approach for the Navier–Stokes equations. *Journal of Computational Physics* **309,** 148–163 (2016).

74. Hamdaoui, M., Le Quilliec, G., Breitkopf, P. & Villon, P. POD surrogates for real-time multi-parametric sheet metal forming problems. *International journal of material forming* **7,** 337–358 (2014).

75. Xiao, D., Du, J., Fang, F., Pain, C. & Li, J. Parameterised non-intrusive reduced order methods for ensemble Kalman filter data assimilation. *Computers & Fluids* **177,** 69–77 (2018).

76. Guénot, M., Lepot, I., Sainvitu, C., Goblet, J. & Filomeno Coelho, R. Adaptive sampling strategies for non-intrusive POD-based surrogates. *Engineering computations* **30,** 521–547 (2013).

77. Sartori, A., Cammi, A., Luzzi, L. & Rozza, G. A multi-physics reduced order model for the analysis of Lead Fast Reactor single channel. *Annals of Nuclear Energy* **87,** 198–208 (2016).

78. Wahi, P. & Kumawat, V. Nonlinear stability analysis of a reduced order model of nuclear reactors: A parametric study relevant to the advanced heavy water reactor. *Nuclear Engineering and Design* **241,** 134–143 (2011).

79. Buchan, A. *et al.* A POD reduced order model for resolving angular direction in neutron/photon transport problems. *Journal of Computational Physics* **296,** 138–157 (2015).

80. Sartori, A., Cammi, A., Luzzi, L. & Rozza, G. A reduced basis approach for modeling the movement of nuclear reactor control rods. *Journal of Nuclear Engineering and Radiation Science* **2,** 021019 (2016).

81. Sartori, A., Cammi, A., Luzzi, L. & Rozza, G. Reduced Basis Approaches in Time-Dependent Non-Coercive Settings for Modelling the Movement of Nuclear Reactor Control Rods. *Communications in Computational Physics* **20,** 23–59 (2016).

82. Chunyu, Z. & Gong, C. Fast solution of neutron diffusion problem by reduced basis finite element method. *Annals of Nuclear Energy* **111,** 702–708 (2018).

83. Lorenzi, S. An Adjoint Proper Orthogonal Decomposition method for a neutronics reduced order model. *Annals of Nuclear Energy* **114,** 245–258 (2018).

84. Reed, R. & Roberts, J. Effectiveness of the discrete generalized multigroup method based on truncated, POD-driven basis sets. *Annals of Nuclear Energy* **126,** 253–261 (2019).

85. Heaney, C. E., Buchan, A. G., Pain, C. C. & Jewer, S. Reactor Simulators and Reduced Order Modelling. *Nuclear Future (May/June)* **May/June,** 49–54 (2018).

86. Heaney, C. E., Buchan, A. G., Pain, C. C. & Jewer, S. Reduced-order modelling applied to the multigroup neutron diffusion equation using a nonlinear interpolation method for control-rod movement. *Energies* **14,** 1350 (2021).

87. German, P. & Ragusa, J. C. Reduced-order modeling of parameterized multi-group diffusion k-eigenvalue problems. *Annals of Nuclear Energy* **134,** 144–157 (2019).

88. Huang, D., Abdel-Khalik, H., Rabiti, C. & Gleicher, F. Dimensionality reducibility for multi-physics reduced order modeling. *Annals of Nuclear Energy* **110,** 526–540 (2017).

89. Georgaka, S., Stabile, G., Rozza, G. & Bluck, M. Parametric POD-Galerkin Model Order Reduction for Unsteady-State Heat Transfer Problems. *arXiv preprint arXiv:arXiv:1808.05175.* arXiv:1808.05175, . https://arxiv.org/abs/1808.05175 (2018).

90. Abadi, M. *et al. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems* Software available from www.tensorflow.org. 2015.

91. Paszke, A. *et al.* in *Advances in Neural Information Processing Systems 32* 8024–8035 (Curran Associates, Inc., 2019). http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

92. Yang, C., Yang, X. & Xiao, X. Data-driven projection method in fluid simulation. *Computer Animation and Virtual Worlds* **27,** 415–424 (2016).

93. Zhu, Y., Zabaras, N., Koutsourelakis, P.-S. & Perdikaris, P. Physics-constrained deep learning for high-dimensional surrogate modeling and uncertainty quantification without labeled data. *Journal of Computational Physics* **394,** 56–81 (2019).

94. Hesthaven, J. S. & Ubbiali, S. Non-intrusive reduced order modeling of nonlinear problems using neural networks. *Journal of Computational Physics* **363,** 55–78 (2018).

95. Ribeiro, M. D., Rehman, A., Ahmed, S. & Dengel, A. DeepCFD: Efficient steady-state laminar flow approximation with deep convolutional neural networks. *arXiv preprint arXiv:2004.08826* (2020).

96. Lopez-Martin, M., Le Clainche, S. & Carro, B. Model-free short-term fluid dynamics estimator with a deep 3D-convolutional neural network. *Expert Systems with Applications* **177,** 114924 (2021).

97. Cui, W., Cao, B., Fan, Q., Fan, J. & Chen, Y. Source term inversion of nuclear accident based on deep feedforward neural network. *Annals of Nuclear Energy* **175,** 109257 (2022).

98. Xiao, K. *et al.* A neural network predictive control method for power control of small pressurized water reactors. *Annals of Nuclear Energy* **169,** 108946 (2022).

99. Schiassi, E., De Florio, M., Ganapol, B. D., Picca, P. & Furfaro, R. Physics-informed neural networks for the point kinetics equations for nuclear reactor dynamics. *Annals of Nuclear Energy* **167,** 108833. ISSN: 0306-4549 (2022).

100. Foad, B., Elzohery, R. & Novog, D. R. Demonstration of combined reduced order model and deep neural network for emulation of a time-dependent reactor transient. *Annals of Nuclear Energy* **171,** 109017 (2022).

101. Qin, S. *et al.* Application of deep neural network for generating resonance self-shielded cross-section. *Annals of Nuclear Energy* **149,** 107785 (2020).

102. Gong, H., Cheng, S., Chen, Z. & Li, Q. Data-Enabled Physics-Informed Machine Learning for Reduced-Order Modeling Digital Twin: Application to Nuclear Reactor Physics. *Nuclear Science and Engineering* **196,** 668–693 (2022).

103. Wang, J. *et al.* Surrogate modeling for neutron diffusion problems based on conservative physics-informed neural networks with boundary conditions enforcement. *Annals of Nuclear Energy* **176,** 109234 (2022).

104. Lu, T., Marin, T., Zhuo, Y., Chen, Y.-F. & Ma, C. *Accelerating MRI Reconstruction on TPUs* in *2020 IEEE High Performance Extreme Computing Conference (HPEC)* (2020), 1–9.

105. Lu, T., Marin, T., Zhuo, Y., Chen, Y.-F. & Ma, C. *Nonuniform Fast Fourier Transform on TPUs* in *2021 IEEE 18th International Symposium on Biomedical Imaging (ISBI)* (2021), 783–787.

106. Belletti, F. *et al. Tensor processing units for financial Monte Carlo* in *Proceedings of the 2020 SIAM Conference on Parallel Processing for Scientific Computing* (2020), 12–23.

107. Morningstar, A. *et al.* Simulation of Quantum Many-Body Dynamics with Tensor Processing Units: Floquet Prethermalization. *PRX Quantum* **3,** 020331 (2 2022).

108. Pederson, R. *et al.* Tensor Processing Units as Quantum Chemistry Supercomputers. *arXiv,* 2202.01255 (2022).

109. Zhao, X.-Z., Xu, T.-Y., Ye, Z.-T. & Liu, W.-J. A TensorFlow-based new high-performance computational framework for CFD. *Journal of Hydrodynamics* **32,** 735–746 (2020).

110. Wang, Q., Ihme, M., Chen, Y.-F. & Anderson, J. A TensorFlow simulation framework for scientific computing of fluid flows on tensor processing units. *Computer Physics Communications* **274,** 108292 (2022).

111. Ma, H., Zhang, Y., Thuerey, N., Hu, X. & Haidn, O. J. Physics-driven Learning of the Steady Navier-Stokes Equations using Deep Convolutional Neural Networks. *arXiv preprint arXiv:2106.09301* (2021).

112. Hinton, G. & Salakhutdinov, R. Reducing the dimensionality of data with neural networks. *science* **313,** 504–507 (2006).

113. Wang, J., He, H. & Prokhorov, D. A folded neural network autoencoder for dimensionality reduction. *Procedia Computer Science* **13,** 120–127 (2012).

114. Mao, X., Shen, C. & Yang, Y.-B. in *Advances in Neural Information Processing Systems 29* (eds Lee, D., Sugiyama, M., UV, L., Guyon, I. & Garnett, R.) 2802–2810 (Curran Associates, Inc, 2016).

115. Pascal, V., Larochelle, H., Lajoie, I., Bengio, Y. & Manzagol, P.-A. Stacked Denoising Autoencoders: Learning Useful Representationsina Deep Network with a Local Denoising Criterion. *Journal of Machine Learning Research* **11,** 3371–3408 (2010).

116. Gonzalez, F. & Balajewicz, M. *Deep convolutional recurrent autoencoders for learning low-dimensional feature dynamics of fluid systems* arXiv:1808.01346 [math.DS], . 2018. https://arxiv.org/abs/1808.01346.

117. Oja, E. A Simplified Neuron Model as a Principal Component Analyzer. *J. Math. Biology* **15,** 267–273 (1982).

118. Bourlard, H. & Kamp, Y. Auto-Association by Multilayer Perceptrons and Singular Value Decomposition. *Biol. Cybern.* **59,** 291–294 (1988).

119. Baldi, P. & Hornik, K. Neural Networks and Principal Component Analysis: Learning from Examples Without Local Minima. *Neural Networks* **2,** 53–58 (1989).

120. Takane, Y. *Nonlinear PCA by neural network models* in *Proceedings of the 63rd Annual Meeting of the Japan Statistical Society* (1998), 258–260.

121. Milano, M. & Koumoutsakos, P. Neural Network Modeling for Near Wall Turbulent Flow. *Journal of Computational Physics* **182,** 1–26 (2002).

122. Wiewel, S., Becher, M. & Thuerey, N. Latent Space Physics: Towards Learning the Temporal Evolution of Fluid Flow. *Computer Graphics Forum* **38,** 71–82 (2019).

123. Hartman, D. & Mestha, L. *A deep learning framework for model reduction of dynamical systems* in *2017 IEEE Conference on Control Technology and Applications (CCTA)* (2017), 1917–1922.

124. Kashima, K. *Nonlinear model reduction by deep autoencoder of noise response data* in *2016 IEEE 55th Conference on Decision and Control (CDC)* (2016), 5750–5755.

125. Lee, K. & Carlberg, K. Model reduction of dynamical systems on nonlinear manifolds using deep convolutional autoencoders. *Journal of Computational Physics* **404,** 108973 (2020).

126. Shady E. Ahmed, S., Rahman, S., Omer, S., Rasheed, A. & Navon, I. Memory embedded non-intrusive reduced order modeling of non-ergodic flows. *Phys. Fluids* **31,** 126602 (2019).

127. Schwarz, H. A. *Ueber einen Grenzübergang durch alternirendes Verfahren* (Zürcher u. Furrer, 1870).

128. Jamelot, E. & Ciarlet Jr, P. Fast non-overlapping Schwarz domain decomposition methods for solving the neutron diffusion equation. *Journal of Computational Physics* **241,** 445–463 (2013).

129. Baiges, J., Codina, R. & Idelsohn, S. A domain decomposition strategy for reduced order models. Application to the incompressible Navier–Stokes equations. *Computer Methods in Applied Mechanics and Engineering* **267,** 23–42 (2013).

130. Xiao, D. *et al.* A domain decomposition non-intrusive reduced order model for turbulent flows. *Computers & Fluids* **182,** 15–27 (2019).

131. Xiao, D., Fang, F., Heaney, C. E., Navon, I. & Pain, C. A domain decomposition method for the non-intrusive reduced order modelling of fluid flow. *Computer Methods in Applied Mechanics and Engineering* **354,** 307–330 (2019).

132. Arcucci, R. *et al.* A domain decomposition reduced order model with data assimilation (dd-roda). *Parallel Computing: Technology Trends* **36,** 189 (2020).

133. Heaney, C. E. *et al.* An AI-based Domain-Decomposition Non-Intrusive Reduced-Order Model for Extended Domains applied to Multiphase Flow in Pipes. *arXiv preprint arXiv:2202.06170* **34,** 055111 (2022).

134. Cherezov, A., Sanchez, R. & Joo, H. G. A reduced-basis element method for pin-by-pin reactor core calculations in diffusion and SP3 approximations. *Annals of Nuclear Energy* **116,** 195–209 (2018).

135. Lewis, A. G. M. *et al.* Large-scale distributed linear algebra with tensor processing units. *Proceedings of the National Academy of Sciences of the United States of America* **119,** e2122762119 (2022).

136. *CS-2: A Revolution in AI Infrastructure* https://www.cerebras.net/product-system/. Accessed: 2022-10-12.

137. Raissi, M., Perdikaris, P. & Karniadakis, G. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics* **378,** 686–707 (2019).

138. Cai, S. *et al.* Flow over an espresso cup: inferring 3-D velocity and pressure fields from tomographic background oriented Schlieren via physics-informed neural networks. *Journal of Fluid Mechanics* **915,** A102 (2021).

139. Raissi, M., Yazdani, A. & Karniadakis, G. E. Hidden fluid mechanics: Learning velocity and pressure fields from flow visualizations. *Science* **367,** 1026–1030 (2020).

140. He, Q., Barajas-Solano, D., Tartakovsky, G. & Tartakovsky, A. M. Physics-informed neural networks for multiphysics data assimilation with application to subsurface transport. *Advances in Water Resources* **141,** 103610 (2020).

141. Von Saldern, J. G. R., Reumschüssel, J. M., Kaiser, T. L., Sieber, M. & Oberleithner, K. Mean flow data assimilation based on physics-informed neural networks. *Physics of Fluids* **34,** 115129 (2022).

142. Ronneberger, O., Fischer, P. & Brox, T. *U-Net: Convolutional Networks for Biomedical Image Segmentation* (available on arXiv:1505.04597 [cs.CV]). 2015. https://arxiv.org/abs/1505.04597.

143. Cervi, E., Lu, X., Cammi, A., Di Maio, F. & Zio, E. Sensitivity-Analysis-Driven Surrogate Model for Molten Salt Reactors Control. *Journal of Nuclear Engineering* **3,** 277–294 (2022).

144. Kabir, H. D., Khosravi, A., Hosen, M. A. & Nahavandi, S. Neural network-based uncertainty quantification: A survey of methodologies and applications. *IEEE access* **6,** 36218–36234 (2018).

145. Amendola, M. *et al.* Data assimilation in the latent space of a neural network. *arXiv preprint arXiv:2012.12056* (2020).

146. Reams, R. Hadamard inverses, square roots and products of almost semi-definite matrices. *Linear Algebra and its Applications* **288,** 35–43 (1999).

147. Chollet, F. e. a. *Keras* https://keras.io. 2015.

148. Phillips, T. R. F., Heaney, C. E., Smith, P. N. & Pain, C. C. An autoencoder-based reduced-order model for eigenvalue problems with application to neutron diffusion. *International Journal for Numerical Methods in Engineering* **122,** 3780–3811 (2021).

149. Cho, Z. *Kaist Benchmark Problem 1A : MOX Fuel-Loaded Small PWR Core* http://http://nurapt.kaist.ac.kr/benchmark/kaist_ben1a.pdf. Accessed: 2021-3-2. June 2000.

150. Margenberg, N., Hartmann, D., Lessig, C. & Richter, T. A neural network multigrid solver for the Navier-Stokes equations. *Journal of Computational Physics* **460,** 110983 (2022).

151. He, J. & Xu, J. MgNet: A unified framework of multigrid and convolutional neural network. *Science China Mathematics* **62,** 1331–1354 (2019).

152. Ibtehaz, N. & Rahman, M. S. MultiResUNet : Rethinking the U-Net architecture for multimodal biomedical image segmentation. *Neural Networks* **121,** 74–87 (2020).

153. Alom, M. Z., Hasan, M., Yakopcic, C., Taha, T. M. & Asari, V. K. *Recurrent Residual Convolutional Neural Network based on U-Net (R2U-Net) for Medical Image Segmentation* 2018. https://arxiv.org/abs/1802.06955.

154. Tsuruga, J. & Iwasaki, K. Sawtooth cycle revisited. *Computer Animation and Virtual Worlds* **29.** e1836 cav.1836, e1836 (2018).

155. Briggs, W. L. *A multigrid tutorial* 2nd ed. (Society for Industrial and Applied Mathematics, Philadelphia, PA, 2000).

156. Dargaville, S., Buchan, A., Smedley-Stevenson, R., Smith, P. & Pain, C. Scalable angular adaptivity for Boltzmann transport. *Journal of Computational Physics* **406,** 109124 (2020).

157. Dargaville, S., Buchan, A., Smedley-Stevenson, R., Smith, P. & Pain, C. A comparison of element agglomeration algorithms for unstructured geometric multigrid. *Journal of Computational and Applied Mathematics* **390,** 113379 (2021).

158. Reed, W. H. & Hill, T. R. Triangular mesh methods for the neutron transport equation. *Tech. Report LA-UR-73–479. Los Alamos Scientific Laboratory.* (1973).

159. Cockburn, B. in *Encyclopedia of Computational Mechanics Second Edition* 1–63 (John Wiley & Sons, Ltd, 2017).

160. Gifford, K. A., Horton, J. L., Wareing, T. A., Failla, G. & Mourtada, F. Comparison of a finite-element multigroup discrete-ordinates code with Monte Carlo for radiotherapy calculations. *Phys Med Biol.* **51,** 2253–65 (2006).

161. Mille, M., Lee, C. & Failla, G. SU-F-T-111: Investigation of the Attila Deterministic Solver as a Supplement to Monte Carlo for Calculating Out-Of-Field Radiotherapy Dose. *Medical Physics* **43,** 3487–3487 (2016).

162. Royston, K. E. *et al.* Application of the Denovo Discrete Ordinates Radiation Transport Code to Large-Scale Fusion Neutronics. *Fusion Science and Technology* **74,** 303–314 (2018).

163. Hirsch, C. *Numerical Computation of Internal and External Flows: The Fundamentals of Computational Fluid Dynamics* Second Edition (Butterworth-Heinemann, Oxford, 2007).

164. Leonard, B. The ULTIMATE conservative difference scheme applied to unsteady one-dimensional advection. *Computer Methods in Applied Mechanics and Engineering* **88,** 17–74. ISSN: 0045-7825. https://www.sciencedirect.com/science/article/pii/0045782591902320 (1991).

165. Hughes, T. J., Mallet, M. & Akira, M. A new finite element formulation for computational fluid dynamics: II. Beyond SUPG. *Computer Methods in Applied Mechanics and Engineering* **54,** 341–355 (1986).

166. Fang, F. *et al.* Non-linear Petrov–Galerkin methods for reduced order hyperbolic equations and discontinuous finite element methods. *Journal of Computational Physics* **234,** 540–559. ISSN: 0021-9991. https://www.sciencedirect.com/science/article/pii/S0021999112006006 (2013).

167. Donea, J. & Huerta, A. *Finite Element Methods for Flow Problems* (John Wiley & Sons, Ltd, 2003).

168. Pain, C. C. *et al.* Space–time streamline upwind Petrov–Galerkin methods for the Boltzmann transport equation. *Computer Methods in Applied Mechanics and Engineering* **195,** 4334–4357 (2006).

169. Merton, S. R., Smedley-Stevenson, R. P., Buchan, A. G. & Eaton, M. D. *A Non-linear Optimal Discontinuous Petrov-Galerkin Method for Stabilising the solution of the Transport Equation* in (2009).

170. Kabai, S. *Octahedron in a Sphere* https://demonstrations.wolfram.com/OctahedronInASphere/ Accessed: 2022-10-12. Oct. 2008.

171. Phillips, T. *Neural Network Transport Solver* https://github.com/trfphillips/Neural-Network-Transport-Solver. 2022.

172. Donea, J. & Huerta, A. *Finite element methods for flow problems* (John Wiley & Sons, 2003).

173. Codina, R. A discontinuity-capturing crosswind-dissipation for the finite element solution of the convection-diffusion equation. *Computer Methods in Applied Mechanics and Engineering* **110,** 325–342 (1993).

174. Phillips, T. R., Heaney, C. E., Chen, B., Buchan, A. G. & Pain, C. C. *Solving the Discretised Neutron Diffusion Equations using Neural Networks* Unpublished Link. Unpublished.

175. Lumley, J. The structure of inhomogeneous turbulent flows. *Atmos. Turbul. Wave Propag.,* 166–178 (1967).

176. Taira, K. *et al.* Modal Analysis of Fluid Flows: An Overview. *AIAA Journal* **55,** 4013–4041 (2017).

177. Eckart, C. & Young, G. The approximation of one matrix by another of lower rank. *Psychometrika* **1,** 211–218 (1936).

178. Baur, U. *et al.* in *Model Reduction and Approximation: Theory and Algorithms* 377–407 (SIAM, 2017).

179. Wirtz, D., Sorensen, D. & Haasdonk, B. A posteriori error estimation for DEIM reduced nonlinear dynamical systems. *SIAM Journal on Scientific Computing* **36,** A311–A338 (2 2014).

180. Kuhn, M. & Johnson, K. *Applied Predictive Modeling* 2nd (Springer, 2018).

181. Rocca, J. & Rocca, B. *Understanding Variational Autoencoders (VAEs)* https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73. (last accessed June 2020). Sept. 2019.

182. Altosaar, J. *Tutorial - What is a variational autoencoder?* https://jaan.io/what-is-variational-autoencoder-vae-tutorial, (last accessed June 2020).

183. Buchan, A., Dargaville, S. & Pain, C. A combined immersed body and adaptive mesh method for simulating neutron transport within complex structures. *Annals of Nuclear Energy* **134,** 88–100 (2019).

184. Morel, J. & McGhee, J. A Self-Adjoint Angular Flux Equation. *Nuclear Science and Engineering* **132,** 312–325 (1999).

185. Golub, G. H. & van der Vorst, H. A. Eigenvalue computation in the 20th century. *Journal of Computational and Applied Mathematics* **123,** 35–65 (2000).

186. Buchan, A., Pain, C., Fang, F. & Navon, I. A POD reduced-order model for eigenvalue problems with application to reactor physics. *International Journal for Numerical Methods in Engineering* **95,** 1011–1032 (2013).

187. Clevert, D.-A., Unterthiner, T. & Hochreiter, S. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289* (2015).

188. Dozat, T. *Incorporating Nesterov Momentum into Adam* in *International Conference on Learning Representations* (2016).

189. Lindley, B. *et al.* Current status of the reactor physics code WIMS and recent developments. *Annals of Nuclear Energy* **102,** 148–157 (2017).

190. Xiao, D., Fang, F., Pain, C. & Hu, G. Non-intrusive reduced-order modelling of the Navier–Stokes equations based on RBF interpolation. *International Journal for Numerical Methods in Fluids* **79,** 580–595 (2015).

Appendix A

Appendix

A.1 The Convolution Finite Element Method (ConvFEM) Filters

Here we assume a uniform grid with $\Delta x = \Delta y =$ constant and write the filters with this assumption. We also note that common to all filter orders is the 1×1 lumped mass filter $\boldsymbol{w_{ml}} = m_L = \Delta x \Delta y$. The filters below, as well as higher order filters, are listed in the GitHub repository [171].

A.1.1 Linear Filters

$$\boldsymbol{w_x} = \frac{1}{\Delta x} \begin{bmatrix} -0.167 & 0 & 0.167 \\ -0.667 & 0 & 0.667 \\ -0.167 & 0 & 0.167 \end{bmatrix} \tag{A.1}$$

$$\boldsymbol{w_y} = \frac{1}{\Delta y} \begin{bmatrix} 0.167 & 0.667 & 0.167 \\ 0 & 0 & 0 \\ -0.167 & -0.667 & -0.167 \end{bmatrix} \tag{A.2}$$

$$\boldsymbol{w_{Diffxx}} = \frac{1}{(\Delta x)^2}\begin{bmatrix} -0.167 & 0.333 & -0.167 \\ -0.667 & 1.333 & -0.667 \\ -0.167 & 0.333 & -0.167 \end{bmatrix} \tag{A.3}$$

$$\boldsymbol{w_{Diffyy}} = \frac{1}{(\Delta y)^2}\begin{bmatrix} -0.167 & -0.667 & -0.167 \\ 0.333 & 1.333 & 0.333 \\ -0.167 & -0.667 & -0.167 \end{bmatrix} \tag{A.4}$$

$$\boldsymbol{w_m} = \frac{1}{(\Delta y)^2}\begin{bmatrix} 2.78 \times 10^{-2} & 1.11 \times 10^{-1} & 2.78 \times 10^{-2} \\ 1.11 \times 10^{-1} & 4.44 \times 10^{-1} & 1.11 \times 10^{-1} \\ 2.78 \times 10^{-2} & 1.11 \times 10^{-1} & 2.78 \times 10^{-2} \end{bmatrix} \tag{A.5}$$

A.1.2 Quadratic Filters

$$\boldsymbol{w_x} = \frac{1}{\Delta x}\begin{bmatrix} -2.78 \times 10^{-3} & 2.22 \times 10^{-2} & 0.00 & -2.22 \times 10^{-2} & 2.78 \times 10^{-3} \\ 1.11 \times 10^{-2} & -8.89 \times 10^{-2} & 0.00 & 8.89 \times 10^{-2} & -1.11 \times 10^{-2} \\ 6.67 \times 10^{-2} & -5.33 \times 10^{-1} & 0.00 & 5.33 \times 10^{-1} & -6.67 \times 10^{-2} \\ 1.11 \times 10^{-2} & -8.89 \times 10^{-2} & 0.00 & 8.89 \times 10^{-2} & -1.11 \times 10^{-2} \\ -2.78 \times 10^{-3} & 2.22 \times 10^{-2} & 0.00 & -2.22 \times 10^{-2} & 2.78 \times 10^{-3} \end{bmatrix} \tag{A.6}$$

$$\boldsymbol{w_y} = \frac{1}{\Delta y}\begin{bmatrix} -2.78 \times 10^{-3} & 1.11 \times 10^{-2} & 6.67 \times 10^{-2} & 1.11 \times 10^{-2} & -2.78 \times 10^{-3} \\ 2.22 \times 10^{-2} & -8.89 \times 10^{-2} & -5.33 \times 10^{-1} & -8.89 \times 10^{-2} & 2.22 \times 10^{-2} \\ 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\ -2.22 \times 10^{-2} & 8.89 \times 10^{-2} & 5.33 \times 10^{-1} & 8.89 \times 10^{-2} & -2.22 \times 10^{-2} \\ 2.78 \times 10^{-3} & -1.11 \times 10^{-2} & -6.67 \times 10^{-2} & -1.11 \times 10^{-2} & 2.78 \times 10^{-3} \end{bmatrix} \tag{A.7}$$

$$\boldsymbol{w_{Diffxx}} = \frac{1}{(\Delta x)^2}\begin{bmatrix} -2.78\times10^{-3} & 4.44\times10^{-2} & -8.33\times10^{-2} & 4.44\times10^{-2} & -2.78\times10^{-3} \\ 1.11\times10^{-2} & -1.78\times10^{-1} & 3.33\times10^{-1} & -1.78\times10^{-1} & 1.11\times10^{-2} \\ 6.67\times10^{-2} & -1.07 & 2.00 & -1.07 & 6.67\times10^{-2} \\ 1.11\times10^{-2} & -1.78\times10^{-1} & 3.33\times10^{-1} & -1.78\times10^{-1} & 1.11\times10^{-2} \\ -2.78\times10^{-3} & 4.44\times10^{-2} & -8.33\times10^{-2} & 4.44\times10^{-2} & -2.78\times10^{-3} \end{bmatrix} \tag{A.8}$$

$$\boldsymbol{w_{Diffyy}} = \frac{1}{(\Delta y)^2}\begin{bmatrix} -2.78\times10^{-3} & 1.11\times10^{-2} & 6.67\times10^{-2} & 1.11\times10^{-2} & -2.78\times10^{-3} \\ 4.44\times10^{-2} & -1.78\times10^{-1} & -1.07 & -1.78\times10^{-1} & 4.44\times10^{-2} \\ -8.33\times10^{-2} & 3.33\times10^{-1} & 2.00 & 3.33\times10^{-1} & -8.33\times10^{-2} \\ 4.44\times10^{-2} & -1.78\times10^{-1} & -1.07 & -1.78\times10^{-1} & 4.44\times10^{-2} \\ -2.78\times10^{-3} & 1.11\times10^{-2} & 6.67\times10^{-2} & 1.11\times10^{-2} & -2.78\times10^{-3} \end{bmatrix} \tag{A.9}$$

$$\boldsymbol{w_{m}} = \frac{1}{(\Delta y)^2}\begin{bmatrix} 1.11\times10^{-3} & -4.44\times10^{-3} & -2.66\times10^{-2} & -4.44\times10^{-3} & 1.11\times10^{-3} \\ -4.44\times10^{-3} & 1.78\times10^{-2} & 1.07\times10^{-1} & 1.78\times10^{-2} & -4.44\times10^{-3} \\ -2.67\times10^{-2} & 1.07\times10^{-1} & 6.40\times10^{-1} & 1.07\times10^{-1} & -2.67\times10^{-2} \\ -4.44\times10^{-3} & 1.78\times10^{-2} & 1.07\times10^{-1} & 1.78\times10^{-2} & -4.44\times10^{-3} \\ 1.11\times10^{-3} & -4.44\times10^{-3} & -2.67\times10^{-2} & -4.44\times10^{-3} & 1.11\times10^{-3} \end{bmatrix} \tag{A.10}$$

A.1.3 Cubic Filters

$$\boldsymbol{w_x} = \frac{1}{\Delta x} \tag{A.11}$$

$$\begin{bmatrix} -3.30 \times 10^{-4} & 2.26 \times 10^{-3} & -9.19 \times 10^{-3} & 0.00 & 9.19 \times 10^{-3} & -2.26 \times 10^{-3} & 3.30 \times 10^{-4} \\ 1.25 \times 10^{-3} & -8.57 \times 10^{-3} & 3.48 \times 10^{-2} & 0.00 & -3.48 \times 10^{-2} & 8.57 \times 10^{-3} & -1.25 \times 10^{-3} \\ -2.03 \times 10^{-3} & 1.39 \times 10^{-2} & -5.66 \times 10^{-2} & 0.00 & 5.66 \times 10^{-2} & -1.39 \times 10^{-2} & 2.03 \times 10^{-3} \\ -2.69 \times 10^{-2} & 1.85 \times 10^{-1} & -7.51 \times 10^{-1} & 0.00 & 7.51 \times 10^{-1} & -1.85 \times 10^{-1} & 2.69 \times 10^{-2} \\ -2.03 \times 10^{-3} & 1.39 \times 10^{-2} & -5.66 \times 10^{-2} & 0.00 & 5.66 \times 10^{-2} & -1.39 \times 10^{-2} & 2.03 \times 10^{-3} \\ 1.25 \times 10^{-3} & -8.57 \times 10^{-3} & 3.48 \times 10^{-2} & 0.00 & -3.48 \times 10^{-2} & 8.57 \times 10^{-3} & -1.25 \times 10^{-3} \\ -3.30 \times 10^{-4} & 2.26 \times 10^{-3} & -9.19 \times 10^{-3} & 0.00 & 9.19 \times 10^{-3} & -2.26 \times 10^{-3} & 3.30 \times 10^{-4} \end{bmatrix} \tag{A.12}$$

and similarly for $\boldsymbol{w_y}$ etc.