Titre: Title:	Parallel Discontinuous Finite Element SN Solver in Cartesian and Hexagonal Geometries for the Boltzmann Transport Equation in DRAGONS5
Auteur: Author:	Atyab Ahmad Calloo
Date:	2022
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Calloo, A. A. (2022). Parallel Discontinuous Finite Element SN Solver in Cartesian and Hexagonal Geometries for the Boltzmann Transport Equation in DRAGONS5 [Thèse de doctorat, Polytechnique Montréal]. PolyPublie. https://publications.polymtl.ca/10518/

URL de PolyPublie: PolyPublie URL:	https://publications.polymtl.ca/10518/
Directeurs de recherche: Advisors:	Alain Hébert
Programme: Program:	Génie nucléaire

POLYTECHNIQUE MONTRÉAL

affiliée à l'Université de Montréal

Parallel Discontinuous Finite Element S$_\text{N}$ Solver in Cartesian and Hexagonal Geometries for the Boltzmann Transport Equation in DRAGON5

ATYAB AHMAD CALLOO

Département de génie physique

Thèse présentée en vue de l'obtention du diplôme de *Philosophiæ Doctor*

Génie nucléaire

Août 2022

POLYTECHNIQUE MONTRÉAL

affiliée à l'Université de Montréal

Cette thèse intitulée :

Parallel Discontinuous Finite Element S$_\mathrm{N}$ Solver in Cartesian and Hexagonal Geometries for the Boltzmann Transport Equation in DRAGON5

présentée par **Atyab Ahmad CALLOO**
en vue de l'obtention du diplôme de *Philosophiæ Doctor*
a été dûment acceptée par le jury d'examen constitué de :

André GARON, président
Alain HÉBERT, membre et directeur de recherche
Guy MARLEAU, membre
Wesley FORD, membre externe

To mum & dad,

ACKNOWLEDGEMENTS

RÉSUMÉ

Cette thèse avait comme but de départ le développement et l'implémentation d'un solveur rapide d'ordonnées discrètes, dit S_N, pour les Réacteurs à Neutrons Rapides (RNRs) dans le code DRAGON5 de Polytechnique Montréal. Cela a entraîné une analyse et une étude de plusieurs aspects de l'algorithme de résolution : la discrétisation spatiale, le maillage hexagonal structuré, l'accélération synthétique et la parallélisation du calcul sur plusieurs processeurs. Chacune de ces quatre parties forme un morceau du puzzle qu'est cette recherche.

Une étude précédente avait démontré que les solveurs S_N étaient parmi les plus précis des méthodes déterministes pour la modélisation des RNRs. Cependant, dépendant des cas de figures, ils pouvaient aussi être 100 à 1000 fois plus lents que d'autres solveurs déterministes. Le désavantage de ces autres solveurs : ils étaient beaucoup moins précis, parfois de l'ordre de plusieurs centaines de pcm du calcul référence Monte-Carlo.

Les solveurs S_N utilisés étaient tous basés sur la méthode des éléments finis Discontinus de Galerkin (DG) pour la discrétisation spatiale. On s'est donc amené à questionner l'importance de cette méthode et l'avons implémenté dans DRAGON5. Une autre discrétisation, le schéma Différences Diamants (DD) d'ordre élevé, était aussi déjà implémentée dans le code. Puisqu'il n'y avait pas eu de comparaison de ces deux méthodes sur des problèmes à valeurs propres, on a creusé la question. On démontre qu'une fois les deux méthodes convergées spatialement et angulairement, les différences en termes de pcm sont négligeables.

Le maillage hexagonal intervenant principalement pour les RNRs, cette fonctionnalité n'était pas présente dans DRAGON5. Une revue de la littérature a démontré plusieurs façons de traiter le problème et on a choisi un sous-maillage des hexagones en losanges. Cette méthode, élégante dans son approche, permet de retrouver des éléments orthogonaux en utilisant une transformation affine. Les interventions sur le code étaient alors stratégiques mais minimales.

On a commencé à explorer l'accélération du calcul à travers l'utilisation d'une accélération synthétique. Cette méthode, bien établie, emploie en général l'équation de diffusion. On s'est attardé cependant dans cette thèse sur l'application du solveur SP_n de DRAGON5, discrétisé avec les éléments finis mixtes-duaux de Raviart-Thomas (RT). On baptise cette accélération RT-SP_nSA. Un travail préliminaire avait mis en place cette méthode mais elle était très instable pour les ordres spatiaux élevés et les conditions frontières de réflexion. Une nouvelle technique impose la correction du flux à travers des fractions mis à l'échelle pour les moments supérieurs. Cela permet donc de coupler l'ordre RT-0 avec n'importe quel ordre de l'équation du transport – ce qui diminue le coût. Cette méthode couplé avec une nouvelle

technique de corriger les flux aux bords s'est révélée être très efficace.

L'équation SP_n a aussi permis de tester des ordres angulaires et des sources de diffusion anisotropes. Ces paramètres améliorent légèrement la qualité des résultats mais c'est l'usage de l'accélération chaque deux à quatre itérations de transport qui a le meilleur résultat. On présente aussi une analyse Fourier qui démontre que la méthode n'est pas stable inconditionnellement même si elle démontre des signes de cohérence partielle. Testée sur des cas plus réalistes, on observe des réductions de temps de calculs d'environ 60% à 80%.

Finalement, contexte actuel oblige, la diminution du temps de calcul à travers la parallélisation sur plusieurs coeurs a été implémentée. La plupart des méthodes de parallélisation emploient ou sont basées sur la stratégie Koch-Baker-Alcouffe (KBA). Un maillage hexagonal présente des subtilités qui rendent le problème légèrement différent d'un maillage cartésien structuré. Cela porte principalement sur l'inter-connectivité de trois entre les hexagones et donc la distribution et flexibilité du nombre de macrocellules dans le plan hexagonal. On décrit ainsi la méthode de parallélisation mise en place, qui est basée sur la distribution des processeurs sur les octants, les macrocellules et les angles, en utilisant l'interface de programmation, *"Message Passing Interface" (MPI).* Cette implémentation a été faite dans une librairie à part, qu'on a nommé WYVERN, et qui fera bientôt partie du code DRAGON5.

Même si le code est loin d'être optimisé, on observe des diminutions de temps de calcul d'environ 40 à 80 fois, sur les benchmarks Takeda. La parallélisation semble mieux marcher sur le maillage hexagonal et l'on postule que c'est peut-être dû à une distribution non-égale des macrocellules mais de plus amples études sont nécessaires. Un modèle de parallélisation hybride permettrait aussi de tirer davantage du potentiel de tous les processeurs.

Le développement de toutes ces méthodes permet la simulation du coeur d'un RNR fictif avec trente-trois groupes d'énergies. Avec une approximation DG cubique et une quadrature S_{10}, ce benchmark possède environ 4.99×10^9 degrés de liberté. On estime le temps de calcul séquentiel non-accéléré à environ 130 jours. L'utilisation de WYVERN avec RT-SP_nSA sur 154 coeurs permet de réduire ce temps à environ 25 heures.

ABSTRACT

In this dissertation, we set out to develop and implement a fast discrete-ordinate (S_N) transport solver for Fast Neutron Reactors (FNRs) in the Polytechnique Montréal DRAGON5 code. This entailed the investigation of various aspects of the resolution algorithm: the spatial discretisation method, structured hexagonal resolution meshes, the synthetic acceleration and the parallelisation over several processors. These arguably disjointed research areas came together to make a cohesive whole in this goal (and solver).

It had been shown through a previous study that solvers based on the S_N method are among the most accurate for resolving FNRs. However, depending on the test case, they were also about two to three orders of magnitude slower than solvers based on other deterministic methods (P_n and SP_n) – although the latter were sometimes hundreds of pcm off from the the reference value. The S_N solvers used were based on the Discontinuous Galerkin Finite Element Method (DGFEM), so for this reason, we began there.

DGFEM is a spatial discretisation method that is quite popular in neutron transport theory. However, there had not been to date an eigenvalue problem comparison with the High Order Diamond Difference (HODD) method, which was already present in the code. After having implemented DGFEM using Legendre polynomials, we embarked on that. It was observed that once each method had converged angularly and spatially (for each spatial discretisation order), there was negligible differences between them. Moreover, linear DGFEM struggled more if the computational mesh was not sufficiently refined, compared to linear HODD.

FNRs mostly are based on a hexagonal geometry. This meant that the solver, which had essentially been developed to work with orthogonal grids, had to be modified accordingly. There are various ways of dealing with honeycomb meshes but after a review of the available literature, a lozenge-based submeshing approach was carried out. This was elegant in its implementation as an affine transformation of lozenges can yield square elements. This implied strategic but minimal modifications to the existing code. We took the opportunity to describe the implementation in detail as we found that to be quite lacking in the literature.

Subsequently, the acceleration of the code was explored through a synthetic acceleration. While the latter usually predominantly features the diffusion equation at its core, we made use of the DRAGON5 SP_n solver, discretised with the mixed-dual Raviart-Thomas (RT) finite elements, to give something we dubbed RT-SP_nSA. Even though a prior study set up the framework for this, the acceleration was widely unstable at higher polynomial orders in the transport equation and with reflective boundary conditions (b.c.). A correction approach

where the zeroth moment correction is applied as a scaled fraction to the higher moments proved to be very successful. This also allowed the use of the zeroth order of RT with every order of the transport equation for minimum intensity. A novel approach to deal with reflective boundary conditions is also outlined in this work.

The use of SP_n allowed the testing of variable angular orders and anisotropic scattering sources for the acceleration. We found that, while they helped to some extent for highly anisotropic problems, nothing was as impactful as applying the acceleration every two or four transport iterations. A Fourier Analysis (FA) of RT-SP_nSA demonstrated that it was not unconditionally stable, although it showed signs of stability that indicated partial consistency. Ultimately, when tested on benchmarks that are more representative of real-world problems, we found reduction in calculation times ranging from 60% (reflective b.c.) to 80% (void b.c.).

Finally, increasingly available and powerful computational resources along with more accessible parallelisation strategies make High Performance Computing (HPC) an inevitable solution to diminishing calculation times. In S_N neutron transport, most parallelisation strategies are based on the Koch-Baker-Alcouffe (KBA) algorithm or variations thereof. A structured hexagonal mesh presents subtle differences with the Cartesian structured grid that we outline in this document. This is mostly regarding the three-fold inter-connectivity of hexagons and consequently, the distribution and flexibility in the amount of macrocells in the hexagonal plane. We also describe the parallelisation strategy chosen, based on a distribution of available processes over a virtual processor grid of octants, macrocells and angles, using the Message Passing Interface (MPI) model. The implementation was done in a separate library called WYVERN, which is expected to become part of the main DRAGON5 code library at some point this year.

Even though the code is far from optimised at this point in time, we observed gains in computational times of up to $40\times$ in Cartesian geometry and up to $80\times$ in hexagonal geometry, in the benchmarks run. While we hypothesise that this difference might be due to the difference in how the macrocells are distributed, more investigation is needed. Hybrid parallelisation strategies should also be considered to extract the most out of the available hardware.

For now, having implemented all the methods and algorithms described, we simulated a mock three-dimensional hexagonal Fast Breeder Reactor (FBR) core with thirty-three energy groups. With an S_{10} quadrature, this corresponded to 6.23×10^8 unknowns for a linear DGFEM approximation and 4.99×10^9 unknowns for cubic. Using a lower order (S_2) calculation, the unaccelerated sequential run times were estimated at about 29 hours and 130 days respectively. Using WYVERN on 154 cores, with RT-SP_nSA, the calculations were brought down to about 75 minutes and just over 25 hours.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ACRONYMS

Nuclear Reactors

ASTRID   Advanced Sodium Technological Reactor for Industrial Demonstration

BWR   Boiling Water Reactor

CANDU   CANada Deuterium Uranium

FBR   Fast Breeder Reactor

FNR   Fast Neutron Reactor

GFR   Gas-cooled Fast Reactor

LFR   Lead-cooled Fast Reactor

NPP   Nuclear Power Plant

PWR   Pressurised Water Reactor

SFR   Sodium-cooled Fast Reactor

SMR   Small Modular Reactor

VVER   *Vodo-Vodyanoi Enyergeticheskiy Reaktor*

Institutions & Companies

ARC   Advanced Reactor Concepts

CEA   Alternative Energies and Atomic Energy Commission (*French:* Commissariat à l'énergie atomique et aux énergies alternatives)

CNSC   Canada Nuclear Safety Commission

DRAC   Digital Research Alliance of Canada

GEH   GE-Hitachi

GIF   Generation IV International Forum

TAMU   Texas A&M University

Transport Theory & Numerical Methods

b.c.   boundary conditions

BTE   Boltzmann Transport Equation

CMR   Coarse Mesh Rebalance

DD   Diamond Difference

DFEM   Discontinuous Finite Element Method

DG   Discontinuous Galerkin

DGFEM   Discontinuous Galerkin Finite Element Method

DPGFEM   Discontinuous Petrov-Galerkin Finite Element Method

DSA   Diffusion Synthetic Acceleration

FA   Fourier Analysis

FEM   Finite Element Method

FVM   Finite Volume Method

HODD   High Order Diamond Difference

LS   Level Symmetric

M4S   Modified Four Step

MMS   Method of Manufactured Solutions

MFP   Mean Free Path

NSR   Numerical Spectral Radius

PDE   Partial Differential Equation

RT   Raviart-Thomas

SA   Synthetic Acceleration

SI   Source Iteration

Parallelisation

API	Application Programming Interface
CPU	Central Processing Unit
DAG	Directed Acyclic Graph
GPU	Graphic Processing Unit
HPC	High Performance Computing
KBA	Koch-Baker-Alcouffe
MIMD	Multiple Instruction, Multiple Data
MISD	Multiple Instruction, Single Data
SIMD	Single Instruction, Multiple Data
SISD	Single Instruction, Single Data
SPMD	Single Program, Multiple Data
MPI	Message Passing Interface
OpenMP	Open Multi-Processing
PCE	Parallel Computational Efficiency
PUE	Processor Usage Efficiency

Other

AIC	Ag-In-Cd (Silver-Indium-Cadmium)
COB	Change-Of-Basis
CRP	Control Rod Position
d.o.f.	degrees of freedom
PDF	Probability Density Function

LIST OF APPENDICES

CHAPTER 1   INTRODUCTION

This chapter presents a brief historical account of nuclear energy as well as an outline of the relevant nuclear reactors. While this might not be immediately pertinent to the project, it serves to frame how and where this project hopes to contribute to making a difference. Some prior nuclear and reactor physics is assumed; however, most of the necessary concepts are recapitulated in Chapter 2. This chapter goes on to describe the preceding study that directly provoked this work and sets forth the scope of this dissertation. It then concludes by outlining the rest of the document.

1.1   Background History and Current Status of Nuclear Power

The neutron was discovered by James Chadwick ninety years ago, in 1932. From that point on, things would progress rapidly. In 1934, Enrico Fermi used neutron bombardment to create artificially radioactive nuclei, while also demonstrating that slow neutrons were more efficient for this purpose. In 1938, the term 'nuclear fission' was coined by Otto Frisch after Otto Hahn and his collaborators successfully fragmented the uranium nucleus into lighter nuclei. In 1942 – a mere four years later, during the Manhattan project, the first human-made self-sustaining chain reaction was set up in Chicago Pile-1 (CP-1). *The first human-made nuclear reactor had been invented.*

Over the course of World War II and development of the atomic weapon, the West and Soviet Union had acquired technologies that scientists realised could be used for commercial electricity production.[1] Hence, after the war, focus very much shifted to that, especially after President Eisenhower's "Atoms for Peace" speech to the UN General Assembly in 1953. Over the '50s, '60s and '70s, the United States, Canada, Europe (mostly the United Kingdom and France) and the Soviet Union concurrently would test out a number of ideas. The Soviet Union built the first power plant intended for civil purposes in Obninsk in 1954. Lacking access to enrichment technologies, the United Kingdom built a number of gas-cooled graphite-moderated reactors using natural uranium. France took a similar approach before settling on the now more classic design of Pressurised Water Reactors (PWRs) which are light-water-cooled and light-water-moderated, and use enriched uranium. The latter had arisen from significant research done by the US Navy, initially meant to power submarines – water being the easily-accessible commodity in that case. The US also developed the Boiling Water Reactor (BWR). Canada went yet another route, using heavy water[1] and natural uranium

---

[1]Heavy water is water, H_2O, where the hydrogen atoms have an extra neutron in the nucleus.

to power the CANada Deuterium Uranium (CANDU) reactors.

While as from the early '80s, the industry seemed to decline somewhat with many planned reactors being cancelled and few new orders, the industry would pick up again in the late '90s and early 2000s with the new wave of so-called third-generation reactors. These are similar to the reactors mentioned so far in that they are *thermal reactors* too – albeit with much improved designs in terms of safety and efficiency. In thermal reactors, after neutrons are produced during the fission process, they have to be slowed down – also called *moderated* – before they are able to cause further fission and sustain a chain reaction. The term *thermal* stems from the fact that neutrons need to lose energy to the point that they are in near thermal equilibrium with uranium atoms. A brief overview of the inner workings of a nuclear reactor is given in Sec. 1.2. However, specifics have been covered extensively in the literature and will not be included in this thesis. We recommend the text in [2] and the references therein for further reading. A handful of third-generation reactors are currently being built in Europe and the US. However, the industry boomed in Asia, more specifically in China and India. China alone has seventeen reactors currently under construction, all planned to go into operation before 2027. The US, UK and France have only six combined. [3]

As we just discussed, so far, the reactors have all been thermal reactors. However, there is another broad subset of nuclear reactors called Fast Neutron Reactors (FNRs). As the name suggests, the aforementioned moderating effect is not at all desired in FNRs so as to keep the neutrons at much higher energies. This leads to differences in design, coolant and fuel composition amongst other things but therein lies the main difference. Experimental and even commercial prototypes of *FNRs* were also built (by most of the countries mentioned), though much less extensively. In fact, the first ever "nuclear power plant" was an FNR; it was the Experimental Breeder Reactor I (EBR-I) in Idaho in the US and powered four 200 W bulbs on the 20th of December 1951. Water could not be used as a coolant because of its moderating properties. Hence, a liquid metal mixture of NaK (sodium-potassium) was used. Despite a partial core meltdown during a scheduled transient test where the main coolant flow was stopped, the results of subsequent testing were so overwhelmingly positive that a second one, the EBR-II was built and started operations in 1964. This should have been followed by the Integral Fast Reactor (IFR). However, for mostly political reasons, this was cancelled in 1994. A similar situation repeated itself in France with the Phenix and Superphenix reactors in the '90s and early 2000s. In spite of these cancellations and shut-downs, the EBR and Phenix series were both highly successful in their goals. Between them, they demonstrated the feasibility of breeding more fuel, the use of liquid metal sodium as a coolant, the ability to burn nuclear waste from thermal reactors and the possibility to operate safely.

ASTRID, the Advanced Sodium Technological Reactor for Industrial Demonstration, was a planned experimental demonstrator Sodium-cooled Fast Reactor (SFR) in France, with an output of 100-200 MWe[2]. It subsequently became a partnered project between France, Japan and the USA. Unfortunately, France indefinitely shelved the project in 2019. However, there definitely seems to be some corporate interest in the US again regarding FNRs. GE-Hitachi (GEH) introduced the PRISM reactor, based on the EBR and IFR designs. It is a compact modular SFR currently being marketed to the UK government as a way of burning its stockpile of plutonium. GEH is also collaborating with a relatively new nuclear reactor design company, TerraPower on the *Natrium* concept [4]. This utilises the PRISM reactor coupled with a thermal storage system of molten salt. In this way, the reactor can always operate at full capacity and provide the follow-through power from the thermal storage during peak demand. They recently obtained a grant from the government and released plans for a demonstration unit in Wyoming.

Another design currently being developed is one by Advanced Reactor Concepts (ARC) for a fast sodium-cooled Small Modular Reactor (SMR) with an output of 100 MWe, also based on the EBR-II. Back in March 2017, ARC and GEH agreed to pool their resources together to collaborate on the design of this reactor. They were looking to deploy in Canada and started the regulatory review process with the Canada Nuclear Safety Commission (CNSC). In 2019, they successfully cleared Phase 1 of the preliminary review process and entered Phase 2. [5] Just recently, in April 2022, they announced that they were able to raise funds from the private sector and the province of New Brunswick. [6] This marks significant progress for the deployment of their SMR at the Point Lepreau Nuclear Generating Station (PLNGS) site, owned by New Brunswick Power (NB Power).

There has also been significant progress in the field of FNRs in the eastern hemisphere. The Prototype Fast Breeder Reactor (PFBR), a fast reactor of 500 MWe, has been built in India. It has, unfortunately, been plagued by delays and budget increases. Construction started in 2004 and it was initially supposed to be commissioned in 2010 but is now scheduled to go fully operational only by the end of this year, 2022. [7] In China, the China Experimental Fast Reactor (CEFR), rated at 20 MWe and located outside Beijing, first became critical in 2010. It successfully maintained full operational power for three days in 2014 [8]. It is assumed that it still operational now; not much information seems to be available. China is also working on the China Fast Reactor 600 (CFR-600), a demonstrator reactor under construction, expected to be connected to the grid in 2023. Japan currently only has the Jōyō experimental research SFR. The Monju reactor, afflicted by incidents, mismanagement

---

[2]*MWe* stands for 'MegaWatt electric', where Watt is the SI unit of power and the term *electric* signifies that it is the actual amount of power output that has been successfully converted from thermal power.

and negative public opinion was never a success, and closed down in 2016.

All that being said, Russia is perhaps the most successful with three operational FNRs and two in construction. The BOR-60, an SFR, has been in operation since December 1969 and had had its licence extended to 2020 [9]. It is unclear whether this has been extended again. There is also the SFR BN-series: the BN-600 and its larger sibling, the BN-800, so named after their nominal electrical power (600 MWe and 864 MWe). The former has been in operation since 1980 and the latter achieved full power in August 2016. They are both connected to the electrical grid. A third proposed reactor, the BN-1200, has now been shelved for decision until the mid 2030s [10]. The two in construction are the BREST reactor, a pilot demonstrator Lead-cooled Fast Reactor (LFR), and the MBIR, a multipurpose research reactor capable of using lead, lead-bismuth mixture, sodium and gas as coolants.

It is perhaps impossible, however, to talk about FNRs without at least mentioning the Generation IV International Forum (GIF). In 2001, through an effort of the US Department of Energy, the international nuclear community convened to create this organisation. This was in recognition of the potential that nuclear energy and advanced reactor designs held but also the acknowledgement that it would take collaboration and cooperation to bring them to fruition. After reviewing roughly a hundred concepts, six reactor designs were chosen to focus research on. They would address a certain number of concerns: environmental and economical sustainability, optimal usage of resources (ideally through a closed fuel cycle), inherent design safety, and resistance against proliferation, amongst others. Six designs were retained, these being the SFR, the SuperCritical Water-cooled Reactor (SCWR), the LFR, the Gas-cooled Fast Reactor (GFR), the Molten Salt Reactor (MSR), and the Very High Temperature Reactor (VHTR).[11, 12] The SFR is undoubtedly the one where the most experience lies, and as we have seen, this is where countries seem to be pouring a lot of attention. This particular research will focus on numerical and computational methods for the modelling of the neutron distribution in SFRs – and PWRs too along the way.

1.2 Basic Concepts of Nuclear Reactors

The primary idea behind nuclear reactors is that they convert the energy released by fission of heavy nuclei in the nuclear reactor core into electricity. These heavy nuclei can be elements such as Uranium-235, Uranium-238 and Plutonium-239[3] amongst others. The number after the element is indicative of the total number of protons and neutrons in the nuclei of atoms. For example, considering that the number of protons is a constant for elements – 92 in the

---

[3]These are oft abbreviated as U-235, U-238 and Pu-239.

case of Uranium, Uranium-238 has 146 neutrons, three more than Uranium-235.

The fission reaction itself is induced by an incident neutron. This, upon collision with a heavy nucleus (such as the aforementioned ones), has a certain probability of splitting it into usually two daughter nuclei, as well as two or three additional neutrons and an average energy of 200 MeV in the form of heat. This released heat is then carried away from the core by a coolant. If this coolant is water, it can be turned into steam immediately and drive conventional turbines to generate electricity. More often however, this coolant is enclosed in a primary circuit, and there is an exchange of heat with a secondary circuit of water/steam which runs the turbines.

The extra neutrons resulting from the fission process can then generate more fission reactions. In time, this can set up a self-sustaining chain reaction. However, this depends entirely on having the right conditions, amongst which are the type of reactor, its geometry, its dimensions, and its fuel composition.

For example, if we consider the fuel, the isotope U-238 makes up for a much more substantial proportion of naturally-occurring uranium, at around 99.3%. This and U-235, the other isotope making up most of the rest, have vastly different *cross sections* of neutron capture and fission. The cross section – commonly denoted as σ_x with the subscript x indicative of the type of interaction – can be understood as the probability of interaction. Fig. 1.1 shows graphs of cross sections versus incident neutron energy for U-235 and U-238.

It can be seen that Uranium-235 has a much higher probability of fission at low energies. Coupled with the fact that more uranium reserves (than were initially thought to exist) were discovered, this pushed early reactor development towards thermal reactors. These slow down the neutrons in a process called *moderation*, which is achieved through nearly-elastic collisions with light nuclei, generally the hydrogen atoms in water molecules. At low energies though, the probability of radiative capture[4] also increases for both isotopes (as seen on Fig. 1.1) as well as for the water that is usually being used as moderator. Because of this complication, within an industrial-sized reactor, there will not be enough neutrons emitted through fission to provide a self-sustaining reaction with natural uranium. As such, uranium enriched in U-235 normally has to be used as fuel.

Today, the most common type of nuclear reactor around the world is the Pressurised Water Reactor (PWR). These use slightly-enriched (2.0 - 5.0%) uranium dioxide pellets as fuel, clad with a zirconium metal alloy (Zircaloy). Light water is used both as a moderator and a coolant. PWRs also generally include two coolant loops to reduce the spread of radioactivity

---

[4]Radiative capture is a reaction where the incident neutron is captured by the nucleus and a gamma photon is emitted, but no fission is induced.
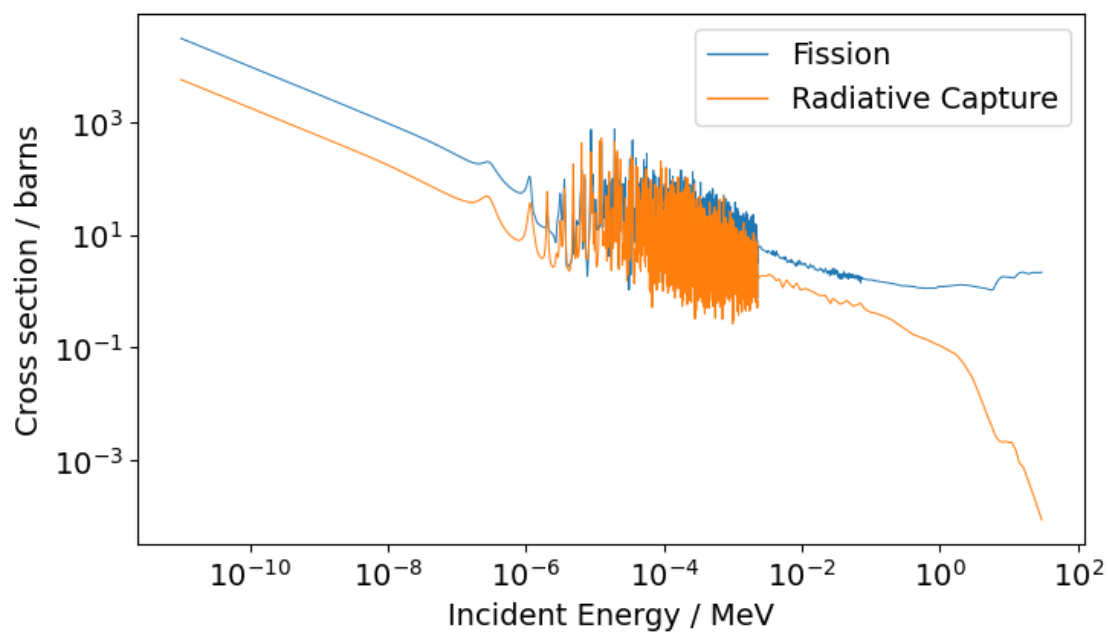
(a) U-235
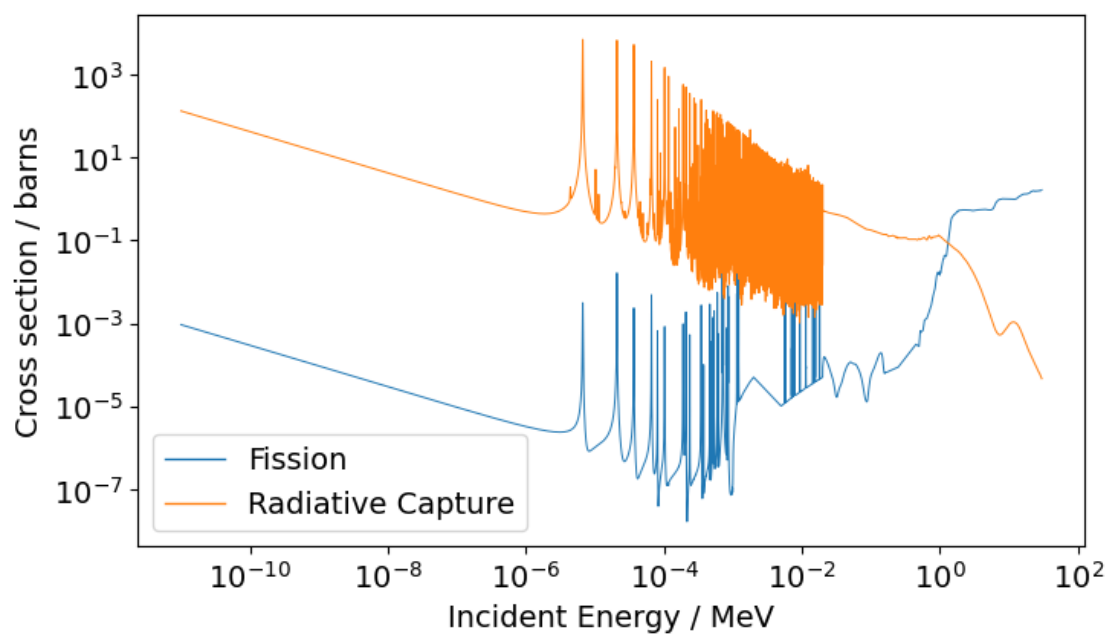
(b) U-238

Figure 1.1 Plots of cross section against incident energy for the fission and radiative capture reactions for Uranium-235 and Uranium-238. Data for plots obtained from `https://www.nndc.bnl.gov/endf`.

within the system. The water in the primary loop stays liquid under high pressure while the water in the second loop is turned into steam to drive the turbines.

Three other types of reactors are the Boiling Water Reactor (BWR), the *Vodo-Vodyanoi Enyergeticheskiy Reaktor* (VVER), and the CANada Deuterium Uranium (CANDU) reactor. The first is very similar to PWRs, except that it consists of a single coolant loop where the water is turned into steam in the reactor pressure vessel itself. The second is also very much like PWRs but feature a hexagonal lattice and is surrounded by a heavy reflector made of stainless steel. VVERs are present in Russia but also India, Turkey, Bulgaria, Slovakia, Finland and Ukraine, amongst other countries. There is a concern about supporting VVERs in European countries for safety analyses and operation. CANDUs, on the other hand, use natural uranium as fuel. To overcome the problem of the resulting lower neutron economy, heavy water is used as moderator and coolant instead of light water. Since the hydrogen already has an extra neutron, it has a much lower affinity for neutron absorption.

In addition, as noted in the preceding section, on the opposite side of thermal reactors are FNRs. These are designed to keep the neutrons at high energies and operate under these conditions. If we look at Fig. 1.1 again, at high energies, we can see that while the absolute probability of fission is lower, the relative probability taken with respect to the probability of radiative capture is much higher. This is the case for *both* uranium isotopes. And while it is difficult to maintain the neutrons at very high energies constantly, there can be much more fission of Uranium-238 now, since the absolute probability of fission of U-238 is higher than that of radiative capture. As an aside, many of the long-lived radioactive waste isotopes have a similar behaviour. This is yet another advantage of FNRs in that they are able to burn these as fuel and get much shorter-lived radioactive waste as as result.

On top of that, FNRs tend to have a better neutron economy than thermal reactors. This means that overall, considering neutrons lost to capture, leakage and otherwise, it is possible for more than one net neutron to result from each fission reaction. Hence, there are now neutrons available to *breed more fuel.* Put simply, one way of achieving this would be to turn Uranium-238 into Uranium-239 via neutron capture and subsequently, into Plutonium-239 via two beta-decays[5]. This, coupled with the higher fission rate of Uranium-238 means that FNRs make much more efficient use of the fuel element (U-238) which is mostly unused in thermal reactors, stretching the available uranium reserves available today that much further into the future.

Another aspect that tends to be specific of FNRs – albeit not exclusively: for *e.g.*, VVERs

---

[5]A beta-decay (or β-decay) is a radioactive process whereby a neutron spontaneously undergoes a transformation into a proton accompanied by the emission of an energetic electron (β particle) and an antineutrino.

too – is the hexagonal design of the reactor fuel pins. For FNRs at least, this is a direct consequence of requiring the fission reactions to occur at high energies. Indeed, for a planar geometry, hexagons are the best polygons when it comes to packing [13] as efficiently and as closely as possible[6]. If neutrons are able to cause fission reactions soon after being born, they would have lost as little energy as conceivable. This is why it is desirable and important to have the fuel pins well packed. For this reason, FNR fuel pins are hexagonal, arranged in a honeycomb pattern, and ultimately the reactor core tends to be in the same shape too.[7] Also, again because of the need to maintain the neutrons at high velocities, water cannot be used to siphon heat away from the fuel in FNRs. For that reason, liquid metal or liquid metal alloys are used. These moderate much less and have better heat transfer properties than water, as well as contribute to passive safety features by allowing for natural circulation of heat in accident scenarios.

Unfortunately, this can also be a drawback. An SFR reactor for example uses liquid sodium. This requires exotic materials for the piping and containment, not to mention the fact that sodium spontaneously catches fire when exposed to air. Other drawbacks include shorter timescales because of fewer delayed neutrons leading to a higher level of unpredictability; a positive void coefficient for the coolant; a higher fissile load required for start-up; and much less experience in operation reactor-years compared to thermal reactors.

1.3 Scope of Research Project

The modern study and safe operation of a nuclear reactor, whether thermal or fast, relies on the computational simulation of its reactor core and power distribution. Accurately representing the growth and distribution of the neutron population is key since neutrons are the direct cause of fission – and hence, release of energy. In fact, the mathematical modelling of neutral particle steady-state transport relies on the representation of movement of these particles in a six-dimensional phase space[8] as influenced by their interaction with underlying materials. This can be described by the *Boltzmann Transport Equation (BTE)*, a linear balance equation. Fast and accurate solutions to the BTE are central to the safe controlled operation of Nuclear Power Plants (NPPs). Owing to the complexity of the equation, computer simulations are always employed to solve the equation in real-world scenarios.

---

[6]While this was first postulated as far as back as 36 BC, it was only proven in 1999 by Hales [13]. Regardless of the proof, this was a well-accepted statement that saw widespread use in industry after taking inspiration from nature [14].

[7]As an aside, thermal reactors usually have a square-ish design because of the need to moderate the neutrons in between its birth and subsequent collision with a fuel nuclei. Therefore, in this case, some distance is a desirable feature.

[8]In steady-state, these are space (3 degrees of freedom (d.o.f.)), energy (1 d.o.f.), and direction (2 d.o.f.).

Numerous solvers and codes currently exist to solve the BTE. However, these were mostly built and optimised with thermal reactors in mind. Fast reactors, as we have just seen, have important differences – the neutron energy spectrum for one of course, but also the design of the core and the layout of the various components inside it. Of course, with the right data set, one could expect that the numerical methods that are the backbone of these solvers, should also work well for fast reactors. That said, it is known that numerical methods can be quite fickle in that respect. And even if they do work well, the question was also one of their performance, in terms of accuracy, speed, and efficiency.

This is exactly what a previous study by Bay [15], endeavoured to quantify in 2013. They had access to several French software as well as the Polytechnique Montréal in-house code, DRAGON5/DONJON5. Some of them had similarities in between them, which made it useful for comparison purposes to ensure that results were consistent across different solvers. Bay compared these solvers in their ability to accurately model a small SFR core for an eigenvalue problem – more on that in Chap. 2. The benchmarks were based on the Takeda Model 4 [16]: the original benchmark as well as one with modifications to more closely resemble the CFV[9] variant [17] of the ASTRID reactor core. These solvers and their respective code or platform were TRIVAC5/DONJON5; SNATCH/PARIS; VARIANT/ERANOS; MINARET/APOLLO3; and MINOS/APOLLO3. [15, 18, 19, 20, 21]

The specifics of each solver are not really important for this discussion. What is, however, are Bay's results. They found that solvers based on the discrete ordinates (S_N)[10] method were overall much more accurate than both P_n and SP_n[10] solvers, with P_n being the better of the two – around up to one order of magnitude for P_n and around up to two for SP_n. However, S_N solvers were also slower – several orders of magnitude slower in fact. Ultimately, Bay's recommendation was that P_n solvers seem to represent a good middle-ground between accuracy and speed. However, none of the S_N solvers they tested had all of the following aspects:

- able to use and compare both the High Order Diamond Difference (HODD) and the Discontinuous Galerkin Finite Element Method (DGFEM) spatial discretisations;

- able to model in Cartesian (1D/2D/3D) and hexagonal geometries (2D/3D);

- a Synthetic Acceleration (SA) in lozenge-based hexagonal geometries; and

- a parallelised sweeping algorithm, made rapid in its execution by using High Perfor-

---

[9]French: *Coeur à Faible Vidange* – a core with low reactivity effect in case of sodium drainage.

[10]Method(s) of discretising the direction (angular) phase space. Th(-is/-ese) will be elaborated upon to some extent in Chap. 2.

mance Computing (HPC) methods such as *Single Program, Multiple Data (SPMD)* constructs.

What we are trying to express with all of this terminology here – and we do ask the reader to please bear with us until Chap. 2 for more details on these – is that with relatively recent advances in both numerical methods analysis and high-performance computing, there are ways of making the S_N solver (*i.e.*, the more accurate one) much faster, without comprising on accuracy of the computation. And in this way, Bay's work forms the basis of this dissertation. *By implementing, testing and researching the shortcomings listed above, what are the answers that can be obtained? And by how much is it possible to reduce the computational time of an S_N solver with the resources that we currently have at our disposal?*

Therefore, the overarching goal of this research and dissertation is *the development in the code* DRAGON5 *of a parallel discontinuous finite element S_N solver in Cartesian and hexagonal geometries for the Boltzmann transport equation.*

Through previous projects at Polytechnique Montréal, an S_N HODD solver for the Cartesian domain had already been implemented, along with a first draft of a compatible acceleration technique. This was the starting point. It was decided to also implement DGFEM since the solvers tested by Bay all relied on this spatial discretisation. There was also a certain belief in the community that DGFEM was just better. As such, it was deemed interesting to compare the two techniques. And because this work sits within the framework of developing an accurate and precise solver for FNRs, we decided to implement high-order polynomials – up to at least parabolic, and cubic in the case of DG.

The next step was handling the hexagonal geometry that is intrinsic to VVERs and FNRs. There are a few ways to do so. However, for reasons of convenience, a lozenge-based approach, coupled with an affine transformation, was chosen. This method, splits each hexagonal mesh element of the domain into three lozenges. These can then be transformed into the Cartesian reference mesh elements. This has a few advantages but the obvious and biggest one is that it enables us to build upon the existing Cartesian solvers.

A significant portion of time was also spent on the investigation and optimisation of the SA (*i.e.*, the acceleration technique) that was previously set up with HODD. The search for consistent or compatible fast SAs in the industry is an ongoing and seemingly never-ending one. As it will be seen, many improvements and additions were made to the method, and it was coupled with DGFEM too. Several analyses based on the Fourier Analysis (FA) technique (which is the standard to use in the field) were run for both methods, as well as numerical benchmarks to evaluate its performance.

On top of accelerating the computation through numerical methods such as the Diffusion Synthetic Acceleration (DSA), it can also be done through High Performance Computing (HPC) – essentially throwing more computer cores (Central Processing Units (CPUs)) at it. This was the last step in this work. A couple of parallelisation solutions were investigated: the Open Multi-Processing (OpenMP) Application Programming Interface (API) and Message Passing Interface (MPI) standard. OpenMP was found to be lacking for our applications and we shifted mostly to MPI. This part of the work was done in a separate library called WYVERN. While there were substantial gains in the computation times, these more or less levelled off after a hundred cores or so. This prevented a *massively* parallel implementation. However, what has been done represents an important first step for the S_N solver.

1.4 Thesis Outline

Owing to the arguably somewhat disjointed nature of the parts that make up this dissertation, a traditional literature review chapter has been eschewed. Instead, the next chapter will succinctly summarise the basic concepts of reactor and nuclear physics as well as give a brief presentation of the DRAGON5 code. Chapters 3 to 6 will then address each of the four main topics in the order: spatial discretisation, hexagonal geometry, synthetic acceleration and parallelisation. Each will be something of a self-contained part with its own literature review, theory, numerical results and discussion. The last chapter will then summarise the whole dissertation and its major findings, before finally taking a look towards the future.

CHAPTER 2 REACTOR PHYSICS BACKGROUND

The aim of this chapter is to provide more technical – albeit still general – background information which will serve as a foundation for the rest of the project. It will go over pertinent already-established knowledge on or around the project. Most of these elements would have been constantly used for this work, but might not have been directly modified, implemented or researched. For this reason, an overview of these finds its place here. It is therefore meant either as a refresh for the already knowledgeable reader, or as a guide for the unfamiliar one.

Considering the popularity and widespread application of the Boltzmann Transport Equation (BTE), its full derivation is omitted here, and it is only briefly described. A more complete presentation can be found in the book by Hébert [22]. The focus of this chapter shifts instead to the deterministic resolution of the transport equation. The discretisation of its various phase spaces is outlined – in more or less detail depending on their relevancy to this project. Also, due to its complex nature, the transport equation requires several iterative loops for its resolution. An overview of these is presented, leading to a description of the general algorithm. Finally, a concise description of the code used, DRAGON5, for most of the development work is given.

2.1 The Boltzmann Transport Equation (BTE)

The Boltzmann Transport Equation (BTE) is an integrodifferential equation governing the balance of neutrons. Considering an infinitesimal phase space volume, $d\boldsymbol{r}d\boldsymbol{\Omega}dE$, in the geometric domain $\mathcal{D}$, defined in $\mathbb{R}^3$, with boundary Γ, it is given by:

$$
\begin{aligned}
\frac{1}{v}\frac{\partial\psi}{\partial t}(\boldsymbol{r},E,\boldsymbol{\Omega},t) = -&\underbrace{\boldsymbol{\Omega}\cdot\nabla\psi(\boldsymbol{r},E,\boldsymbol{\Omega},t)}_{\text{Streaming loss}} - \underbrace{\Sigma(\boldsymbol{r},E,t)\psi(\boldsymbol{r},E,\boldsymbol{\Omega},t)}_{\text{Collision}} \\
&+ \underbrace{\int_0^\infty dE'\int_{4\pi}d^2\Omega'\ \Sigma_s\left(\boldsymbol{r},E\leftarrow E',\boldsymbol{\Omega}\leftarrow\boldsymbol{\Omega}',t\right)\psi\left(\boldsymbol{r},E',\boldsymbol{\Omega}',t\right)}_{\text{Scattering}} \\
&+ \underbrace{\sum_{j=1}^{J_{\text{fiss}}}\frac{\chi_j(\boldsymbol{r},E)}{4\pi}\int_0^\infty dE'\ \nu\Sigma_{f,j}\left(\boldsymbol{r},E',t\right)\phi\left(\boldsymbol{r},E',t\right)}_{\text{Fission}} \\
&+ \underbrace{Q_{\text{ext}}\left(\boldsymbol{r},E,\boldsymbol{\Omega},t\right)}_{\text{External source}},
\end{aligned}
\tag{2.1}
$$

where

- v – the neutron speed ;

- $\psi(\boldsymbol{r}, E, \boldsymbol{\Omega}, t)$ – the angular flux of particles at the point $\boldsymbol{r}\,(= \{x, y, z\} \in \mathbb{R}^3)$, in direction $\boldsymbol{\Omega}\,(= \{\theta, \varphi\} \in [0, \pi] \times [0, 2\pi])$ (where θ and φ are the polar and azimuthal angles respectively) with energy E, at time t ;

- $\phi(\boldsymbol{r}, E', t)$ – the integrated flux of particles over the unit sphere, given by

$$\phi(\boldsymbol{r}, E', t) = \int_{4\pi} d^2\Omega \; \psi(\boldsymbol{r}, E', \boldsymbol{\Omega}, t) \; ; \tag{2.2}$$

- $\Sigma(\boldsymbol{r}, E, t)$ – the total macroscopic cross section[1], which comprises all the possibilities of interaction ;

- $\Sigma_s(\boldsymbol{r}, E \leftarrow E', \boldsymbol{\Omega} \leftarrow \boldsymbol{\Omega}')$ – the scattering macroscopic cross section at the point $\boldsymbol{r}$ for a neutron with direction $\boldsymbol{\Omega}'$ and energy E' which is *scattered* into the direction $\boldsymbol{\Omega}$ with energy E ;

- J_{fiss} – the number of fissionable isotopes, with each represented by subscript j ;

- $\chi(\boldsymbol{r}, E)$ – the fission spectrum of the neutrons ;

- $\nu\Sigma_{f,j}(\boldsymbol{r}, E')$ – the macroscopic cross section for neutron production by fission for each fissionable isotope, j, multiplied by ν, the average number of neutrons produced per fission ; and

- $Q_{\text{ext}}\,(\boldsymbol{r}, E, \boldsymbol{\Omega}, t)$ – an external source of neutrons if present.

Quite evidently, the term on the left hand side is the rate of change of neutrons. On the right hand side, the first two terms are respectively the neutrons streaming out of the considered infinitesimal phase space volume and neutrons colliding with nuclei – both of which represent losses. The other terms describe neutron production: neutrons scattering into the phase volume, fission and an external source. The division by 4π in the fission term indicates that neutrons produced by fission are assumed to be emitted in an isotropic distribution. Equation 2.1 can also be condensed in operator form as

$$\frac{1}{\nu}\frac{\partial\psi}{\partial t}(\boldsymbol{r}, E, \boldsymbol{\Omega}, t) = -\mathcal{L}\psi(\boldsymbol{r}, E, \boldsymbol{\Omega}, t) + \mathcal{S}\psi(\boldsymbol{r}, E, \boldsymbol{\Omega}, t) + \mathcal{F}\phi(\boldsymbol{r}, E, t) + Q_{\text{ext}}\,(\boldsymbol{r}, E, \boldsymbol{\Omega}, t) \;, \tag{2.3}$$

where $\mathcal{L}$ is the streaming and collision operator, $\mathcal{S}$ the scattering operator and $\mathcal{F}$, of course, the fission operator.

---

[1]The cross section is a property of the material in which the neutron is travelling and can be understood as a measure of the probability of an interaction event.

The two most common boundary conditions used in reactor physics for the transport equation are the vacuum and reflective boundary conditions, both of which can represented with the albedo boundary condition which is given as, $\forall \boldsymbol{r} \in \Gamma_-$

$$\psi(\boldsymbol{r}, E, \boldsymbol{\Omega}, t) = \beta \; \psi(\boldsymbol{r}, E, \boldsymbol{\Omega}', t) \; , \tag{2.4}$$

where Γ_- is the inflow boundary of $\mathcal{D}$, defined by $\Gamma_- = \{\boldsymbol{r} \in \Gamma : \boldsymbol{\Omega}_n \cdot \boldsymbol{n}(\boldsymbol{r}) < 0\}$, with $\boldsymbol{n}$ being the unit outward normal to Γ, and where $\boldsymbol{\Omega}'$ is the direction of the outgoing particle. For vacuum boundary conditions (b.c.), $\beta = 0$ while for reflective b.c., $\beta = 1$ and $\boldsymbol{\Omega} \cdot \boldsymbol{n} = -\boldsymbol{\Omega}' \cdot \boldsymbol{n}$.

2.1.1 Steady state

A nuclear reactor operating under normal conditions is considered a multiplying medium because of the presence of fissionable material which can lead to a cascade of neutrons. Such a system is a critical when the rate of production of neutrons is equal to the rate of removal of neutrons from the system, *i.e.*, when there is a time-independent self-sustaining chain reaction. If more neutrons are produced than removed, the system is said to be *supercritical*, and *subcritical* inversely.

If we consider the time-independent form of Eqn 2.3 with, for example, void boundary conditions as given above, *i.e.*,

$$\mathcal{L}\psi(\boldsymbol{r}, E, \boldsymbol{\Omega}) = \mathcal{S}\psi(\boldsymbol{r}, E, \boldsymbol{\Omega}) + \mathcal{F}\phi(\boldsymbol{r}, E) + Q_{\text{ext}}(\boldsymbol{r}, E, \boldsymbol{\Omega}) \; , \tag{2.5}$$

this can be difficult to solve as, mathematically, this only yields non-negative solutions when the system is critical. However, this equilibrium state is not known *a priori* when carrying out calculations, and can often only be attained with a fine tuning of the parameters of the simulated reactor or domain.

Hence, the equality between both sides of the equation is artificially conserved by introducing an eigenvalue, the *effective multiplication factor*, k_{eff}.[2] This eigenvalue adjusts the average number, ν, of neutrons produced per fission, so that the steady-state BTE is given by,

$$\mathcal{L}\psi(\boldsymbol{r}, E, \boldsymbol{\Omega}) = \mathcal{S}\psi(\boldsymbol{r}, E, \boldsymbol{\Omega}) + \frac{1}{k_{\text{eff}}}\mathcal{F}\phi(\boldsymbol{r}, E) + Q_{\text{ext}}(\boldsymbol{r}, E, \boldsymbol{\Omega}) \; . \tag{2.6}$$

---

[2]Another method of balancing the steady state equation is through the α eigenvalue method. The k eigenvalue method has grown in preference with the community over the years because of its advantages such as its ease of numerical implementation and physical interpretation amongst others. Please refer to [23] for more details.

The k_{eff} governs the *criticality* of the reactor, *i.e.*, whether the nuclear chain reaction is self-sustaining or not. In this way, information is obtained about the state of the reactor and which adjustments have to be made to achieve a critical reactor.

- $k_{\text{eff}} < 1$ – more neutrons are consumed than produced, hence the reactor is *subcritical*, and the flux decays exponentially.

- $k_{\text{eff}} = 1$ – the neutron population holds steady, the reactor is said to be *critical*, and the reaction is self-sustaining.

- $k_{\text{eff}} > 1$ – more neutrons are produced than consumed, resulting in a *supercritical* reactor.

2.1.2   Overview of numerical resolution

Analytical resolution of the transport equation is generally possible only for extremely simple scenarios. This is obviously not the case for a nuclear reactor. The geometries involved are exceedingly complex, as are the material compositions. For this reason, one has to model the problem numerically.

There are two main classes of methods for solving the transport equation: Monte Carlo and deterministic methods. A brief overview of each is given below. But, as this work focuses on a subset of deterministic methods, they are discussed in more detail thereafter, in Sec. 2.2.

The Monte Carlo method   Also known as the stochastic method, the Monte Carlo method involves the simulation of a colossal number of particles and uses the generated data to obtain a statistical representation of the problem.

Here, solving the partial differential equations are not required per se since the underlying physics of the problem is represented by Probability Density Functions (PDFs) defined using a *continuous* energy representation of the cross sections. Random sampling (using a random number generator) from these PDFs yield starting conditions for a neutron which is then followed until its 'death' by capture (absorption or fission) or leakage. The interactions of the neutron throughout its life are simulated according to the geometry and composition of the system, as well as the collision laws. All the interactions of the neutron are recorded. This process is repeated for an enormous number of neutrons, leading to a fairly representative view of reality. It is the best estimate model using the input parameters for that problem. Statistical averages of the desired observations (also known as *scores*), with a calculated variance, can then be obtained from the data collected. [2, 22, 24]

However, because of its need to simulate the 'lives' of a huge number of neutrons so as to have an accurate picture, the Monte Carlo method is also usually characterised by very long computational times. For this reason, in the nuclear industry, it is mostly used as a validation and benchmarking tool for deterministic methods. It is also used in cases where the problem is too difficult to model deterministically, such as in very low-flux regions far from the core, for example, to calculate the neutron fluence at the wall of a reactor pressure vessel.

Deterministic methods Deterministic methods are based on a discretisation of the phase space variables: space $\boldsymbol{r}$ ($= \{x, y, z\}$), energy E, and direction $\boldsymbol{\Omega}$ ($= \{\theta, \varphi\}$). This leads to a system of coupled equations in matrix form with matricial operators which can then be solved numerically through efficient algorithms. As such, they are much faster, which is why it is mostly used in industry for production calculations. However, they are also not as exact since discretising inherently implies approximating.

2.2 Deterministic Resolution of the Transport Equation

This section will describe each step of the deterministic resolution of the transport equation, in a top-down approach. This will include the different discretisation methods.

2.2.1 k_{eff} and external iterations

Solving for the effective multiplication factor, k_{eff}, is carried out via the *power iteration method*. These iterations are said to be *external* as they are over the outer level of the equation, as opposed to the *inner* iterations that are over the flux. This will be clearer when summarising in Sec. 2.2.6.

If we consider the source-free time-independent BTE, *i.e.*,

$$\mathcal{L}\psi(\boldsymbol{r}, E, \boldsymbol{\Omega}) = \mathcal{S}\psi(\boldsymbol{r}, E, \boldsymbol{\Omega}) + \frac{1}{k_{\text{eff}}}\mathcal{F}\phi(\boldsymbol{r}, E) \ . \tag{2.7}$$

and have starting guesses for the k_{eff} and scalar flux, $\phi(\boldsymbol{r}, E)$, the iterative scheme can be written as

$$\mathcal{L}\psi^{(p+1)}(\boldsymbol{r}, E, \boldsymbol{\Omega}) = \mathcal{S}\psi^{(p+1)}(\boldsymbol{r}, E, \boldsymbol{\Omega}) + \frac{1}{k_{\text{eff}}^{(p)}}\mathcal{F}\phi^{(p)}(\boldsymbol{r}, E) \ , \tag{2.8}$$

where (p) indicates the power iteration number. The value for the next iterate $\phi(\boldsymbol{r}, E)$ is computed by inverting the streaming-and-collision and scattering operators, and integrating

over the unit sphere, as such:

$$\phi^{(p+1)}(\boldsymbol{r}, E) = \int_{4\pi} d^2\Omega \ \psi^{(p+1)}(\boldsymbol{r}, E, \boldsymbol{\Omega})$$
$$= \frac{1}{k_{\text{eff}}^{(p)}} \int_{4\pi} d^2\Omega \ (\mathcal{L} - \mathcal{S})^{-1} \left(\mathcal{F}\phi^{(p)}(\boldsymbol{r}, E) \right) . \tag{2.9}$$

Integrating Eqns. 2.7 and 2.8 over the complete phase space domain, we get respectively

$$\int \int \int_{r,E,\boldsymbol{\Omega}} d\boldsymbol{r} \ dE \ d\boldsymbol{\Omega} \left((\mathcal{L} - \mathcal{S})\psi^{(p+1)}(\boldsymbol{r}, E, \boldsymbol{\Omega}) \right) = \frac{1}{k_{\text{eff}}^{(p+1)}} \int \int \int_{r,E,\boldsymbol{\Omega}} d\boldsymbol{r} \ dE \ d\boldsymbol{\Omega} \ \mathcal{F}\phi^{(p+1)}(\boldsymbol{r}, E) ,$$

and

$$\int \int \int_{r,E,\boldsymbol{\Omega}} d\boldsymbol{r} \ dE \ d\boldsymbol{\Omega} \left((\mathcal{L} - \mathcal{S})\psi^{(p+1)}(\boldsymbol{r}, E, \boldsymbol{\Omega}) \right) = \frac{1}{k_{\text{eff}}^{(p)}} \int \int \int_{r,E,\boldsymbol{\Omega}} d\boldsymbol{r} \ dE \ d\boldsymbol{\Omega} \ \mathcal{F}\phi^{(p)}(\boldsymbol{r}, E) .$$

Equating the right sides of the two equations and rearranging, we are able to find

$$k_{\text{eff}}^{(p+1)} = k_{\text{eff}}^{(p)} \frac{\int \int \int_{r,E,\boldsymbol{\Omega}} d\boldsymbol{r} \ dE \ d\boldsymbol{\Omega} \ \mathcal{F}\phi^{(p+1)}(\boldsymbol{r}, E)}{\int \int \int_{r,E,\boldsymbol{\Omega}} d\boldsymbol{r} \ dE \ d\boldsymbol{\Omega} \ \mathcal{F}\phi^{(p)}(\boldsymbol{r}, E)} , \tag{2.10}$$

which represents the overall external iteration scheme.

2.2.2 Energy discretisation

Energy discretisation, through the *multigroup formalism*, involves dividing the energy domain into a number, G, of groups. This results into a system of G equations which are then coupled through the source terms. For brevity's sake and because it is not the focus of this work, the multigroup formalism will not be detailed. Only the result is stated below (see [22] for more details), with the group-wise transport equation given by

$$\boldsymbol{\Omega} \cdot \nabla \psi^g(\boldsymbol{r}, \boldsymbol{\Omega}) + \Sigma^g(\boldsymbol{r})\psi^g(\boldsymbol{r}, \boldsymbol{\Omega}) = Q^g(\boldsymbol{r}, \boldsymbol{\Omega}) , \ g = [1, G] , \tag{2.11}$$

with

$$Q^g(\boldsymbol{r}, \boldsymbol{\Omega}) = \sum_{g'=1}^{G} \int_{4\pi} d^2\Omega' \ \Sigma_s^{g \leftarrow g'}(\boldsymbol{r}, \boldsymbol{\Omega} \leftarrow \boldsymbol{\Omega}') \ \psi^{g'}(\boldsymbol{r}, \boldsymbol{\Omega}') +$$
$$\frac{1}{4\pi k_{\text{eff}}} \sum_i \chi_i^g \sum_{g'=1}^{G} \nu \Sigma_{f,i}^{g'}(\boldsymbol{r})\phi^{g'}(\boldsymbol{r}) , \tag{2.12}$$

where the summation over i represents the sum of various fissile isotopes. Also, it should be noted that the cross-sections are assumed to be piecewise constant over each energy group.

This system of equations is solved using an iterative scheme as well. In the high-energy groups, because the neutrons are only losing energy (*down-scattering*), the scattering source is known. As there is no coupling between the groups, each group can be solved for individually. However, in the thermal region of energy, there is the possibility of neutrons gaining in energy. In this case, couplings arise between the implicated energy groups and they have to be solved for simultaneously using something like a Gauss-Seidel scheme. More details can be found in the texts [2], [22], and [24], and the references therein.

For the rest of the document, the g index will be omitted but implied, in an effort to alleviate the notation.

2.2.3 Anisotropic scattering source density

In order to handle anisotropic scattering, the scattering cross section is usually expanded in terms of Legendre polynomials. Most domains are more or less homogeneous such that the collision incident angle is usually negligible and $\Sigma_s(\boldsymbol{r}, \boldsymbol{\Omega} \leftarrow \boldsymbol{\Omega}')$ instead depends on the outgoing scattering angle. Using the addition theorem of spherical harmonics [22], the end result for the scattering source term is given as,

$$Q(\boldsymbol{r}, \boldsymbol{\Omega}) = \sum_{l=0}^{\infty} \frac{2l+1}{4\pi} \Sigma_{s,l}(\boldsymbol{r}) \sum_{m=-l}^{l} R_l^m(\boldsymbol{\Omega}) \, \phi_l^m(\boldsymbol{r}) \tag{2.13}$$

where $R_l^m(\boldsymbol{\Omega})$ are the real spherical harmonics, and the summation to infinity represents the exact solution; however in calculations, this is usually truncated to some order L. $\phi_l^m(\boldsymbol{r})$ are the flux moments defined by

$$\phi_l^m(\boldsymbol{r}) = \int_{4\pi} d^2\Omega' \, R_l^m(\boldsymbol{\Omega}')\psi(\boldsymbol{r}, \boldsymbol{\Omega}') \ . \tag{2.14}$$

2.2.4 Angular discretisation: the P_n, SP_n and S_N methods

The two main types of angular discretisation are the discrete ordinates, S_N, and spherical harmonics, P_n, methods. Another method that is derived somewhat heuristically from the P_n method is the SP_n method. While the diffusion equation is not an angular discretisation – it is instead used in full-core calculations due to its low computational cost – a brief outline is presented in this subsection due to its relationship to the P_n method.

The P_n method

The P_n method consists of expanding the flux in terms of the real spherical harmonics, $R_l^m(\boldsymbol{\Omega})$, up to a certain order, n. [25, 26] These spherical harmonics form an orthogonal set of basis functions and allows for the representation of the flux over the surface of a unit sphere. Again, an infinite expansion would lead to an exact representation of the flux but this is of course not realistic feasible. This leads to the angular flux being expressed as

$$\psi(\boldsymbol{r}, \boldsymbol{\Omega}) = \sum_{l=0}^{n} \frac{2l+1}{4\pi} \sum_{m=-l}^{l} \phi_l^m(\boldsymbol{r}) R_l^m(\boldsymbol{\Omega}) \ , \tag{2.15}$$

where n is odd and is the term to which the expansion is truncated.

Similarly to the preceding section, $\phi_l^m(\boldsymbol{r})$ is given by

$$\phi_l^m(\boldsymbol{r}) = \int_{4\pi} d^2\Omega R_l^m(\boldsymbol{\Omega}) \psi(\boldsymbol{r}, \boldsymbol{\Omega}) \tag{2.16}$$

and the within-group scattering source by

$$\begin{aligned}
Q(\boldsymbol{r}, \boldsymbol{\Omega}) &= \sum_{l=0}^{L} \frac{2l+1}{4\pi} \Sigma_{s,l}(\boldsymbol{r}) \sum_{m=-l}^{l} R_l^m(\boldsymbol{\Omega}) \ \phi_l^m(\boldsymbol{r}) \\
&= \sum_{l=0}^{L} \frac{2l+1}{4\pi} \sum_{m=-l}^{l} R_l^m(\boldsymbol{\Omega}) \ Q_l^m(\boldsymbol{r}) \ ,
\end{aligned} \tag{2.17}$$

where $L \le n$.

The P_n equations can then be derived by multiplying the BTE by the real spherical harmonics and integrating over the unit sphere. This leads to a system of $(n+1)^2$ coupled equations for each energy group, which implies a quadratic increase in the number of unknowns as the order is increased. Hence, beyond the first few orders, this method is somewhat less desirable in 3D (and to some extent, 2D).

That said, in 1D slab geometry, the equations are much simpler since there is only one angular variable, μ. Following the same method just outlined but where the real spherical harmonics reduce to Legendre polynomials and the unit sphere is instead the domain $-1 \le \mu \le 1$, the resulting $(n+1)$ P_n equations can be written as [22]

$$\frac{l}{2l+1}\frac{d}{dx}\phi_{l-1}(x) + \frac{l+1}{2l+1}\frac{d}{dx}\phi_{l+1}(x) + \Sigma(x)\phi_l(x) = Q_l(x) \ , \tag{2.18}$$

where $0 \le l \le n$.

The spherical harmonics method does a good job for periodic regular solutions but struggles with discontinuous solutions. For reactor physics applications, this means it is efficient for optically thick diffusive media, but inefficient conversely or in highly anisotropic media. A potential disadvantage is that for high P_n orders, it might be necessary to store more angular moments than needed by the scattering source. This leads to increased memory requirements. Also, because the streaming term is not diagonal, as the order increases, this leads to a denser matrix which means a heavier computational requirement.

P_1 **approximation and diffusion equation** One way of deriving the diffusion equation is by considering the first two equations of 2.18, *i.e.*, the P_1 equations. Assuming the flux, as well as the scattering source, is a linear function of angle, we get,

$$\frac{d}{dx}\phi_1 + \Sigma(x)\phi_0 = \Sigma_{s0}\phi_0$$
$$\frac{d}{dx}\left(\frac{1}{3}\phi_0\right) + \Sigma(x)\phi_1 = \Sigma_{s1}\phi_1 \ . \tag{2.19}$$

From the second equation,

$$\phi_1 = -\left[\frac{1}{3(\Sigma - \Sigma_{s1})}\right]\frac{d}{dx}\phi_0 = -D\frac{d}{dx}\phi_0 \ , \tag{2.20}$$

where D is defined as the *diffusion coefficient*. Substituting for ϕ_1 in the first equation then yields the 1D diffusion equation as

$$-\frac{d}{dx}D\frac{d}{dx}\phi_0 + \Sigma\phi_0 = \Sigma_{s0}\phi_0 \ . \tag{2.21}$$

Eqn. 2.21 is the legacy diffusion equation. However, production calculations do not rely on Eqn. 2.20 for the definition of the diffusion coefficient [22].

The SP_n method

A common misconception is that the SP_n method is also based on a spherical harmonics expansion. This is not entirely correct. It is instead based on the 1D P_n equations (given by Eqn. 2.18) which are then heuristically assumed to be correct in three dimensions. It was originally presented by Gelbard [27] whose approach was based on the following methodology

1. for even l, replace $\frac{d}{dx}$ with the divergence operator, and

2. for odd l, replace $\frac{d}{dx}$ with the gradient operator.

This yields, for $0 \leq l \leq n$,

$$\frac{l}{2l+1}\boldsymbol{\nabla} \cdot \Phi_{l-1}(\boldsymbol{r}) + \frac{l+1}{2l+1}\boldsymbol{\nabla} \cdot \Phi_{l+1}(\boldsymbol{r}) + \Sigma(r)\phi_l(\boldsymbol{r}) = Q_l(\boldsymbol{r}) \ , \ l \text{ even}$$
$$\frac{l}{2l+1}\boldsymbol{\nabla}\phi_{l-1}(\boldsymbol{r}) + \frac{l+1}{2l+1}\boldsymbol{\nabla}\phi_{l+1}(\boldsymbol{r}) + \Sigma(r)\Phi_l(\boldsymbol{r}) = \boldsymbol{Q}_l(\boldsymbol{r}) \ , \ l \text{ odd}.$$

(2.22)

While this was initially a heuristic approach, there has since been asymptotic and variational derivations of these equations. We especially find that the asymptotic presentation by Pomraning [28] is easy to understand and makes the most intuitive sense. It demonstrates the equations by considering a domain where a local planar symmetry can be asymptotically assumed within the context of a slowly-varying flux. This allows for the angular variable, $\boldsymbol{\Omega}$, to be simplified by projection on the rotational axis, leading to a 1D approximation.

This addresses in some part the issue of the square increase in the number of variables with regular P_n, while at the same time resulting in a better approximation than a 3D P_1 or straightforward 1D calculation. However, it has been observed that increasing the order does not significantly improve the solution, which is why low orders are used. SP_n is also generally used for full core calculations (similar to the diffusion method).

It should be noted that, in the linear anisotropic approximation of the flux and scattering source, the diffusion equation, the P_1 equations and SP_1 are all equivalent.

The discrete ordinates (S_N) method

The discrete ordinates method – also called the S_N method – is a collocation method which was first used for radiative transfer. It was implemented by Chandrasekhar [29] in the early 1940's and then adapted and applied to reactor physics problems by Carlson [30] in the early to mid 1950s.

The S_N method uses quadrature rules to recast the integral over angle to a summation. Therefore, when considering a three-dimensional domain, this essentially discretises the continuous angular phase space of the unit sphere into a discrete number of permitted directions of travel for the neutrons over that sphere. This integral over angle is performed for the angular flux, $\psi(\boldsymbol{r}, \boldsymbol{\Omega})$ of the transport equation so as to obtain the scalar (or integrated) flux, $\phi(\boldsymbol{r})$ at each point in space.

In 3D, over one octant of a considered unit sphere at a point in space, this is given by

$$\frac{2}{\pi}\int_0^1 d\mu \int_0^{\pi/2} d\varphi f(\mu, \eta, \xi) \approx \sum_{n=1}^{M} \omega_n f(\mu_n, \eta_n, \xi_n),$$

(2.23)

where $\eta = \sqrt{1-\mu^2}\cos\varphi$, $\xi = \sqrt{1-\mu^2}\sin\varphi$, M ($= N(N+2)/8$)[3] is the number of directions for one octant and N is known as the *order* of the method. The sum of all the weights is 4π, and each is given by

$$\omega_n = \int_{\mathbf{\Omega}_n} d^2\Omega \ . \tag{2.24}$$

In 1D, this quadrature rule is written

$$\int_{-1}^{1} d\mu f(\mu) \approx \sum_{n=1}^{M} \omega_n f(\mu_n), \tag{2.25}$$

where ω_n are the weights, μ_n the base points, and $M = N(N+2)/4$ for the whole angular domain.

Several types of quadratures have been used with the discrete ordinates method. Initially, quadratures were based on Gauss mechanical quadratures, *i.e.*, on the zeroes of the Legendre polynomials of the first kind and the associated weights - these are the Gauss-Legendre quadratures.

The *product quadrature* is somewhat based on this too. It relies on using one type of quadrature in one dimension, and another for the two remaining dimensions. For example, a Gauss-Legendre quadrature (on the polar axis) can be used together with a Gauss-Chebyshev quadrature (on the azimuthal plane). An example is given in Fig. 2.1a.

Perhaps the most popular and the one used throughout this work, the *Level Symmetric (LS)* quadrature (see Fig. 2.1b), was developed in 1961 by Carlson and Lee [31]. These have the advantage of being rotationally invariant in $\pi/2$ increments. Hence, the base points on one octant can be duplicated to the others. These are useful in domains where there is no preferential direction of travel and hence, results in a better representation of the flux. Without this, undesirable bias could be introduced in the computation by the labelling of the axes. [31] The main drawback here is the limitation in the number of discrete directions: beyond a certain point, physically unrealistic weights appear, and hinder the resolution.

Various types of quadratures have been developed, and a review of them all is beyond the scope of this work. However, one point of note is that there is no intrinsic LS quadrature for a three-dimensional hexagonal geometry. Indeed, it would not be possible to reconcile the planar three-fold ($\pi/3$) azimuthal symmetry with the polar $\pi/2$ symmetry. That being said, it is entirely possible to use the usual LS even with hexagonal geometry. With a sufficient number of directions, the biases mentioned earlier can be mitigated.

Applying the quadrature rule to the compact-form transport equation given by Eqn. 2.11,

---

[3] This is only strictly true for the Level Symmetric (LS) quadrature and might not be for others.

(a) Product quadrature.　　　　　　　　　(b) LS quadrature.
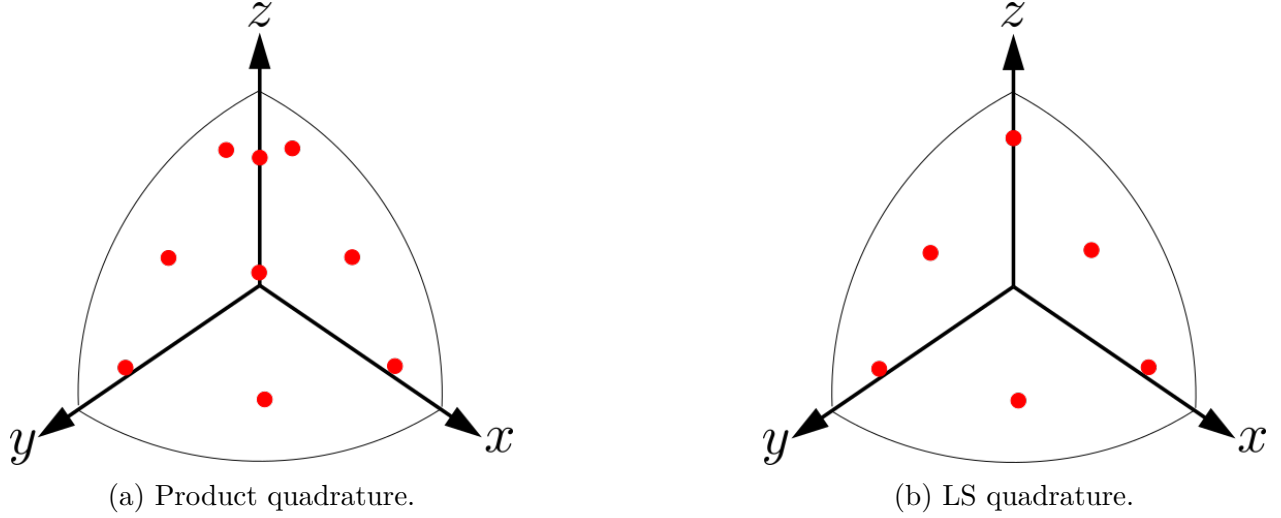
Figure 2.1 Octant showing the quadrature points for the S_6 order, illustrating the differences between the LS and product quadratures.

we obtain the decoupled set of equations for each direction given by,

$$\boldsymbol{\Omega}_n \cdot \nabla \psi_n(\boldsymbol{r}) + \Sigma(\boldsymbol{r})\psi_n(\boldsymbol{r}) = Q_n(\boldsymbol{r}) \ , \tag{2.26}$$

where $1 \leq n \leq M$ and $M = N(N+2)^3$ for the whole unit sphere, $\psi_n(\boldsymbol{r}) = \psi(\boldsymbol{r}, \boldsymbol{\Omega}_n)$ and $Q_n(\boldsymbol{r}) = Q(\boldsymbol{r}, \boldsymbol{\Omega}_n)$ for notation simplicity, and where $\boldsymbol{\Omega}_n$ denotes the permitted direction of travel by the quadrature rule. LS quadratures will be assumed for the rest of the document unless otherwise specified. The spherical harmonics moments of the flux, equivalent to Eqn. 2.14 without the multigroup discretisation, then becomes

$$\phi_l^m(\boldsymbol{r}) = \sum_{n=1}^{M} \omega_n R_l^m(\boldsymbol{\Omega}_n)\psi_n(\boldsymbol{r}) \ . \tag{2.27}$$

Solving Eqn. 2.26 for each direction and then finding the integrated flux using Eqn. 2.27 allows for the resolution of the BTE.

Compared to the P_n method, S_N is better able to model highly anisotropic or optically thin media, without needing to substantially increase the order. In turn, this leads to decreased memory requirements. Also, owing to this decoupling of the directions of neutron travel, the streaming term is diagonal which leads to a more computationally efficient resolution. The directions are only dependent on each other for the calculation of the scalar flux, which is necessary for the source calculation. As it will be seen in later sections, this is one of the parallelisation solutions employed for increased performance.

However, it also does come with its fair share of disadvantages. The flip side of not being dependent on the entire angular phase space is that it is inefficient in optically thick (*i.e.*, diffusive) media. Also, in domains with point sources and low diffusivity, *ray effects* can appear. This is the result of high angular discontinuities between the permitted directions of travel. It is more prominent at low orders, and can be mitigated by increasing the order or using angular smearing techniques.

2.2.5 Space-angle sweep operation and Source Iteration (SI)

At this point, the next step in the discretisation process is about dealing with the spatial variable/domain. However, because part of the focus of this work was on the implementation of a new – at least as far as DRAGON5 is concerned – spatial discretisation method, we will leave the full discussion of this part to the next chapter.

For now, suffice it to say that the domain is usually meshed into a number of elements and the discretised transport equation is solved over each mesh element. Depending on the discretisation schemes, the resolution might be global, *i.e.*, over the whole domain at once, or local, *i.e.*, over each element at a time. For reasons that will be clearer in Chap. 3, the latter method is preferred. In that case, there are different ways that the elements in the domain can be traversed, a couple of which are shown in Fig. 2.2.

Thus, in this way, the domain can be swept for each direction, $\mathbf{\Omega}_n$, to obtain the solution in each mesh element. This is repeated for all directions in the quadrature set, and after applying the quadrature rule summation, the scalar flux is obtained. This is a single *space-angle sweep*. This also forms the basis of the *Source Iteration (SI)* method. An educated (or random) guess is made for the starting incoming angular flux which allows the space-angle sweep to start. Once the scalar flux is calculated, the scattering source can be recomputed. This is one source iteration, also called *inner iteration*, and is better illustrated below by considering the source-free scattering S_N equations,

$$
\mathbf{\Omega}_n \cdot \nabla \psi_n^{(\kappa+1)}(\mathbf{r}) + \Sigma(\mathbf{r})\psi_n^{(\kappa+1)}(\mathbf{r}) = \sum_{l=0}^{L} \frac{2l+1}{4\pi} \Sigma_{s,l}(\mathbf{r}) \sum_{m=-l}^{l} R_l^m(\mathbf{\Omega}_n) \ \phi_l^{m,(\kappa)}(\mathbf{r}) \ ,
$$
$$
\phi_l^{m,(\kappa+1)} = \sum_{n=1}^{M} \omega_n R_l^m(\mathbf{\Omega}_n)\psi_n^{\kappa+1}(\mathbf{r}) \ ,
$$
(2.28)

where κ is the source iteration index, which is not the same as the power iteration index, p, that we saw earlier. The equation does not change significantly if there are fission or external sources present, except for the added terms. This can also be expressed in operator notation,

(a) "Columnar" sweep where the domain is traversed vertically for the given direction. Cells are computed in a soley sequential manner.

(b) Wavefront-like sweep. Cells in a wavefront perpendicular to the direction of propagation could be computed at the same time.
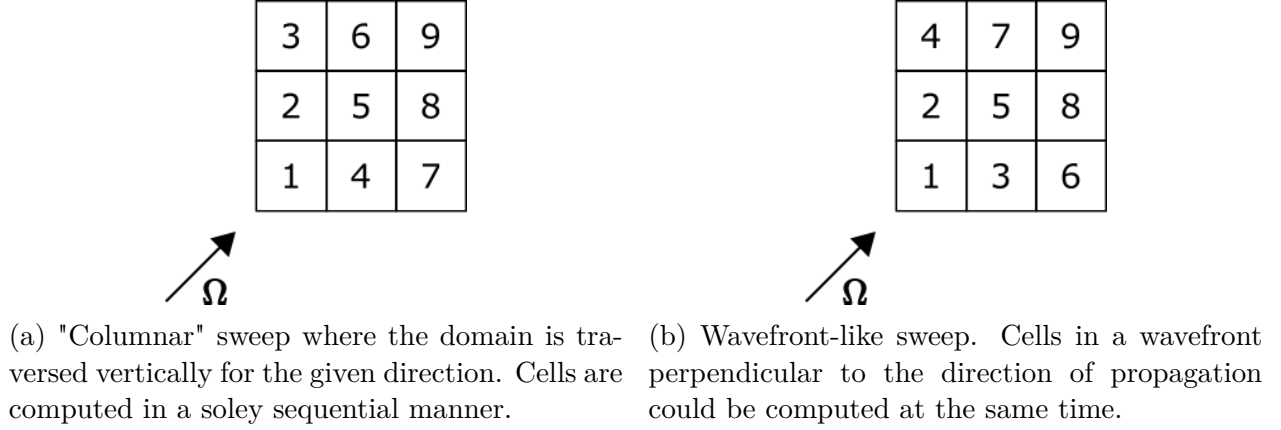
Figure 2.2 Two different types of sweep. The numbers correspond to the order in which the cells are computed in each case.

as done before,

$$
\begin{aligned}
\psi_n^{(\kappa+1)} &= \mathcal{L}^{-1}\mathcal{S}\phi^{(\kappa)} \; , \\
\phi^{(\kappa+1)} &= \mathcal{D}\psi_n^{(\kappa+1)} \; ,
\end{aligned}
\tag{2.29}
$$

where $\mathcal{D}$ is the quadrature operator, and it is understood that the $\mathcal{L}$ operator is inverted within the space-angle sweep framework.

2.2.6 Summary of resolution algorithm

As a means of perhaps more easily visualising the steps discussed thus far, the general resolution algorithm is summarised as a flow diagram in Fig. 2.3. This also makes it more obvious why the source and power iterations are respectively called the inner and outer iterations.

2.3 The DRAGON5 Code

The DRAGON5 code is an open-source reactor physics code developed at Polytechnique Montréal [18], with more than forty years of expertise and counting. There are already a lot of resources, including its data structure, user and developer manuals, detailing its many functions and capabilities amongst other things. However, since most of the development carried out in this dissertation were implemented there, we seek to present a brief overview of the code. This will help to understand the why or how of some algorithms implementations, design decisions or code limitations.

Start computation:
Initilise fluxes and
k_eff

Space-angle mesh
sweep computation

Update scattering
sources

Convergence?

No

Yes

Source Iteration

Power Iteration

Update scalar flux and
fission sources

Update k_eff

Convergence?

No

Yes

End computation
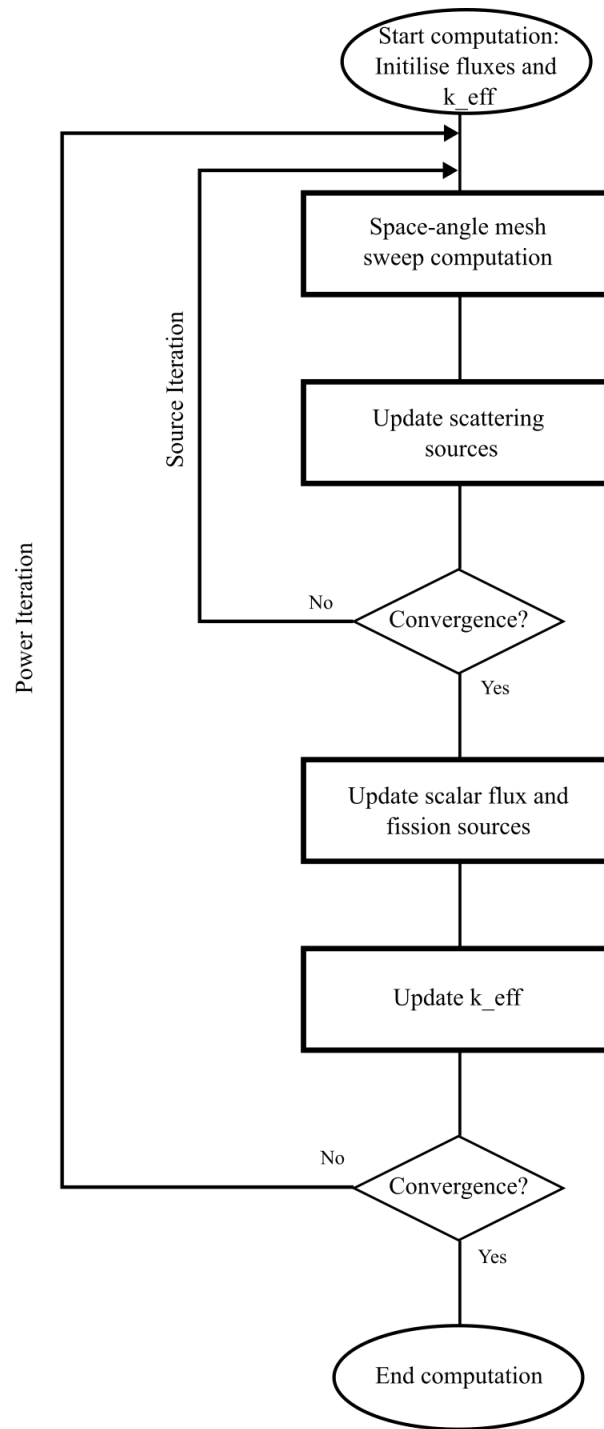
Figure 2.3 Simplified flowchart highlighting some of the most salient steps and loops required in the deterministic resolution of the Boltzmann Transport Equation (BTE). This is, by no means, an exhaustive diagram. It only highlights the points discussed in this dissertation.

2.3.1 General overview

DRAGON5 is actually made up of four main static libraries each having a somewhat different function. A concise description of each is given below.

- Utilib – contains some linear algebra functions, steam tables, plotting tools, pseudo-random number generators and some other basic mathematical functions.

- Ganlib – is the kernel of the program, insofar that it provides for the interface between the input files, the software and the output. This is where the information for the input language CLE-2000 is housed, as well as the structure for the custom data type, the *LCM object*. Through these, Ganlib is able to provide the modularity of the code. Interoperability for multiphysics applications with other software is also possible by ensuring that the data is in the correct format.

- Trivac – is mostly the finite element solver for the diffusion and SP_n equations.

- Dragon – is the core of the software. Amongst other things, it contains modules for defining the geometry, the *tracking* information for different types of discretisation, the material information (*i.e.*, cross-sections) and the flux resolution, including self-shielding, leakage and burnup calculations.

These are all mostly coded in Fortran and compiled using the Fortran2008 standard, except for the Ganlib kernel which is mostly in C. The compilation of the software is usually carried out in the order shown above because of the dependency of the libraries, although Utilib and Ganlib are independent.

An additional library, Donjon, encompasses everything related to full-core computations, including but not limited to supporting the definition of fuel channels, building extended material information objects, and simulating reactivity mechanisms and transient calculations. Consideration of this additional library gives the nuclear core code, DONJON5 – which still technically depends on all the above libraries to function.

2.3.2 Calculation overview and relevant modules

A simple calculation is usually carried on an input file written in CLE-2000. This scripting language was developed specifically for this purpose, to allow for relatively easy customisation of the test problem without the need for developer knowledge of the code. This input file will contain calls to the relevant modules, after which the output will be contained in an *LCM object* data type. This data type can be of two forms: associative tables or heterogeneous

lists. Associative tables have a directory-like structure, where each contained record or sub-directory is accessed using a name key. Heterogeneous lists are based on the same idea but accessed with integer keys.

The LCM object can either be stored in memory (termed a `LINKED_LIST` in this case) if it is to be used within the same calculation or saved to disk as `ASCII` or binary files for use later. Indeed, some modules require, as (an) input, LCM object(s) which were output by other modules. This flow of information is shown to some extent in Fig. 2.4. As one might expect, most calculations start with description of the geometry, handled by the aptly-named module `GEO:`.

The `MAC:` module (not portrayed on Fig. 2.4) reads a *macrolib* containing the material information (macroscopic cross-sections) directly. A macrolib can be generated from multiparameter databases in SAPHYB or MPO format [32] using interpolation modules `SCR:` or `MCR:`. A multiparameter database contains information for each isotope present in fuel channels in the reactor and processed by Donjon modules. The `MAC:` module ensures that the information is coherent with the geometry and formats the data accordingly.

The resulting output from `GEO:` is then needed by one of the tracking modules. Depending on the method of angular and/or spatial discretisation, the latter are responsible for the sub-meshing of the domain if needed and its subsequent numbering as well as the calculation of the relevant parameters such as lengths/areas/volumes of elements and/or integration lines and/or graph of operations and so on. The `SNT:` module was capable of handling 1D spherical, 1D/2D tube, 1D/2D/3D Cartesian geometries; additional 2D/3D hexagonal-geometry capabilities were added through this dissertation.

The `ASM:` module is used to build the collision probability matrices or other assembly matrices needed by some resolution methods such as the Raviart-Thomas spatial discretisation used in the SP_n or diffusion solver. For this reason, it is not very pertinent to the S_N resolution except in the case where a Synthetic Acceleration (SA) (please refer to Chap. 5) is requested by the user.

Finally, using the LCM objects from `MAC:`, the tracking module (here, `SNT:`), and `ASM:`, `FLU:` handles the flux resolution. This is taken care of by one of the 'sub-modules' (termed `CDOOR` in DRAGON5's development jargon, for 'computation door') depending on the type of tracking selected by the user. In our case, it is the sub-module[4] that contains the S_N solver. The bulk of the modifications and additions were done in the `FLU:` module.

---

[4]Note the use of the term 'sub-module'. Although not employed in DRAGON5's jargon, we adopt the term here somewhat liberally to highlight the fragmentation between each solver.
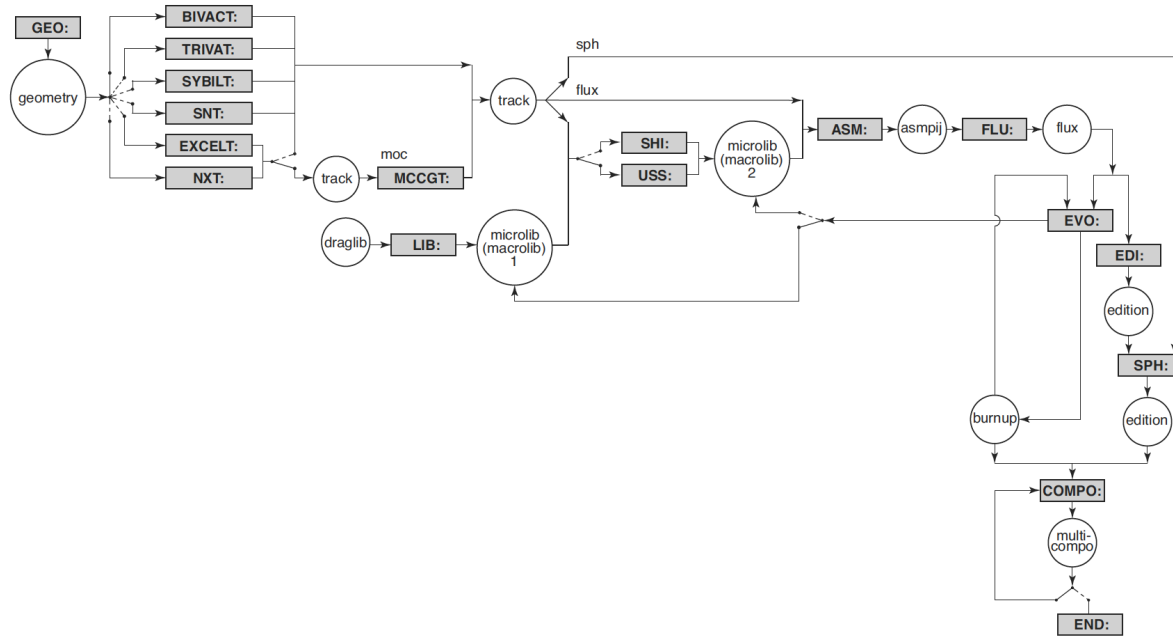
Figure 2.4 Diagram showing some of the core modules and functions of the DRAGON5 code. This is by no means exhaustive. For example, the module `MAC:` designed to read cross-section values directly into memory (*e.g.*, for simple benchmarking problems) is not portrayed. Also not shown are the 'sub-modules' encapsulating each of the solvers in the `FLU:` module, amongst which is the one containing the S_N solver. More information can be found in the user manual [33]. The modules are shown in rectangular shaded boxes and the module input/output are encircled. This was reproduced, with permission, from [22].

CHAPTER 3   SPATIAL DISCRETISATION METHODS

Several codes in the industry use a discontinuous finite element approximation as spatial discretisation, and there is a general consensus that because of its perceived stronger mathematical foundation, the Discontinuous Galerkin Finite Element Method (DGFEM) is intrinsically better. On top of that, as we have already talked about in Chap. 1, the S_N codes tested by Bay [15] for their study of Fast Neutron Reactors (FNRs) were all based on DGFEM. Hence, it was deemed important that the discontinuous finite element method be investigated and implemented within the DRAGON5 code for this project.

Also, as the High Order Diamond Difference (HODD) method was already available in the code, we endeavoured to try and discern the differences between the two spatial discretisations. That comparison work was done to some extent by Schunert [34], albeit on a Method of Manufactured Solutions (MMS) benchmark suite. In this project, we present results for numerical eigenvalue-problem benchmarks for each method, as it has been implemented within DRAGON5.

This chapter begins with a review of the HODD before discussing the finite element method more generally. We then derive the DGFEM equations and describe prior work in the field. The next section outlines its implementation in DRAGON5 by explaining the rationale behind the choice of solution space and giving an implementation example. A pseudo-code algorithm also helps to portray the functioning of the method within the domain sweep. Numerical results are then put forth for three benchmarks. These serve to both verify and validate the implementation of DGFEM and for comparison purposes. The two different spatial discretisation methods are analysed mostly in terms of the k_{eff}, the absorption rates and the computation times.

3.1   Review of Discretisation Methods

An in-depth review of all spatial discretisation methods applied to the transport equation, while interesting, would be well beyond the confines of this dissertation. Hence, the focus will really be on the diamond difference and finite element methods. Other methods will be referenced at times if only to provide context. For a thorough review, we highly recommend the works of Schunert [34] and Hébert [35], and the references therein as a starting point.

3.1.1 The Diamond Difference (DD) method

The High Order Diamond Difference (HODD) method, as its name suggests, is but a high-order polynomial expansion of the Diamond Difference (DD) method. Hence, that is the starting point of our discussion.

In this subsection, as this is a preliminary discussion, we will consider the one-dimensional form of the S_N transport equation (given in the preceding chapter by Eqn. 2.26), *i.e.*,

$$\mu_n \frac{\partial \psi_n(x)}{\partial x} + \Sigma(x)\psi_n(x) = Q_n(x) \ , 1 \leq n \leq M \ , \tag{3.1}$$

where $M = N(N+2)/4$. Now, let us consider a slab geometry domain $\mathcal{D}$ as shown in Fig. 3.1, with a meshing $\mathcal{M}_h$ into elements i such that $\bigcup_{i \in \mathcal{M}_h} i = \mathcal{D}$ and the cross-sections are constant per mesh element, *i.e.*, $\forall x \in i, \Sigma(x) = \Sigma_i$. Integrating Eqn. 3.1 over each mesh, we obtain

$$\frac{\mu_n}{\Delta x_i}(\psi_{n,i+1/2} - \psi_{n,i-1/2}) + \Sigma_i \psi_{n,i} = Q_{n,i} \ , \tag{3.2}$$

where the index over each element is made explicit, the $i \pm 1/2$ fluxes are the interface values and $\psi_{n,i}$ is the mesh-averaged flux, and Δx_i is the mesh width. Within the mesh-sweep paradigm discussed in Sec. 2.2.5, it can be assumed that one of the interface fluxes is known as an *incoming flux*.

However, we still have one equation with two unknowns. An *auxiliary*[1] equation is needed. This is usually obtained by assuming that the dependent variable has a given shape so that an interpolation equation can be constructed. The system of equation(s) can then be closed. The diamond difference scheme gives this equation as,

$$\psi_{n,i} = \frac{1}{2}(\psi_{n,i-1/2} + \psi_{n,i+1/2}) \ . \tag{3.3}$$
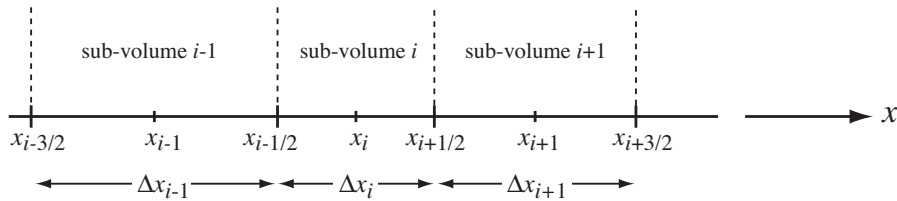
Figure 3.1 Domain subdivision for slab geometry.

---

[1]These are sometimes also called *complementary* or *closure* or *completeness* equations but all refer to more or less the same thing.

It is also interesting to note that while this method itself assumes an average single-value flux in the cell element (*i.e.*, $\psi_{n,i}$), the auxiliary equation represents a linear variation of flux within the cell.

3.1.2 The High Order Diamond Difference (HODD) method

The High Order Diamond Difference (HODD) method builds upon the DD scheme through a generalisation of the representation of the flux to high-order spatial polynomials. The idea being, as in finite element methods, that as the order is increased, there is a better representation of the flux. This was demonstrated for the discrete ordinates transport equation by Hébert [35] in 1D and 2D, and then further extended to 3D by Martin and Hébert [36]. We present in this subsection the method for the 2D case. This, in our opinion, alleviates the text of the extra notation associated with a 3D problem without over-simplifying the concepts. As shown by Schunert [34] and Jeffers [37], there can be a couple of ways to arrive at the equations governing the method but our development here is based on Hebert's method.

We begin with the 2D S_N transport equation,

$$\mu_n \frac{\partial \psi_n(x,y)}{\partial x} + \eta_n \frac{\partial \psi_n(x,y)}{\partial y} + \Sigma(x,y)\psi_n(x,y) = Q_n(x,y) \ , \tag{3.4}$$

where $1 \leq n \leq M = N(N+2)/2$. As in the previous subsection, the domain is meshed albeit this time with rectangular elements, each identified by its position $\{i,j\}$ with sides, Δx_i and Δy_j, and constant cross-section, $\Sigma_{i,j}$. We then apply the following change of variables,

$$\begin{aligned}
u &= \frac{1}{\Delta x_i}\left[x - \frac{1}{2}(x_{i-1/2} + x_{i+1/2})\right] \ , \\
v &= \frac{1}{\Delta y_j}\left[y - \frac{1}{2}(y_{j-1/2} + y_{j+1/2})\right] \ .
\end{aligned} \tag{3.5}$$

such that we are on the reference mesh element defined by $-1/2 \leq u \leq 1/2$ and $-1/2 \leq v \leq 1/2$. This is a common practice in spatial discretisation methods so as to have standardised equations that are scaled through mesh element constants. This gives the discretised equation on the reference element as

$$\frac{\mu_n}{\Delta x_i}\frac{\partial \psi_{n,i,j}(u,v)}{\partial u} + \frac{\eta_n}{\Delta y_j}\frac{\partial \psi_{n,i,j}(u,v)}{\partial v} + \Sigma_{i,j}\psi_{n,i,j}(u,v) = Q_{n,i,j}(u,v) \ . \tag{3.6}$$

It is now assumed that the flux (and source) can be developed using a polynomial series expansion. This series expansion is usually truncated to an arbitrary order, Λ. Any polynomial series could potentially be used. In the DRAGON5 S_N solver, the normalised Legendre

polynomials, $\widetilde{P}(u)_\alpha$ and $\widetilde{P}(v)_\beta$, were implemented, with $0 \leq \alpha \leq \Lambda$ and $0 \leq \beta \leq \Lambda$. The first four polynomials in the series are

$$
\begin{aligned}
\widetilde{P}_0(u) &= 1 \ , \\
\widetilde{P}_1(u) &= 2\sqrt{3}u \ , \\
\widetilde{P}_2(u) &= \sqrt{5}(6u^2 - 1/2) \ , \\
\widetilde{P}_3(u) &= \sqrt{7}(20u^3 - 3u) \ .
\end{aligned}
\tag{3.7}
$$

The flux and source are then expressed as

$$
\begin{aligned}
\psi_{n,i,j}(u,v) &= \sum_{\alpha=0}^{\Lambda} \sum_{\beta=0}^{\Lambda} \widetilde{P}_\alpha(u)\widetilde{P}_\beta(v) \ \psi_{n,i,j}^{[\alpha,\beta]} \ , \\
Q_{n,i,j}(u,v) &= \sum_{\alpha=0}^{\Lambda} \sum_{\beta=0}^{\Lambda} \widetilde{P}_\alpha(u)\widetilde{P}_\beta(v) \ Q_{n,i,j}^{[\alpha,\beta]} \ ,
\end{aligned}
\tag{3.8}
$$

where each expansion coefficient (also called Legendre moments here) is defined as

$$
\begin{aligned}
\psi_{n,i,j}^{[\alpha,\beta]} &= \int_{-1/2}^{1/2} du \widetilde{P}_\alpha(u) \ \int_{-1/2}^{1/2} dv \widetilde{P}_\beta(v) \ \psi_{n,i,j}(u,v) \ , \\
Q_{n,i,j}^{[\alpha,\beta]} &= \int_{-1/2}^{1/2} du \widetilde{P}_\alpha(u) \ \int_{-1/2}^{1/2} dv \widetilde{P}_\beta(v) \ Q_{n,i,j}(u,v) \ ,
\end{aligned}
\tag{3.9}
$$

where we have made use of the orthogonality of Legendre polynomials.

Multiplying Eqn. 3.6 by a series of test functions – in this case, the Legendre polynomials – and integrating over the domain gives,

$$
\frac{\mu_n}{\Delta x_i} \int_{-1/2}^{1/2} du \widetilde{P}_\alpha(u) \int_{-1/2}^{1/2} dv \widetilde{P}_\beta(v) \frac{\partial \psi_{n,i,j}(u,v)}{\partial u} + \frac{\eta_n}{\Delta y_j} \int_{-1/2}^{1/2} du \widetilde{P}_\alpha(u) \int_{-1/2}^{1/2} dv \widetilde{P}_\beta(v) \frac{\partial \psi_{n,i,j}(u,v)}{\partial v} +
$$
$$
\Sigma_{i,j} \int_{-1/2}^{1/2} du \widetilde{P}_\alpha(u) \int_{-1/2}^{1/2} dv \widetilde{P}_\beta(v) \psi_{n,i,j}(u,v) = \int_{-1/2}^{1/2} du \widetilde{P}_\alpha(u) \int_{-1/2}^{1/2} dv \widetilde{P}_\beta(v) Q_{n,i,j}(u,v) \ .
\tag{3.10}
$$

In DRAGON5, HODD was already implemented for solution expansions from the zeroth/flat order up to parabolic, *i.e.*, $0 \leq \Lambda \leq 3$ for the 1D/2D/3D Cartesian geometries. We added hexagonal geometry capability in this work (see Chap. 4). Also, as variable-order expansion has not been implemented (yet), the same order is used for expansion for every dimension.

As an example, for a value of Λ of 1, after considerable algebra, the first four Legendre

moments of the flux are given by the following equations,

$$\frac{\mu_n}{\Delta x_i}(\psi_{n,i+1/2,j}^{[*,0]} - \psi_{n,i-1/2,j}^{[*,0]}) + \frac{\eta_n}{\Delta y_j}(\psi_{n,i,j+1/2}^{[0,*]} - \psi_{n,i,j-1/2}^{[0,*]}) + \Sigma_{i,j}\psi_{n,i,j}^{[0,0]} = Q_{n,i,j}^{[0,0]}$$

$$\sqrt{3}\frac{\mu_n}{\Delta x_i}(\psi_{n,i-1/2,j}^{[*,0]} + \psi_{n,i+1/2,j}^{[*,0]} - 2\psi_{n,i,j}^{[0,0]}) + \frac{\eta_n}{\Delta y_j}(\psi_{n,i,j+1/2}^{[1,*]} - \psi_{n,i,j-1/2}^{[1,*]}) +$$
$$\Sigma_{i,j}\psi_{n,i,j}^{[1,0]} = Q_{n,i,j}^{[1,0]}$$

$$\text{(3.11)}$$

$$\frac{\mu_n}{\Delta x_i}(\psi_{n,i+1/2,j}^{[*,1]} - \psi_{n,i-1/2,j}^{[*,1]}) + \sqrt{3}\frac{\eta_n}{\Delta y_j}(\psi_{n,i,j-1/2}^{[0,*]} + \psi_{n,i,j+1/2}^{[0,*]} - 2\psi_{n,i,j}^{[0,0]}) +$$
$$\Sigma_{i,j}\psi_{n,i,j}^{[0,1]} = Q_{n,i,j}^{[0,1]}$$

$$\sqrt{3}\frac{\mu_n}{\Delta x_i}(\psi_{n,i-1/2,j}^{[*,1]} + \psi_{n,i+1/2,j}^{[*,1]} - 2\psi_{n,i,j}^{[0,1]}) + \sqrt{3}\frac{\eta_n}{\Delta y_j}(\psi_{n,i,j-1/2}^{[1,*]} + \psi_{n,i,j+1/2}^{[1,*]} - 2\psi_{n,i,j}^{[1,0]}) +$$
$$\Sigma_{i,j}\psi_{n,i,j}^{[1,1]} = Q_{n,i,j}^{[1,1]} \ ,$$

where we have made use of the following definitions,

$$\psi_{n,i\pm1/2,j}^{[*,\beta]} = \int_{-1/2}^{1/2} dv \, \widetilde{P}_\beta(v) \ \psi_{n,i,j}(\pm1/2, v)$$

$$\psi_{n,i,j\pm1/2}^{[\alpha,*]} = \int_{-1/2}^{1/2} du \, \widetilde{P}_\alpha(u) \ \psi_{n,i,j}(u, \pm1/2) \ ,$$

$$\text{(3.12)}$$

which represent the Legendre moments of the boundary fluxes on the trace of the element domain (*i.e.*, the edges of the mesh element), which is itself the reduction of the flux defined on the whole domain. We also point out that lowest-order HODD (*i.e.*, HODD-0) simply amounts to the usual DD scheme, as expected.

However, we still have the same problem as before: while the obtained balance equations are exact, there are more unknowns than equations, even when accounting for the incoming fluxes being known. In fact, if we define ζ as the number of dimensions, there are $\left[(\Lambda+1)^\zeta + (\zeta-1)(\Lambda+1)^{(\zeta-1)}\right]$ unknowns and $(\Lambda+1)^\zeta$ equations. To solve this, Hébert [35] obtained the closure relations first in 1D by generalising the diamond difference scheme in such a way that the $(\Lambda+1)^{\text{th}}$ order equation had a trivial solution. By analogy, these were

then extrapolated to 2D, and are given as

$$\psi_{n,i\mp1/2,j}^{[*,\beta]} = \begin{cases} 2\psi_{n,i,j}^{[0,\beta]} - \psi_{n,i\pm1/2,j}^{[*,\beta]} & \text{if } \Lambda = 0, \\ \psi_{n,i\pm1/2,j}^{[*,\beta]} \mp 2\sqrt{3}\psi_{n,i,j}^{[1,\beta]} & \text{if } \Lambda = 1, \\ 2\psi_{n,i,j}^{[0,\beta]} + 2\sqrt{5}\psi_{n,i,j}^{[2,\beta]} - \psi_{n,i\pm1/2,j}^{[*,\beta]} & \text{if } \Lambda = 2, \\ \dots \end{cases} \qquad (3.13)$$

with the upper sign for $\mu_n < 0$, and lower sign for $\mu_n > 0$, as well as,

$$\psi_{n,i,j\mp1/2}^{[\alpha,*]} = \begin{cases} 2\psi_{n,i,j}^{[\alpha,0]} - \psi_{n,i,j\pm1/2}^{[\alpha,*]} & \text{if } \Lambda = 0, \\ \psi_{n,i,j\pm1/2}^{[\alpha,*]} \mp 2\sqrt{3}\psi_{n,i,j}^{[\alpha,1]} & \text{if } \Lambda = 1, \\ 2\psi_{n,i,j}^{[\alpha,0]} + 2\sqrt{5}\psi_{n,i,j}^{[\alpha,2]} - \psi_{n,i,j\pm1/2}^{[\alpha,*]} & \text{if } \Lambda = 2, \\ \dots \end{cases} \qquad (3.14)$$

with the upper sign for $\eta < 0$, and lower sign for $\eta > 0$

Substituting the unknown outgoing fluxes in the Legendre moments S_N equations (Eqns. 3.11) using the relevant closure relations, a system of equations is obtained allowing us to solve for the Legendre moments, *i.e.* the mesh-centred fluxes, $\psi_{n,i,j}^{[\alpha,\beta]}$. There are $(\Lambda+1)^\zeta$ degrees of freedoms (d.o.f.s) or unknowns, corresponding to these Legendre moments.

In this form, the High Order Diamond Difference (HODD) method has often been classified as a variant of the Finite Volume Method (FVM), insofar that this method decomposes the domain into homogeneous mesh elements before integrating the equation of conservation over each element. Using the product rule and divergence theorem (the equivalent of integration by parts in multiple dimensions), the volume (in 3D) integrals over the streaming terms are recast to surface integrals. The remaining volume and surface integrals can then be written in terms of volume- and surface-averaged expressions of the dependent variable. However, we shall see later that this is actually not correct (refer to section 3.1.5).

Nevertheless, the DD method has been analysed extensively. While it was originally expected to feature second-order accuracy [38] while retaining only the zeroth moment source, it has since been demonstrated that this does not hold in practice for realistic problems [39, 40], and is dependent on the smoothness of the problem at hand and the error norm used in computations. This result appears to have led to some decline in its popularity. Furthermore, it seems that this conclusion was assumed heuristically for HODD by part of the community.

Another aspect of HODD is that it enforces more coupling of the dependent variable in between mesh elements, when compared with discontinuous finite element methods. This can

lead to a lack of robustness of the method in optically thick regions and also lead to negative average angular (and potentially, scalar) flux values. These negative values can actually lead to instabilities sometimes when coupled with an acceleration method. For this reason, a quick and dirty fix usually is to check for negative values and set these to zero, although this is actually limited to the HODD-0 (*i.e.*, the classical DD scheme) approximation.

3.1.3   General overview of Finite Element Methods (FEMs)

Within the finite element method framework, the solution to a Partial Differential Equation (PDE) under consideration is approximated by a linear combination of polynomials – or more, generally, functions, that are taken from a finite-dimensional trial function space. Similar to the HODD method we just discussed (refer to Eqn. 3.8), the unknowns being solved for are the expansion coefficients of this linear combination of functions.

There are a few approaches to obtaining the FEM formulation. All of them, though, require an integral formulation whereby the PDEs are integrated over some domain of interest. One approach is the direct variational method, also called the *Rayleigh-Ritz* method. In this case, the equations are obtained from substitution of the trial functions into a functional equivalent to the PDE, and then finding the stationary points of that functional with respect to the expansion coefficients [41, 42].

Perhaps the more common approach though is the *weighted-residual* method whereby the residual of the approximate solution is taken to be orthogonal to a set of weighting (or test) functions with respect to some inner product. For a general PDE,

$$\Delta u(\boldsymbol{r}) = f(\boldsymbol{r}) \text{ , with } u(\boldsymbol{a}) = u_a, \ u(\boldsymbol{b}) = u_b \ , \tag{3.15}$$

defined over some domain $\mathcal{D}$ such that $\boldsymbol{r} \in \mathcal{D}$, let us consider the approximate solution, $\bar{u}$,

$$\bar{u}(\boldsymbol{r}) = \sum_{i=1}^{\Lambda} u_i v_i(\boldsymbol{r}) \tag{3.16}$$

where the trial functions, $v_i(\boldsymbol{r})$, are taken from a solution space P, subset of the Sobolev space, $L^2(\mathcal{D})$ such that $\bar{u} \in V = \{v \in P\}$. The residual, $\mathcal{R}$, is then given by

$$\mathcal{R} = \Delta \bar{u}(\boldsymbol{r}) - f(\boldsymbol{r}) \ . \tag{3.17}$$

Considering test functions, $w_j(\boldsymbol{r})$ $(j = [1, \Lambda])$ taken from another space, Q, with similar

properties, the weighted-residual method gives Λ equations represented by

$$\int_{\mathcal{D}} w_j \mathcal{R} d\boldsymbol{r} = 0 \ , j = [1, \Lambda] \ , \tag{3.18}$$

forming a system of equations allowing for the resolution of the expansion coefficients, u_i. If the solution space for the trial and test functions are the same, the method is sometimes referred to as the *Galerkin* or *Bubnov-Galerkin* (finite element) method.

The finite element scheme can be further subdivided into two different types: continuous and discontinuous. The main difference between the two is whether the approximate solution is continuous *globally* or not, *i.e.*, across the whole domain. In continuous FEM, the test and trial function spaces are supported across two or more adjacent cells of the domain mesh such that continuity is forced at the cell boundaries. This means that the solution boundary values are unique at the interface independent of the cell from which the trace is calculated. Because the boundary value is unique, this means that the solution is unique, and that the solution is pointwise continuous.

On the other hand, in the Discontinuous Finite Element Method (DFEM), the support for the test and trial functions is restricted *locally* to each cell element in the domain mesh. This entails that the flux is expanded in terms of piece-wise basis functions which are zero everywhere except on the element on which they are defined. This now means that the solution boundary values are *not* unique and will differ depending on which cell element the interface is approached from. Some form of 'continuity' is applied between cells only in a weak (integral) sense.

While this might not appear ground-breaking, the difference is massive when it comes to the resolution algorithm and its implementation. Continuous FEM leads to a globally coupled, albeit sparse, matrix with the end result of having to compute a global system of simultaneous equations. The computational requirements can be enormous, both in terms of memory and processing power.

However, with DFEM, the global domain-wise matrix features a block-like arrangement with very little coupling between each block. Indeed, this arises because of the local and uncoupled supports of the function spaces. Each block correspond to a cell element and is characterised by that cell and the test and trial functions. Due to this local property, the resolution can be done by a mesh sweep if the information (the aforementioned coupling) only transmits in one direction. This was illustrated on Fig. 2.2 in Sec. 2.2.5.

This allows each cell element to be considered individually. While this local matrix is fully dense, it is several orders of magnitude smaller (depending on the size of the problem, of

course). The size will depend solely on the expansion order of the function spaces – and the dimensionality of the problem at hand. Mathematically, the mesh sweep makes use of the fact that the global domain-wise matrix in this case is either lower or upper triangular. Treating one cell at a time is then somewhat akin to using forward or backward substitution, respectively.

In neutron transport, when using the discrete-ordinate method, the information only propagates downstream. For this reason, the DFEM is a perfect fit. As the neutron is only allowed to travel along certain directions and with these already known, the sweep can start upstream and then move downstream along with the allowed direction of neutron travel.

This method was actually first developed within the reactor physics community by Reed and Hill [43] at the Los Alamos National Laboratory (LANL) in 1973 and has since gained tremendous popularity. Numerous applications can be found in the literature although until recently, most applications were limited to solution spaces with linear polynomials. Some recent applications of high-order polynomial expansions for the solution space are the works of Wang and Ragusa [44, 45], Schunert *et al.* [46], Schunert [34], and Le Tellier [47].

In [44] and [45], Wang and Ragusa implemented high-order DFEM for two-dimensional unstructured triangular meshes using hierarchical basis sets. They analysed the numerical convergence of the method and were able to verify theoretical convergence: in the L_2 norm, the rate is given by at least $(p+1)$, with p being the spatial approximation order, depending on the regularity of the solution. Wang and Ragusa also showed, as expected, that quadratic, cubic or quartic polynomial expansions yielded successively more accurate results, albeit at the cost of more calculation resources and time. However, they found out that beyond cubic polynomial expansions, the gains were not worth the extra effort.

Moreover, a comparison work, carried out in [46] and [34], actually compared DFEM, HODD with a couple of other discretisation methods on a Method of Manufactured Solutions (MMS) benchmark suite. They showed that DFEM and HODD have about the same grind times, although as the order increased, the time for DFEM seemed to increase more than the others. That said, their results showed the discontinuous finite element scheme outperforming the DD and HODD methods in more or less every scenario, with *"HODD method for all orders fail[ing] dramatically for optically coarse meshes"*. While this does not bode well for our comparison, we will endeavour to see if we obtain similar results on our numerical benchmarks.

3.1.4 The Discontinuous Galerkin Finite Element Method (DGFEM)

We present in this section the Discontinuous Galerkin Finite Element Method (DGFEM) as applied to the (two-dimensional[2]) discrete-ordinate transport equation. This demonstration was heavily inspired by the works of Schunert [34] and Le Tellier [47].

We repeat once more the discrete-ordinate transport equation for each direction of neutron travel, $\mathbf{\Omega}_n$, and $\forall \boldsymbol{r} \in \mathcal{D}$,

$$\mathbf{\Omega}_n \cdot \nabla \psi_n(\boldsymbol{r}) + \Sigma(\boldsymbol{r})\psi_n(\boldsymbol{r}) = Q_n(\boldsymbol{r}) \ , \tag{3.19}$$

where $\mathcal{D}$ is the domain, defined in $\mathbb{R}^2$, with boundary Γ, and with boundary conditions, $\forall \boldsymbol{r} \in \Gamma_-$,

$$\psi_n(\boldsymbol{r}) = \psi_n^{BC}(\boldsymbol{r}) \ , \tag{3.20}$$

where Γ_- is the inflow flow boundary of $\mathcal{D}$, defined by $\Gamma_- = \{\boldsymbol{r} \in \Gamma : \mathbf{\Omega}_n \cdot \boldsymbol{n}(\boldsymbol{r}) < 0\}$, with $\boldsymbol{n}$ being the unit outward normal to Γ.

If we consider a meshing $\mathcal{M}_h$ of the domain $\mathcal{D}$ into elements ξ, each with boundary $\partial \xi$, such that $\bigcup_{\xi \in \mathcal{M}_h} \xi = \mathcal{D}$, the trial and test solution space, V_h^p, of discontinuous piece-wise polynomial functions, υ, is given by

$$V_h^p = \left\{ \upsilon \in L^2(\mathcal{D}) : \forall \xi \in \mathcal{M}_h, \upsilon|_\xi \in Q_p(\xi) \right\} \tag{3.21}$$

where $Q_p(\xi)$ is the space of polynomials up to degree p, supported on the element ξ.

Also, for two arbitrary neighbouring elements, ξ_1 and ξ_2, with shared boundary $b = \partial\xi_1 \cap \partial\xi_2$, functions from the solution space are double-valued such that

- for the trace taken within ξ_1, $\upsilon|_{b \cap \partial\xi_1}^+ = \upsilon|_{b \cap \partial\xi_2}^-$; and

- for the trace taken within ξ_2, $\upsilon|_{b \cap \partial\xi_2}^+ = \upsilon|_{b \cap \partial\xi_1}^-$;

where the $\pm$ superscripts indicates whether the restriction of the function is from the element or the neighbouring element respectively.

With this formalism in place, we can apply the method of weighted residuals: substituting in Eqn. 3.19 a solution, $\psi_{n,h} \in V_h^p$, multiplying the resulting equation by a function $w_h \in V_h^p$ and integrating by parts over each element, the local Discontinuous Galerkin (DG) formulation

---

[2]For consistency with the HODD demonstration.

is given as, in its *weak* or *variational* form,

$$\int_\xi \left(-\psi_{n,h}\left(\mathbf{\Omega}_n \cdot \nabla w_h\right) + \Sigma\psi_{n,h}w_h\right)ds + \int_{\partial\xi\backslash\Gamma_-} \left(\mathbf{n}\cdot\mathbf{\Omega}_n\right)\mathcal{F}^* w_h^+ dl =$$
$$\int_\xi Q_n w_h ds - \int_{\partial\xi\cap\Gamma_-}\left(\mathbf{n}\cdot\mathbf{\Omega}_n\right)\psi_n^{BC}w_h^+ dl \ , \tag{3.22}$$

where *ds* and *dl* indicate surface and line integrals respectively. $\mathcal{F}^*$ is the numerical flux on the boundaries. In neutron transport, the upwinding rule is used such that

$$\mathcal{F}^* = \begin{cases} \psi_{n,h}|^+ & \text{if } b\cap\partial\xi_+ \\ \psi_{n,h}|^- & \text{if } b\cap\partial\xi_- \ , \end{cases} \tag{3.23}$$

where $\partial\xi_+$ and $\partial\xi_-$ are respectively the outflow and inflow boundary of the element, ξ. Applying the above, we get

$$\int_\xi \left(-\psi_{n,h}\left(\mathbf{\Omega}_n\cdot\nabla w_h\right) + \Sigma\psi_{n,h}w_h\right)ds + \int_{\partial\xi_+}\left(\mathbf{n}\cdot\mathbf{\Omega}_n\right)\psi_{n,h}^+ w_h^+ dl +$$
$$\int_{\partial\xi_-\backslash\Gamma_-}\left(\mathbf{n}\cdot\mathbf{\Omega}_n\right)\psi_{n,h}^- w_h^+ dl = \int_\xi Q_n w_h ds - \int_{\partial\xi_-\cap\Gamma_-}\left(\mathbf{n}\cdot\mathbf{\Omega}_n\right)\psi_n^{BC}w_h^+ dl \ , \tag{3.24}$$

and after integrating by parts again, we get the *strong* form,

$$\int_\xi \left(\left(\mathbf{\Omega}_n\cdot\nabla\psi_{n,h}\right)w_h + \Sigma\psi_{n,h}w_h\right)ds - \int_{\partial\xi_-\backslash\Gamma_-}\left(\mathbf{n}\cdot\mathbf{\Omega}_n\right)\left(\psi_{n,h}^+ - \psi_{n,h}^-\right)w_h^+ dl -$$
$$\int_{\partial\xi_-\cap\Gamma_-}\left(\mathbf{n}\cdot\mathbf{\Omega}_n\right)\psi_{n,h}^+ w_h^+ dl = \int_\xi Q_n w_h ds - \int_{\partial\xi_-\cap\Gamma_-}\left(\mathbf{n}\cdot\mathbf{\Omega}_n\right)\psi_n^{BC}w_h^+ dl \ . \tag{3.25}$$

If we now consider that our element lies at inflow boundary, Γ_-, of the domain such that $\partial\xi_- \in \Gamma_-$, the DGFEM scheme is written as

$$\int_\xi \left(\left(\mathbf{\Omega}_n\cdot\nabla\psi_{n,h}\right)w_h + \Sigma\psi_{n,h}w_h\right)ds - \int_{\partial\xi_-}\left(\mathbf{n}\cdot\mathbf{\Omega}_n\right)\psi_{n,h}^+ w_h^+ dl =$$
$$\int_\xi Q_n w_h ds - \int_{\partial\xi_-}\left(\mathbf{n}\cdot\mathbf{\Omega}_n\right)\psi_{n,h}^- w_h^+ dl \ . \tag{3.26}$$

This can be more compactly summarised as

$$\int_\xi \left(\left(\mathbf{\Omega}_n\cdot\nabla\psi_{n,h}\right)w_h + \Sigma\psi_{n,h}w_h\right)ds - \int_{\partial\xi_-}\left(\mathbf{n}\cdot\mathbf{\Omega}_n\right)[\![\psi_{n,h}]\!]w_h^+ dl = \int_\xi Q_n w_h ds \ . \tag{3.27}$$

where $[\![\psi_{n,h}]\!] = \left(\psi_{n,h}^+ - \psi_{n,h}^-\right)$ was used as the *jump operator*.

3.1.5 High Order Diamond Difference (HODD) as Discontinuous Petrov-Galerkin Finite Element Method (DPGFEM)

As we mentioned earlier in Sec. 3.1.2, the HODD scheme is frequently seen as a variant of FVM. However, in their PhD dissertation, Schunert [34] goes on to demonstrate how it is actually a discontinuous *Petrov-Galerkin* finite element scheme. This is a finite element method whereby the set of trial functions is taken from a different space than that for the test functions. While the test functions are still given by the set of Legendre polynomials (refer to Eqn. 3.8), the flux is now given by trial functions of the form,

$$
\psi_{n,i,j}(u,v) = \sum_{\alpha=0}^{\Lambda} \sum_{\beta=0}^{\Lambda} \widetilde{P}_\alpha(u) \widetilde{P}_\beta(v)\ \psi_{n,i,j}^{[\alpha,\beta]} + \sum_{\beta=0}^{\Lambda} \widetilde{P}_{\Lambda+1}(u)\widetilde{P}_\beta(v)\ \psi_{n,i,j}^{[\Lambda+1,\beta]}
$$
$$
+ \sum_{\alpha=0}^{\Lambda} \widetilde{P}_\alpha(u)\widetilde{P}_{\Lambda+1}(v)\ \psi_{n,i,j}^{[\alpha,\Lambda+1]}\ .
$$
(3.28)

We find here, embedded in the trial functions, the $(\Lambda+1)^{\text{th}}$ order that Hébert uses to derive the closure relations. Formulated thusly, these auxiliary equations arise differently. The boundary fluxes on the exterior and interior traces of each element are taken to be distinct. Indeed, on the exterior incoming boundary, the flux is expanded only up to order Λ. The difference between that and the restriction of the flux taken on the interior trace is then required to be orthogonal to the test space with respect to an inner product. Hence, continuity is enforced on the boundary fluxes only in an integral sense and only up to order Λ. We present an outline of Schunert's proof in Appendix B but more detail can be found in the dissertation. As an aside, we point out that this was also demonstrated to some extent, seemingly separately, by Jeffers [37] who built their demonstration upon the work of Hennart and Valle [48].

3.2 Implementation in DRAGON5

In this section, we will look at how the DG method was implemented in the S_N solver of the Polytechnique Montréal lattice code, DRAGON5. The choice of the polynomial basis set for spanning the solution will be presented and similar to the HODD case, some example equations will be given. Also, anticipating the discussion around the hexagonal geometry and parallel computation implementation, the single-cell solution algorithm will be outlined.

3.2.1 Choice of function space

Once the DG equations have been obtained, as in Eqn. 3.27, all that is left is choosing the polynomial function space for the trial/test functions. From there, it is a simple matter of

some algebra to derive the equation for each cell unknown and the implementation. Theoretically, as long as function spaces have the same span, the numerical results should be similar. Hence, what matters would be the order and family of the function space – as well as numerical constraints, if any.

Order can sometimes be a confusing term, as distinct fields or even authors employ it differently. In this work, order will designate the highest power of either the x, y or z variable in the spanning polynomial function. The same highest power is used for every variable. The *family* of the function space is characterised by which cross-terms are kept for a given order. There are two main ones that tend to be used in finite element analysis, these being the *Lagrange* and *'serendipity'* families [49].

A common basis set used to expand the solution is the *Lagrange polynomials*. If we consider a line element with support points (points at which we wish to determine to value of the dependent variable) placed at equidistant intervals, Lagrange polynomials have the property of having a value of unity at one point, and zero at all the others, and are given by

$$l_k^n(x) = \frac{(x - x_0)(x - x_1)\dots(x - x_{k-1})(x - x_{k+1})\dots(x - x_n)}{(x_k - x_0)(x_k - x_1)\dots(x_k - x_{k-1})(x_k - x_{k+1})\dots(x_k - x_n)} = \prod_{\substack{i=0 \\ i \neq k}}^{n} \frac{(x - x_i)}{(x_k - x_i)}, \quad (3.29)$$

thereby being equal to 1 at x_k and zero at the other n points.

From this, one can devise a systematic way of generating shape functions for a solution space in any dimension, simply by multiplying Lagrange polynomials in the number of coordinates required. If we look at this solution space, it can be seen that all cross-terms are retained, with support points laid out in a regular grid. For this reason, the Lagrange basis set lends its name to the *Lagrange* family of polynomials. Indeed, other polynomial basis set can be part of the Lagrange as long as all cross-terms are included.

With the seredipity family however, not all cross-terms are included. Only support points along the edges of the cell element are retained[3]. They were originally devised heuristically (hence, the name) but a formal method has since been devised [49]. The direct consequence is in the reduced number of degrees of freedom or unknowns to solve for per cell.

For the Lagrange family, in 2D the total number of unknowns for Cartesian cell elements is $(\Lambda + 1)^2$ where Λ is the order, while for the serendipity family, this is $(\Lambda + 1)^2 - (\Lambda - 1)$, $1 \leq \Lambda \leq 3$. As a result, it is expected that the serendipity set would be faster, but also less accurate. It was known from the work of Bay [15] that high-order expansions were needed to

---

[3]This is true up to the cubic order; beyond that, some internal nodes need to be added to ensure completeness of the polynomials.

accurately resolve test cases involving FNRs. For this reason, accuracy was deemed important and it was chosen to go with the Lagrange family polynomials.

3.2.2   Lagrange vs Legendre polynomials in DRAGON5

We initially started with Lagrange polynomials as the spanning set for the solution space. However, various subroutines in DRAGON5 were programmed to work with the average flux within a cell element to compute outputs and parameters correctly. With the Legendre polynomials used with HODD, the averages just correspond to the zeroth moment of the flux. Indeed, if the average for a 1D reference element is defined as,

$$\bar{\phi}_i = \int_{-1/2}^{1/2} du \ \phi_{l,i}(u) \ , \tag{3.30}$$

with

$$\begin{aligned}
\phi_{l,i}(u) &= \sum_{n=1}^{M} \omega_n \widetilde{P}_l(\mu_n)\psi_{n,i}(u) \\
&= \sum_{n=1}^{M} \omega_n \widetilde{P}_l(\mu_n) \sum_{\alpha=0}^{\Lambda} \psi_{n,i}^{\alpha}\widetilde{P}_\alpha(u) \ ,
\end{aligned} \tag{3.31}$$

the average can be written as

$$\begin{aligned}
\bar{\phi}_i &= \sum_{n=1}^{M} \omega_n \widetilde{P}_l(\mu_n) \sum_{\alpha=0}^{\Lambda} \psi_{n,i}^{\alpha} \int_{-1/2}^{1/2} du \ \widetilde{P}_\alpha(u) \\
&= \sum_{n=1}^{M} \omega_n \widetilde{P}_l(\mu_n) \sum_{\alpha=0}^{\Lambda} \psi_{n,i}^{\alpha} \int_{-1/2}^{1/2} du \ \widetilde{P}_\alpha(u)\widetilde{P}_0(u) \ ,
\end{aligned} \tag{3.32}$$

since $\widetilde{P}_0(u) = 1$. Using the orthogonality property of Legendre polynomials, we can now write,

$$\bar{\phi}_i = \sum_{n=1}^{M} \omega_n \widetilde{P}_l(\mu_n)\psi_{n,i}^0 \ , \tag{3.33}$$

since $\int_{-1/2}^{1/2} du \ \widetilde{P}_\alpha(u)\widetilde{P}_0(u) = 0$ if $\alpha \neq 0$.

Now, this is not the case with Lagrange polynomials obviously. So there needed to be an added computation after each inner iteration to compute the averages for each element so that the scattering source for the next iteration could be correctly calculated. And although the number of degrees of freedom between HODD and DGFEM is the same for a given order and problem dimensionality, the main array of unknowns, termed `FUNKNO` in the code, had to be increased in length to accommodate these extra values. While, at the time, we did not immediately realise it, all of this caused our benchmarks running with DGFEM to be

perceivably slower than with HODD [50].

Once this was recognised, as a lot of the coding was already done with Lagrange polynomials, we tried using a Change-Of-Basis (COB) matrix to make the switch to Legendre polynomials. Unfortunately, this yielded mixed results. In the end, the solver was recoded from scratch using Legendre polynomials, and we will see in Sec. 3.3, the results were much better.

3.2.3 Implementation example and details

As an example, we will now derive the Legendre moments equations for the two-dimensional linear order case. The Legendre polynomials as well as the compact DGFEM transport equation are repeated respectively below for convenience,

$$
\begin{aligned}
\widetilde{P}_0(u) &= 1 \ , \\
\widetilde{P}_1(u) &= 2\sqrt{3}u \ , \\
\widetilde{P}_2(u) &= \sqrt{5}(6u^2 - 1/2) \ , \\
\widetilde{P}_3(u) &= \sqrt{7}(20u^3 - 3u) \ ,
\end{aligned}
\tag{3.34}
$$

and

$$
\int_{\xi} \left((\boldsymbol{\Omega}_n \cdot \nabla \psi_{n,h}) \, w_h + \Sigma \psi_{n,h} w_h \right) ds - \int_{\partial \xi_-} (\boldsymbol{n} \cdot \boldsymbol{\Omega}_n) \left(\psi_{n,h}^+ - \psi_{n,h}^- \right) w_h^+ dl = \int_{\xi} Q_n w_h ds \ . \tag{3.35}
$$

Assuming an orthogonal meshing (Fig. 3.2a) of the domain – as is the case in DRAGON5– such that each element ξ can also be denoted by conventional indices $\{i, j\}$. Each element has sides Δx_i and Δy_j. As with the HODD method, we apply a change of variable (Eqn. 3.5) such that we are on a reference element (shown in Fig. 3.2b), $\hat{\xi}$, bounded by $-1/2 \le u \le 1/2$ and $-1/2 \le v \le 1/2$.

We thus obtain

$$
\frac{\mu_n}{\Delta x_i} \int_{\hat{\xi}} du \ dv \ \widetilde{P}_\alpha(u) \widetilde{P}_\beta(v) \ \frac{\partial}{\partial u} \psi_{n,i,j}(u,v) + \frac{\eta_n}{\Delta y_j} \int_{\hat{\xi}} du \ dv \ \widetilde{P}_\alpha(u) \widetilde{P}_\beta(v) \ \frac{\partial}{\partial v} \psi_{n,i,j}(u,v) +
$$

$$
\int_{\hat{\xi}} du \ dv \ \widetilde{P}_\alpha(u) \widetilde{P}_\beta(v) \Sigma \psi_{n,i,j}(u,v) -
$$

$$
\frac{\mu_n}{\Delta x_i} \int_{\xi_{r-}} du \ (\boldsymbol{n} \cdot \boldsymbol{\Omega}_n) \ \widetilde{P}_\alpha(u) \left(\psi_{n,i,j}(u, \xi_{r-})|^+ - \psi_{n,i,j}(u, \xi_{r-})|^- \right) - \tag{3.36}
$$

$$
\frac{\eta_n}{\Delta y_j} \int_{\xi_{r-}} dv \ (\boldsymbol{n} \cdot \boldsymbol{\Omega}_n) \ \widetilde{P}_\beta(v) \left(\psi_{n,i,j}(\xi_{r-}, v)|^+ - \psi_{n,i,j}(\xi_{r-}, v)|^- \right) =
$$

$$
\int_{\hat{\xi}} du \ dv \ \widetilde{P}_\alpha(u) \widetilde{P}_\beta(v) Q_{n,i,j}(u,v) \ ,
$$

(a) Example of regular structured meshing in DRAGON5.

(b) Reference element: square of side length 1, centred at $(0,0)$.
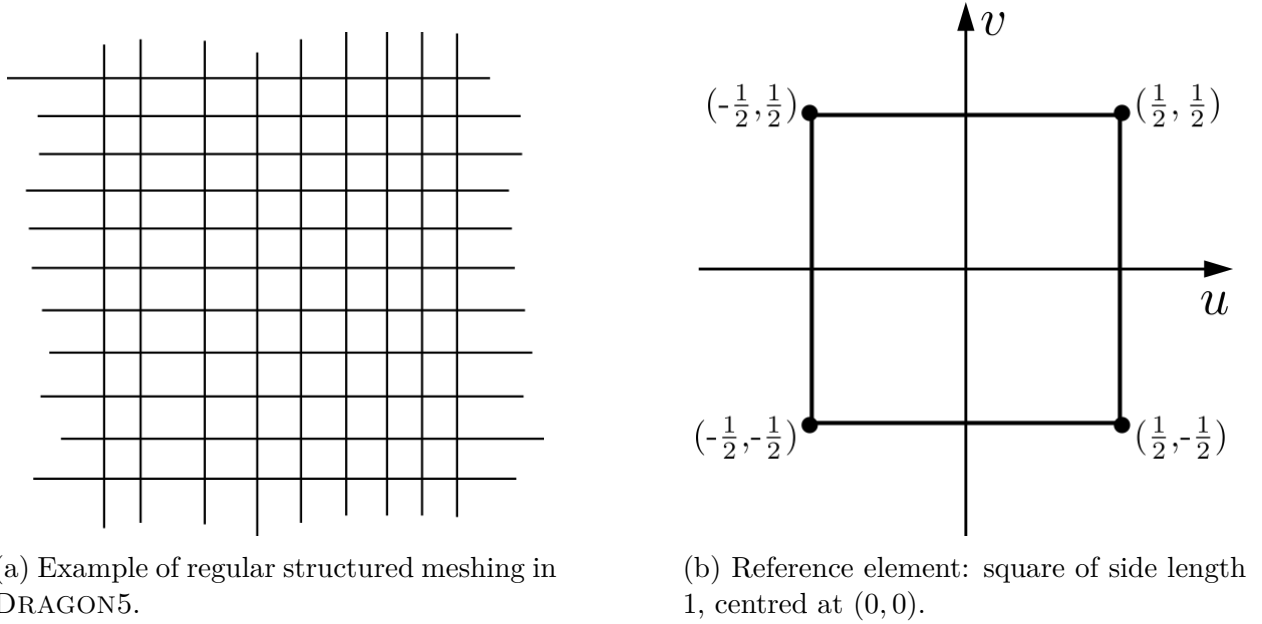
Figure 3.2 Illustration of meshing and reference element in 2D.

where we have dropped the h subscript in favour of the element indices $\{i, j\}$, and as before, the $\pm$ superscripts respectively indicates whether the restriction of the function on the incoming boundary is on the element or from the neighbouring inflow element. In the case of the latter, these would have been computed prior in the sweep and are hence *known* values and ultimately form part of the source.

Approximating the flux on an element, $\hat{\xi}$ as,

$$\psi_{n,i,j}(u,v) = \sum_{\alpha=0}^{\Lambda} \sum_{\beta=0}^{\Lambda} \widetilde{P}_\alpha(u) \widetilde{P}_\beta(v) \; \psi_{n,i,j}^{[\alpha,\beta]} \; , \tag{3.37}$$

this can be substituted into Eqn. 3.36 above. The equations for each Legendre moment, $\psi_{n,i,j}^{[\alpha,\beta]}$, can then be obtained after testing the transport equation against each of the combination of functions, $\widetilde{P}_\alpha(u)\widetilde{P}_\beta(v)$, with α or β having range $[0, \Lambda]^4$.

Before deriving the equations, we also would like to somewhat more elegantly represent the flux on the boundaries. It is not necessary: the flux restriction on the boundary from the incoming cells, *i.e.*, the Legendre moments that were calculated from previous neighbouring cells in the sweep can be used directly. However, we find the following representation way more intuitive as well as result in less confusingly written equations.

---

[4]To be explicit, for example, for linear order in 2D, $\Lambda = 1$, this means testing against $\widetilde{P}_0(u)\widetilde{P}_0(v)$, $\widetilde{P}_1(u)\widetilde{P}_0(v)$, $\widetilde{P}_0(u)\widetilde{P}_1(v)$, and $\widetilde{P}_1(u)\widetilde{P}_1(v)$.

Assuming that the flux on the bottom edge (*i.e.*, $v = -1/2$) of the cell can be expanded as

$$\varphi_{n,j-}(u) = \sum_{\alpha=0}^{\Lambda} \widetilde{P}_\alpha(u)\, \varphi_{n,j-}^{[\alpha]} , \tag{3.38}$$

and given the restriction of the adjacent flux on that boundary as

$$\psi_{n,i,j}(u, -1/2)|^- = \sum_{\alpha=0}^{\Lambda} \sum_{\beta=0}^{\Lambda} \widetilde{P}_\alpha(u) \widetilde{P}_\beta(-1/2)\, \psi_{n,i,j}^{[\alpha,\beta]}|^- , \tag{3.39}$$

an equivalence can be found between the two sets of Legendre moments, $\varphi_{n,j-}^{[\alpha]}$ and $\psi_{n,i,j}^{[\alpha,\beta]}|^-$. Similar relations can be found for each of the – in the 2D case – four boundaries.

For the 2D linear order, we hence derive in a direction-agnostic form,

$$\left(\frac{|\mu_n|}{\Delta x_i} + \frac{|\eta_n|}{\Delta y_j}\right)\psi_{n,i,j}^{[0,0]} + \sqrt{3}\frac{\mu_n}{\Delta x_i}\psi_{n,i,j}^{[1,0]} + \sqrt{3}\frac{\eta_n}{\Delta y_j}\psi_{n,i,j}^{[0,1]} + \Sigma_{i,j}\psi_{n,i,j}^{[0,0]} =$$
$$Q_{n,i,j}^{[0,0]} + \text{sgn}(\mu_n)\frac{\mu_n}{\Delta x_i}\varphi_{n,i-}^{[0]} + \text{sgn}(\eta_n)\frac{\eta_n}{\Delta y_j}\varphi_{n,j-}^{[0]}$$

$$-\sqrt{3}\frac{\mu_n}{\Delta x_i}\psi_{n,i,j}^{[0,0]} + \left(3\frac{|\mu_n|}{\Delta x_i} + \frac{|\eta_n|}{\Delta y_j}\right)\psi_{n,i,j}^{[1,0]} + \sqrt{3}\frac{\eta_n}{\Delta y_j}\psi_{n,i,j}^{[1,1]} + \Sigma_{i,j}\psi_{n,i,j}^{[1,0]} =$$
$$Q_{n,i,j}^{[1,0]} - \sqrt{3}\frac{\mu_n}{\Delta x_i}\varphi_{n,i-}^{[0]} + \text{sgn}(\eta_n)\frac{\eta_n}{\Delta y_j}\varphi_{n,j-}^{[1]}$$

$$\tag{3.40}$$

$$-\sqrt{3}\frac{\eta_n}{\Delta y_j}\psi_{n,i,j}^{[0,0]} + \left(\frac{|\mu_n|}{\Delta x_i} + 3\frac{|\eta_n|}{\Delta y_j}\right)\psi_{n,i,j}^{[0,1]} + \sqrt{3}\frac{\mu_n}{\Delta x_i}\psi_{n,i,j}^{[1,1]} + \Sigma_{i,j}\psi_{n,i,j}^{[0,1]} =$$
$$Q_{n,i,j}^{[0,1]} + \text{sgn}(\mu_n)\frac{\mu_n}{\Delta x_i}\varphi_{n,i-}^{[1]} - \sqrt{3}\frac{\eta_n}{\Delta y_j}\varphi_{n,j-}^{[0]}$$

$$-\sqrt{3}\frac{\eta_n}{\Delta y_j}\psi_{n,i,j}^{[1,0]} - \sqrt{3}\frac{\mu_n}{\Delta x_i}\psi_{n,i,j}^{[0,1]} + \left(3\frac{|\mu_n|}{\Delta x_i} + 3\frac{|\eta_n|}{\Delta y_j}\right)\psi_{n,i,j}^{[1,1]} + \Sigma_{i,j}\psi_{n,i,j}^{[1,1]} =$$
$$Q_{n,i,j}^{[1,1]} - \sqrt{3}\frac{\mu_n}{\Delta x_i}\varphi_{n,i-}^{[1]} - \sqrt{3}\frac{\eta_n}{\Delta y_j}\varphi_{n,j-}^{[1]} ,$$

with the inflow boundary Legendre moments given as,

$$
\begin{aligned}
\varphi_{n,j-}^{[0]} &= \psi_{n,i,j}^{[0,0]}|^- + \mathrm{sgn}(\mu_n)\sqrt{3}\psi_{n,i,j}^{[1,0]}|^- \\
\varphi_{n,j-}^{[1]} &= \psi_{n,i,j}^{[0,1]}|^- + \mathrm{sgn}(\mu_n)\sqrt{3}\psi_{n,i,j}^{[1,1]}|^- \\
\varphi_{n,i-}^{[0]} &= \psi_{n,i,j}^{[0,0]}|^- + \mathrm{sgn}(\eta_n)\sqrt{3}\psi_{n,i,j}^{[0,1]}|^- \\
\varphi_{n,i-}^{[1]} &= \psi_{n,i,j}^{[1,0]}|^- + \mathrm{sgn}(\eta_n)\sqrt{3}\psi_{n,i,j}^{[1,1]}|^- ,
\end{aligned}
\tag{3.41}
$$

where we have made use of the sign operator, sgn(). This can also be written in matrix form

$$
\mathbb{T}\Psi_{n,i,j} = \mathbb{Q}_{n,i,j} ,
\tag{3.42}
$$

where

$\mathbb{T} =$

$$
\begin{bmatrix}
\left(\frac{|\mu_n|}{\Delta x_i} + \frac{|\eta_n|}{\Delta y_j} + \Sigma_{i,j}\right) & \sqrt{3}\frac{\mu_n}{\Delta x_i} & \sqrt{3}\frac{\eta_n}{\Delta y_j} & 0 \\
-\sqrt{3}\frac{\mu_n}{\Delta x_i} & \left(3\frac{|\mu_n|}{\Delta x_i} + \frac{|\eta_n|}{\Delta y_j} + \Sigma_{i,j}\right) & 0 & \sqrt{3}\frac{\eta_n}{\Delta y_j} \\
-\sqrt{3}\frac{\eta_n}{\Delta y_j} & 0 & \left(\frac{|\mu_n|}{\Delta x_i} + 3\frac{|\eta_n|}{\Delta y_j} + \Sigma_{i,j}\right) & \sqrt{3}\frac{\mu_n}{\Delta x_i} \\
0 & -\sqrt{3}\frac{\eta_n}{\Delta y_j} & -\sqrt{3}\frac{\mu_n}{\Delta x_i} & \left(3\frac{|\mu_n|}{\Delta x_i} + 3\frac{|\eta_n|}{\Delta y_j} + \Sigma_{i,j}\right)
\end{bmatrix}
\tag{3.43}
$$

$$
\Psi_{n,i,j} =
\begin{bmatrix}
\psi_{n,i,j}^{[0,0]} \\
\psi_{n,i,j}^{[1,0]} \\
\psi_{n,i,j}^{[0,1]} \\
\psi_{n,i,j}^{[1,1]}
\end{bmatrix} ,
\tag{3.44}
$$

and

$$
\mathbb{Q}_{n,i,j} =
\begin{bmatrix}
Q_{n,i,j}^{[0,0]} + \mathrm{sgn}(\mu_n)\frac{\mu_n}{\Delta x_i}\varphi_{n,i-}^{[0]} + \mathrm{sgn}(\eta_n)\frac{\eta_n}{\Delta y_j}\varphi_{n,j-}^{[0]} \\
Q_{n,i,j}^{[1,0]} - \sqrt{3}\frac{\mu_n}{\Delta x_i}\varphi_{n,i-}^{[0]} + \mathrm{sgn}(\eta_n)\frac{\eta_n}{\Delta y_j}\varphi_{n,j-}^{[1]} \\
Q_{n,i,j}^{[0,1]} + \mathrm{sgn}(\mu_n)\frac{\mu_n}{\Delta x_i}\varphi_{n,i-}^{[1]} - \sqrt{3}\frac{\eta_n}{\Delta y_j}\varphi_{n,j-}^{[0]} \\
Q_{n,i,j}^{[1,1]} - \sqrt{3}\frac{\mu_n}{\Delta x_i}\varphi_{n,i-}^{[1]} - \sqrt{3}\frac{\eta_n}{\Delta y_j}\varphi_{n,j-}^{[1]}
\end{bmatrix} .
\tag{3.45}
$$

In DRAGON5, the DG method was implemented from the zeroth/flat up to the cubic order, *i.e.*, $0 \leq \Lambda \leq 3$ for the 1D/2D/3D Cartesian and hexagonal geometries (more on that in Chap. 4). Similar to HODD, variable-order expansion is not used, leading to the number of d.o.f.s per cell per angle to be also given by $(\Lambda + 1)^\zeta$, where ζ is the dimension. Of course, the matrices, $\mathbb{T}$, $\Psi_{n,i,j}$, and $\mathbb{Q}_{n,i,j}$, will change accordingly.

It is feasible to derive these by hand for simple cases. But at high dimensions and orders, the

matrices become quite large and it is much easier to use a scripting language. Indeed, for the 3D case at cubic order, the matrix of coefficients is of size 64×64. It would be unrealistic to derive this by hand. Hence, we used Matlab and its symbolic functions capabilities. Part of the scripts are given in Appendix A. The rest will be hosted on the DRAGON5 Archives webpage [51].

3.2.4 Single-cell and inner iteration solution algorithm

With the equations now in hand, we can have a look at the resolution algorithm for one cell element and how that fits in the inner iteration procedure. While this is essentially the same as with HODD, it is useful to document so as to better grasp the changes necessary for the hexagonal geometry and the parallelism implementations.

The procedure for a single cell for a single direction basically consists of four stages:

1. identifying the incoming faces/edges and picking up the incoming fluxes corresponding for the cell being solved. For the first iteration, this can consist of a rough estimate, a random number, unity or zero ;

2. assembling the matrix of coefficients and source vector to be computed according to Eqn. 3.40, more generally, Eqn. 3.36 ;

3. solving the system of equations ; and

4. computing the outgoing fluxes – which will correspond to the restriction of the adjacent flux on the boundary for some cell(s) further down in the sweep.

A slighted more detailed (compared to Sec. 2.2.5) look at one inner iteration is shown in Alg. 1. This is essentially the backbone of an inner iteration subroutine[5] in DRAGON5; there is one per dimension for each of HODD and DG. The single cell procedure outlined in points 1-4 above are represented at line numbers 15 to 18. Moreover, we would like to add the following comments:

■ The source term, including scattering and fission, has already been calculated outside of this algorithm.

■ The order in which the directions over the unit circle (sphere) are looped over does not matter much, but this is usually carried out quadrant by quadrant.

---

[5]The past tense might have been a more appropriate tense here. This algorithm was modified following the parallelism implementation, and does not exist in this exact form anymore – more on this in Chap. 6. That being said, this is suitable for the purpose of our discussion right now.

- The resolution of the system of equations, $\mathbb{T}\Psi_{n,i,j} = \mathbb{Q}_{n,i,j}$, is done in the `ALSBD` subroutine in DRAGON5. It uses a relatively straightforward Gaussian elimination method with partial pivoting.

- Depending on the direction of neutron travel, there are *four* main ways of sweeping the domain. While this might make intuitive sense for some, this is represented in Fig. 3.3. So, while we represented the loop in the $x-$ and $y-$ directions quite simply in the algorithm, the loops are usually reversed using a index.

- If the entries of all relevant arrays (such as the volume, material, flux, source and others) are all stored in the same particular order, then the sweep over the domain can be done *implicitly* without the use of a graph. This is exemplified in Alg. 1 by the use of the two loops over the elements in the $x-$ and $y-$ directions. Indeed, it is important to ensure that the outgoing fluxes are passed on to the correct elements during the sweep.

- The scalar fluxes $(\phi_{l,i,j}^{m,[\alpha,\beta]})$ and outgoing surface fluxes $(\varphi_{n,BC}^{[\alpha]}, \varphi_{n,BC}^{[\beta]})$ are all held in the same array in the code, `FUNKNO`.
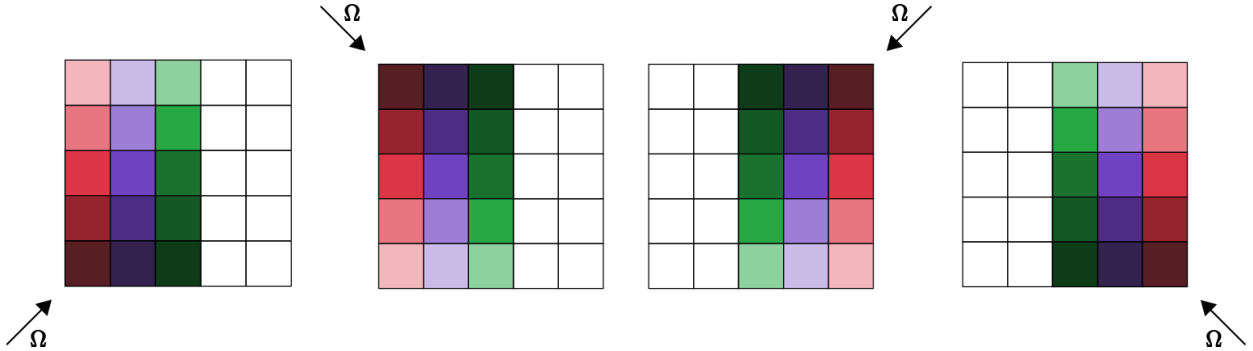
Figure 3.3 The four different ways that the columnar sweep proceeds depending on direction of neutron travel. The fading colour represents progression along the y-axis and the changing colour progression along the x-axis. The orientation of the domain is the same throughout.

3.3 Numerical Results

This section will present numerical results for two- and three-dimensional eigenvalue-problem benchmarks. These will serve to verify and validate the DGFEM implementation within the DRAGON5 S_N solver, as well as showcase the differences, if any, with the HODD method.

Results for both methods will be compared against reference solutions. These might have been obtained from the literature or from a calculation run at high angular and spatial

---

Algorithm 1: Representation of one inner iteration for the 2D case.

    input : $Q_{l,i,j}^{m,[\alpha,\beta]}$, $\varphi_{n,BC}^{[\alpha]}$, $\varphi_{n,BC}^{[\beta]}$

    output: $\phi_{l,i,j}^{m,[\alpha,\beta]}$, $\varphi_{n,BC}^{[\alpha]}$, $\varphi_{n,BC}^{[\beta]}$

    /* Initialise flux.   */

1 $\phi_{l,i,j}^{m,[\alpha,\beta]} = 0.0$

    /* Loop over directions in unit sphere.   */

2   **for** $n = 1$ **to** $N(N+2)/2$ **do**

        /* Swap $\varphi_{n,BC}^{[\cdot]}$ with $\varphi_{m,BC}^{[\cdot]}$ such that $\boldsymbol{\Omega}_m \cdot \boldsymbol{n} = -\boldsymbol{\Omega}_n \cdot \boldsymbol{n}$ if needed   */

        /* $\boldsymbol{n}$ is the normal to the boundary in question   */

3   **if** $u = \pm 1/2$ *or* $v = \pm 1/2$ *boundary is reflective* **then**

4 $\Upsilon = \varphi_{n,BC}^{[\alpha/\beta]}$ /* Υ is a temporary variable.   */

5 $\varphi_{n,BC}^{[\alpha/\beta]} \leftarrow \varphi_{m,BC}^{[\alpha/\beta]}$

6 $\varphi_{m,BC}^{[\alpha/\beta]} \leftarrow \Upsilon$

        /* Loop over elements in domain.   */

7   **for** $i = 1$ **to** l_x **do**   /* l_x = # of elements along x-axis */

8   **if** $i == 1$ **then**

9 $\varphi_{n,i-}^{[\beta]} \leftarrow \varphi_{n,BC}^{[\beta]}$

10   **else**

11 $\varphi_{n,i-}^{[\beta]} \leftarrow \varphi_{n,i+}^{[\beta]}$

12   **for** $j = 1$ **to** l_y **do**

13   **if** $j == 1$ **then**

14 $\varphi_{n,j-}^{[\alpha]} \leftarrow \varphi_{n,BC}^{[\alpha]}$

15   **else**

16 $\varphi_{n,j-}^{[\alpha]} \leftarrow \varphi_{n,j+}^{[\alpha]}$

17   Assemble matrix of coefficients, $\mathbb{T}$, and source vector, $\mathbb{Q}_{n,i,j}$.

18   Solve system of equations $\mathbb{T}\Psi_{n,i,j} = \mathbb{Q}_{n,i,j}$.

19   Compute the outgoing fluxes, $\varphi_{n,i+}^{[\beta]}$ and $\varphi_{n,j+}^{[\alpha]}$.

20   Sum angular fluxes over directions to obtain scalar flux such that

21 $\phi_{l,i,j}^{m,[\alpha,\beta]} = \phi_{l,i,j}^{m,[\alpha,\beta]} + 2\omega_n \psi_{l,i,j}^{m,[\alpha,\beta]}$.

            /* Store out. fluxes on $v = \pm 1/2$ for next inner iteration.   */

22 $\varphi_{n,BC}^{[\alpha]} \leftarrow \varphi_{n,j+}^{[\alpha]}$

        /* Store outgoing fluxes on $u = \pm 1/2$ for next inner iteration.   */

23 $\varphi_{n,BC}^{[\beta]} \leftarrow \varphi_{n,j+}^{[\beta]}$

discretisation with a relatively fine submeshing. In the latter, HODD will be used for the spatial discretisation as that is the already established method in DRAGON5. A mix of eigenvalues (the k_{eff}), errors on group-wise absorption rates, and computational times will be given. The absorption rate is calculated using

$$R_{\text{a},i} = \frac{1}{V_i} \int_{V_i} d^3r \left[\Sigma(\boldsymbol{r}) - \Sigma_{\text{s}}(\boldsymbol{r}) \right] \phi(\boldsymbol{r}) \ , \tag{3.46}$$

where i denotes the cell element of calculation, and V_i its volume. The maximum and average errors are then respectively worked out on these reaction rate values using the equations below:

$$\epsilon_{\text{max}} = \max_i \left\{ \frac{|R_{\text{a},i} - R_{\text{a},i}^*|}{R_{\text{a},i}^*} \right\} \text{ and } \bar{\epsilon} = \frac{1}{V_{\text{core}}} \sum_i V_i \frac{|R_{\text{a},i} - R_{\text{a},i}^*|}{R_{\text{a},i}^*} \ . \tag{3.47}$$

where the asterisk denotes the reference value, and V_{core} is the volume of the whole domain. In the case of multi-group problems, this is carried out for each group, and the largest values are then retained across all groups.

The results will be detailed for various parameters:

- the angular order, N,

- the spatial order, Λ, and

- the spatial submesh, *subm.*, i.e., the number of sub-regions into which each cell element of the domain is further divided.

Finally, a convergence criterion of 1×10^{-5} was applied on both the source (inner) iteration and power (outer) loops. It should be noted that in DRAGON5, the convergence criterion is applied to the whole unknown flux vector, the aforementioned FUNKNO. This is interesting because it means the convergence is also tested on the outgoing surface boundary fluxes.

3.3.1 One-group 2D simple benchmark: 2D-CNS

This is a simple monoenergetic criticality benchmark with anisotropic scattering that was devised by Hébert [35]. As we first used it in a conference paper [50] in the Canadian Nuclear Society (CNS) annual meeting, we will refer to it as the 2D-CNS benchmark in this work.

Given the simplicity of this benchmark, it was used mostly as a quick litmus test to verify whether our DGFEM implementation was correct. We also frequently used it as a simple benchmark when debugging. Fig. 3.4 shows the domain geometry and material mixtures, with the cross-section data given in Tab. 3.1. Reflective boundary conditions were used on the left and bottom sides of the domain while vacuum is applied to the right and top sides.

The initial meshing (before any further submeshing) was an equidistant 5×5 grid.
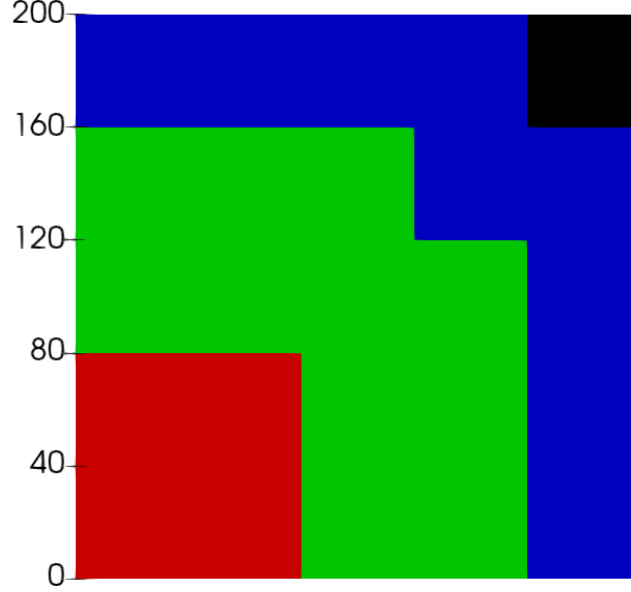
Figure 3.4 Description of the simple monoenergetic 2D benchmark, 2D-CNS. The dimensions are in cm. The domain is symmetric along the diagonal, and reflective boundary conditions are applied on left and bottom sides, and vacuum on the right and top. See Table 3.1 for the cross-section data. Red corresponds to mixture 1, green to 2, and blue to 3. The black region indicates a void. The un-submeshed calculation grid is 5×5.

A reference solution was calculated using an S_{18} HODD-2 discretisation with a submeshing of 5. The obtained results are summarised in Tab. 3.2. As mentioned, HODD is only implemented up to parabolic order, *i.e.*, $\Lambda = 2$, so this part of the table is empty. Also empty is DG-0, *i.e.*, the flat order of DGFEM. While this is technically present, DG-0 performs so poorly that this is not worth including. Going forward, this will be true for the rest of the document.

The results are also partly represented graphically in Figures 3.5 to 3.7, with plots of the maximum error, ϵ_{max}, against the number of unknowns. Here, we are using the number of unknowns in the whole domain, *per direction, per group and per anisotropy level.* Accounting

Table 3.1 Cross-section data for the 2D-CNS benchmark.

Mix.	Σ (cm^{-1})	Σ_{s0} (cm^{-1})	Σ_{s1} (cm^{-1})	$\nu\Sigma_{\text{f}}$ (cm^{-1})
1	0.025	0.013	0.0	0.0155
2	0.025	0.024	0.006	0.0
3	0.075	0.0	0.0	0.0

for these would not change the shape of the plots, just shift them further up. Hence, this leaves the number of unknowns as the number of cell elements multiplied by the number of Legendre moments to be solved for.

An aside here about the number of unknowns: It was mentioned earlier that the number of d.o.f.s or unknowns is given by $(\Lambda + 1)^\zeta$ for both HODD and DGFEM, where Λ is the expansion order for each dimension, and ζ is the number of dimensions. We would like to draw the reader's attention to the fact that depending on the implementation (see subsection 3.1.5), this might not always be the case. For example, it can be seen in the work of [46], where the number of unknowns for HODD is given as $(\Lambda + 1)^\zeta + \zeta(\Lambda + 1)$.

Looking over the results from Tab. 3.2, we can see that the results are very similar for comparable parameters. There are some slight differences, with no submeshing for linear order, HODD appears to consistently produce better results. However, looking at the maximum error for the same data points, we can see that the error for DG is actually quite smaller, roughly half as small. This is perhaps better illustrated in Figures 3.5-3.7 with the convergence plots. The linear case is where the difference is more clear, with DG dropping to around the converged maximum error with just one mesh refinement.
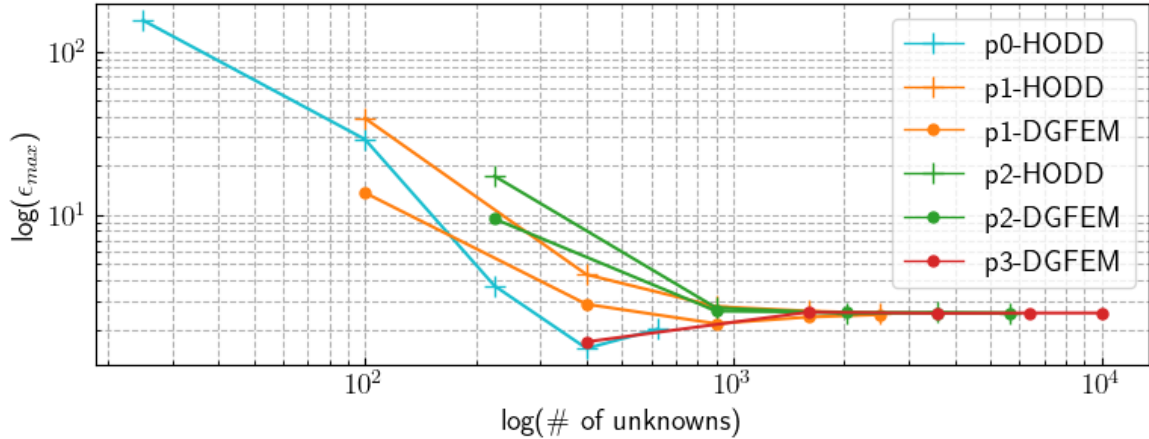
Figure 3.5 2D-CNS benchmark: S_4 convergence rates for the maximum error, ϵ_{max}, as a function of the number of unknowns.
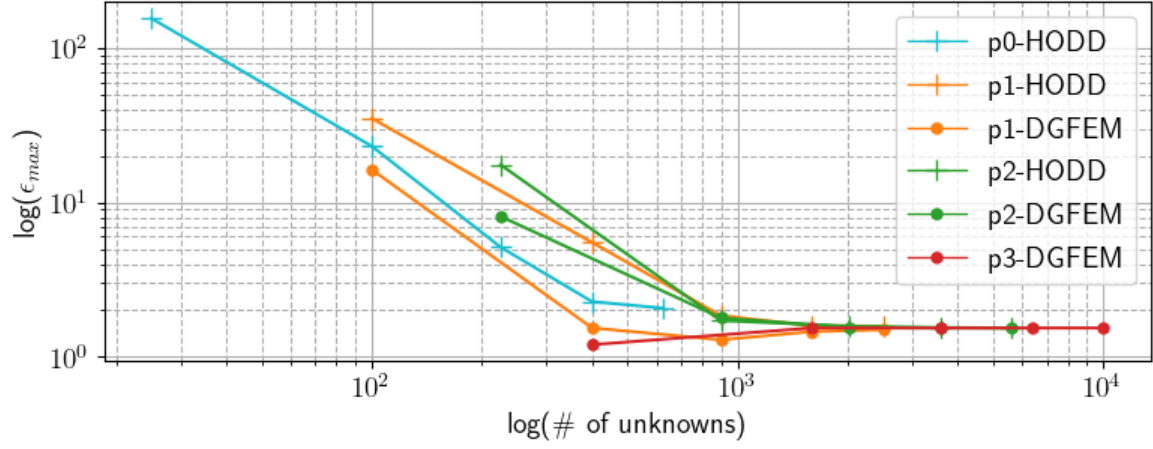
Figure 3.6 2D-CNS benchmark: S_6 convergence rates for the maximum error, ϵ_{max}, as a function of the number of unknowns.
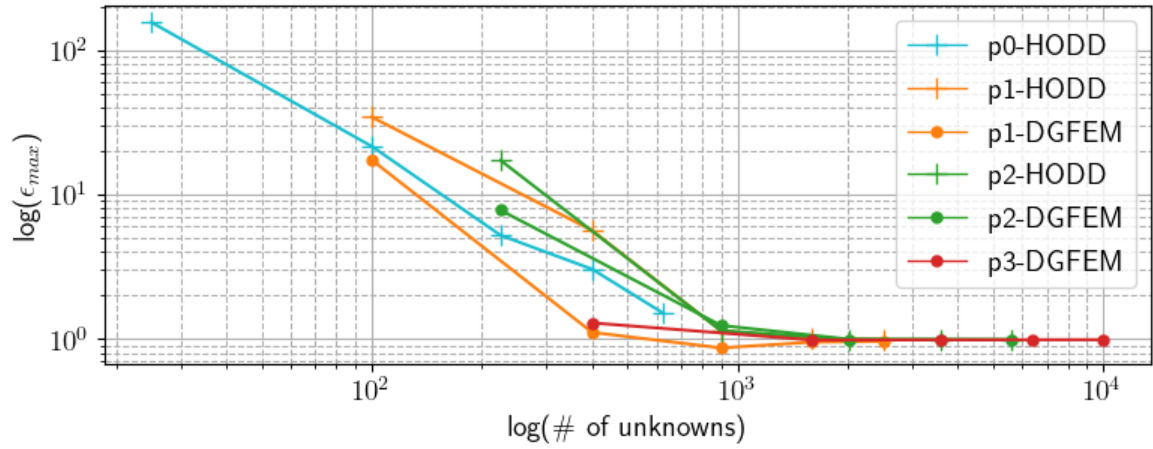
Figure 3.7 2D-CNS benchmark: S_8 convergence rates for the maximum error, ϵ_{max}, as a function of the number of unknowns.

Table 3.2 Summarised results for the 2D-CNS benchmark. Λ denotes the polynomial order, and *subm.* the submeshing. Three significant figures are given, except for k_{eff}.

S_N	Λ	*subm.*	DD					DG				
			$k_{\text{eff}}^{\dagger}$	Δk_{eff} (pcm)	$\epsilon_{\max}$ (%)	$\bar{\epsilon}$ (%)	Time (s)	$k_{\text{eff}}^{\dagger}$	Δk_{eff} (pcm)	$\epsilon_{\max}$ (%)	$\bar{\epsilon}$ (%)	Time (s)
4	0	1	0.986067	-623	155	25	0.005	-	-	-	-	-
		2	0.9903339	-196	29.1	3.67	0.012	-	-	-	-	-
		3	0.9911037	-120	3.69	1.21	0.045	-	-	-	-	-
		4	0.9913744	-92	1.53	0.836	0.04	-	-	-	-	-
		5	0.9914996	-79.9	2.01	0.88	0.074	-	-	-	-	-
	1	1	0.991689	-60.9	39.1	6.82	0.012	0.990421	-188	13.7	3.87	0.014
		2	0.9917193	-57.9	4.33	0.949	0.045	0.991528	-77.0	2.85	0.959	0.036
		3	0.9917208	-57.8	2.76	0.864	0.083	0.9916601	-63.8	2.17	0.855	0.08
		4	0.9917211	-57.7	2.59	0.855	0.14	0.9916945	-60.4	2.38	0.855	0.179
		5	0.9917212	-57.7	2.51	0.85	0.256	0.9917072	-59.1	2.46	0.854	0.217
	2	1	0.9917202	-57.8	17.3	2.7	0.031	0.9917133	-58.5	9.48	1.66	0.03
		2	0.9917212	-57.7	2.71	0.867	0.191	0.9917208	-57.8	2.59	0.852	0.152
		3	0.9917212	-57.7	2.5	0.85	0.308	0.9917211	-57.7	2.54	0.852	0.282
		4	0.9917213	-57.7	2.54	0.852	0.451	0.9917212	-57.7	2.52	0.851	0.461
		5	0.9917213	-57.7	2.51	0.85	0.693	0.9917212	-57.7	2.52	0.851	0.678
	3	1	-	-	-	-	-	0.9917208	-57.8	1.68	0.809	0.089
		2	-	-	-	-	-	0.9917212	-57.7	2.55	0.853	0.35
		3	-	-	-	-	-	0.9917212	-57.7	2.51	0.851	0.805
		4	-	-	-	-	-	0.9917213	-57.7	2.52	0.851	1.43
		5	-	-	-	-	-	0.9917212	-57.7	2.52	0.851	2.23
6	0	1	0.986241	-606	155	24.9	0.007	-	-	-	-	-
		2	0.9906151	-168	23.1	2.91	0.022	-	-	-	-	-
		3	0.9913819	-91.7	5.13	1.16	0.059	-	-	-	-	-
		4	0.9916507	-64.8	2.28	0.658	0.092	-	-	-	-	-
		5	0.9917746	-52.4	2.07	0.596	0.131	-	-	-	-	-
	1	1	0.991968	-33.0	35	6.48	0.03	0.9907091	-159	16.3	3.88	0.024
		2	0.9919937	-30.5	5.51	0.784	0.076	0.9918028	-49.6	1.54	0.578	0.118
		3	0.9919946	-30.4	1.85	0.49	0.174	0.9919335	-36.5	1.29	0.494	0.161
		4	0.9919948	-30.4	1.59	0.475	0.291	0.991968	-33.1	1.46	0.485	0.257
		5	0.9919949	-30.4	1.58	0.476	0.445	0.9919807	-31.8	1.5	0.48	0.46
	2	1	0.9919942	-30.4	17.5	2.55	0.069	0.9919858	-31.3	8.11	1.29	0.057
		2	0.9919949	-30.4	1.72	0.487	0.227	0.9919943	-30.4	1.79	0.488	0.24
		3	0.9919949	-30.4	1.58	0.476	0.503	0.9919947	-30.4	1.55	0.474	0.488
		4	0.991995	-30.4	1.55	0.474	0.933	0.9919949	-30.4	1.54	0.473	0.865
		5	0.9919949	-30.4	1.54	0.473	1.43	0.9919949	-30.4	1.54	0.473	1.41

Table 3.2 *(continued)*

			DD					DG				
S_N	Λ	*subm.*	$k_{\text{eff}}^{\dagger}$	Δk_{eff} (pcm)	$\epsilon_{\max}$ (%)	$\bar{\epsilon}$ (%)	Time (s)	$k_{\text{eff}}^{\dagger}$	Δk_{eff} (pcm)	$\epsilon_{\max}$ (%)	$\bar{\epsilon}$ (%)	Time (s)
3		1	-	-	-	-	-	0.9919944	-30.4	1.2	0.536	0.198
		2	-	-	-	-	-	0.9919949	-30.4	1.54	0.473	0.736
		3	-	-	-	-	-	0.9919949	-30.4	1.54	0.473	1.67
		4	-	-	-	-	-	0.9919949	-30.4	1.54	0.473	3
		5	-	-	-	-	-	0.9919949	-30.4	1.54	0.473	4.44
8	0	1	0.9863902	-591	155	24.9	0.008	-	-	-	-	-
		2	0.9907833	-152	21.5	2.8	0.068	-	-	-	-	-
		3	0.991543	-75.5	5.21	1.07	0.099	-	-	-	-	-
		4	0.9918108	-48.8	3.04	0.527	0.157	-	-	-	-	-
		5	0.9919342	-36.4	1.51	0.362	0.216	-	-	-	-	-
	1	1	0.992126	-17.2	34.3	6.42	0.042	0.9908831	-142	17.3	3.85	0.042
		2	0.9921527	-14.6	5.63	0.692	0.131	0.9919631	-33.5	1.11	0.368	0.138
		3	0.9921537	-14.5	1.14	0.25	0.292	0.9920926	-20.6	0.866	0.278	0.277
		4	0.992154	-14.4	1.01	0.247	0.469	0.9921269	-17.2	0.95	0.261	0.443
		5	0.9921541	-14.4	0.987	0.246	0.702	0.9921398	-15.9	0.961	0.254	0.702
	2	1	0.9921531	-14.5	17.2	2.43	0.105	0.9921443	-15.4	7.77	1.17	0.09
		2	0.9921541	-14.4	1.14	0.258	0.449	0.9921533	-14.5	1.24	0.26	0.408
		3	0.9921541	-14.4	0.991	0.247	0.964	0.9921539	-14.5	0.994	0.247	0.929
		4	0.9921541	-14.4	0.991	0.247	1.6	0.9921541	-14.4	0.989	0.247	1.41
		5	0.9921541	-14.4	0.987	0.247	2.29	0.9921541	-14.4	0.987	0.247	2.25
	3	1	-	-	-	-	-	0.9921534	-14.5	1.29	0.362	0.34
		2	-	-	-	-	-	0.9921541	-14.4	0.982	0.246	1.3
		3	-	-	-	-	-	0.9921541	-14.4	0.985	0.247	2.82
		4	-	-	-	-	-	0.9921541	-14.4	0.986	0.247	4.75
		5	-	-	-	-	-	0.9921541	-14.4	0.987	0.247	7.73

† Reference k_{eff} is 0.9922985, obtained using an S_{18} HODD-2 method with a submeshing of 5.

3.3.2 Four-group 2D AIC assembly: 2D-AIC

The 2D-AIC benchmark was also devised by Hébert [52] and is quite a bit more complex. It is a four-group mock-up of a production Pressurised Water Reactor (PWR) assembly and features large spatial and angular anistropic effects because of the presence of inserted poison pins. These pins are made of a silver-indium-cadmium alloy – these elements have respective symbols Ag, In and Cd, hence the name AIC.

The domain is represented in Fig. 3.8, and the four-group cross-section data given in Tab.

3.3. Reflective boundary conditions are used on all sides of the domain to simulate an infinite lattice, as is usually the case in industry lattice calculations.

This benchmark was very much simplified from its original case [53]: it was made purely Cartesian and the number of energy groups decreased from 26 to 4, with limits at 53, 4 and 0.353 eV. However, Hébert [52] points out that many characteristics from the actual production assembly are kept. For example, the difference in k_{eff} between an isotropic (P_0) and anisotropic (P_1) scattering source is around ≈ 2380 pcm.
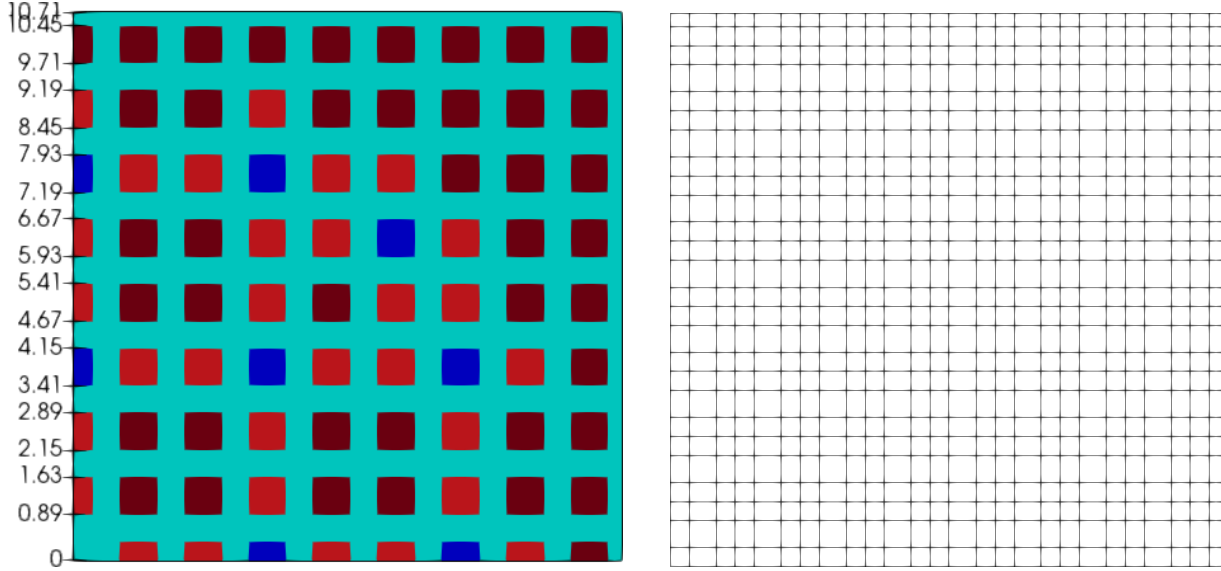
Figure 3.8 Representation of the 2D-AIC benchmark domain on the left. Reflective boundary conditions are applied on all sides. The dimensions are in cm, and the cross-section data is given in Tab. 3.3. Cyan is mix 1, red mix 2, dark red mix 3 and blue mix 4. On the right is the initial computational mesh before any further refinement.

Similarly to the previous benchmark, the reference solution was calculated using an S_{18} HODD-2 discretisation with a five-fold mesh refinement, and the results are summarised in Table 3.4. These calculations were run in series with no synthetic acceleration on the Digital Research Alliance of Canada (DRAC) cluster Graham.

The results are very similar to the 2D-CNS benchmark. The DGFEM implementation performs very similarly to HODD. One thing of note is that for higher order (especially parabolic and above), it would seem that mesh refinement does not really do much – at least, for this benchmark. The solution has already converged. Except perhaps in the case of HODD-1, it would seem that it struggles a bit more with the unrefined mesh, yielding values for the k_{eff} further away from its converged values, compared to DG. The maximum error results are also partially represented in Figure 3.9.
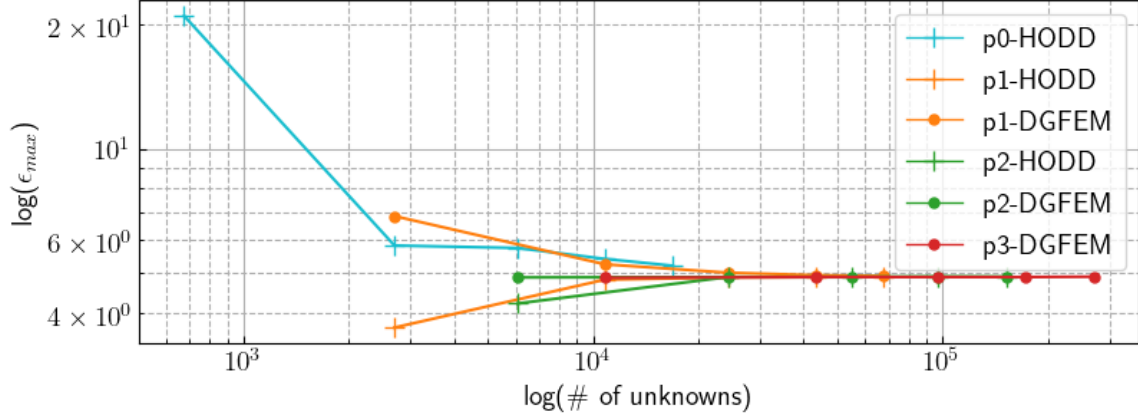
(a) With HODD-0.
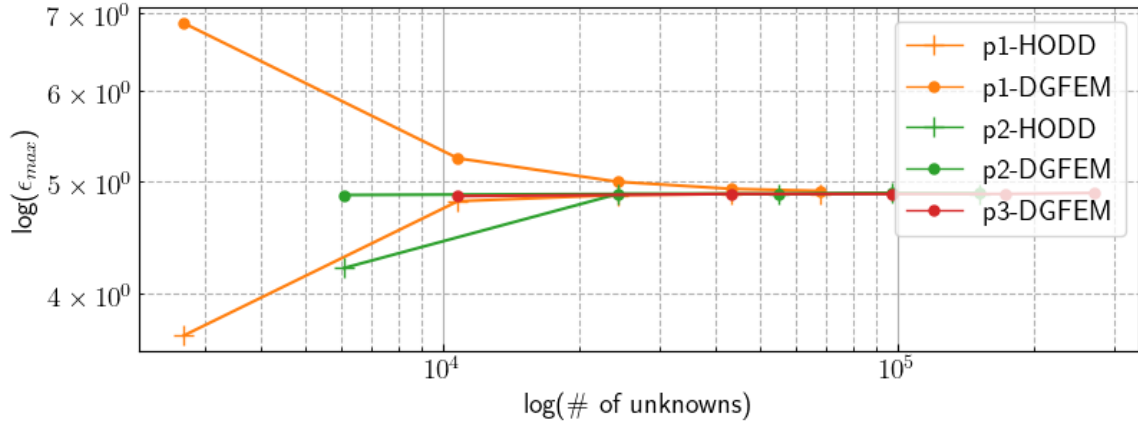
(b) Without HODD-0.

Figure 3.9 2D-AIC benchmark: S_6 convergence rates for the maximum error, ϵ_{max}, as a function of the number of unknowns. The second plot forgoes HODD-0 for better clarity.

The computational time taken the computations are very much in line with what we would expect. As the number of mesh elements or directions increase, the CPU time increases proportionally. For example, going from S_4 to S_6, the number of directions doubles, and so does the time taken. There are small differences between the two methods, with HODD being very slightly faster. Plots of errors in k_{eff} against time are given in Figure 3.10.

We have noticed that running calculations on the DRAC can lead to slight variations in the computational time taken for computations. These can range anywhere from 1% to around 15%. We speculate that this might be due to things like how much load is on the node at the time or maybe the data is not stored in a contiguous manner. In any case, while this has not been investigated, they *appear* to be random. So, with that in mind,
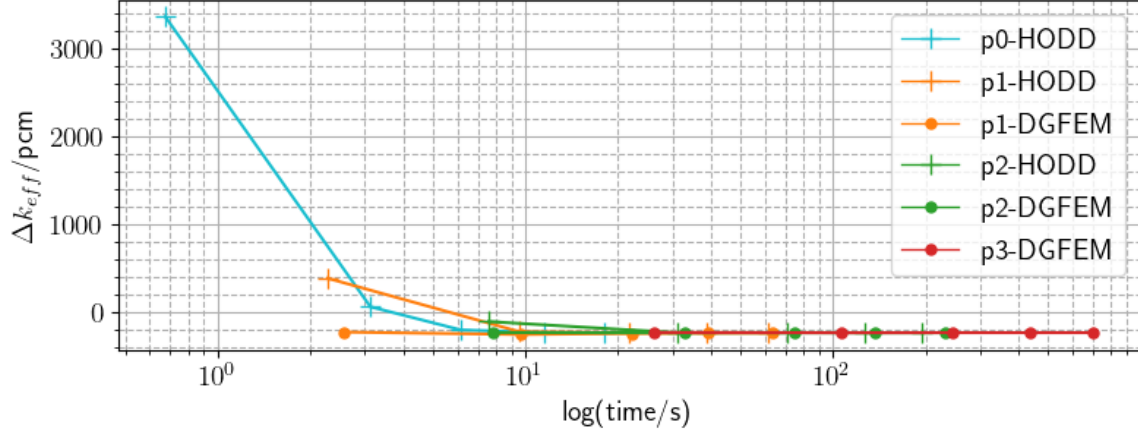
(a) With HODD-0.
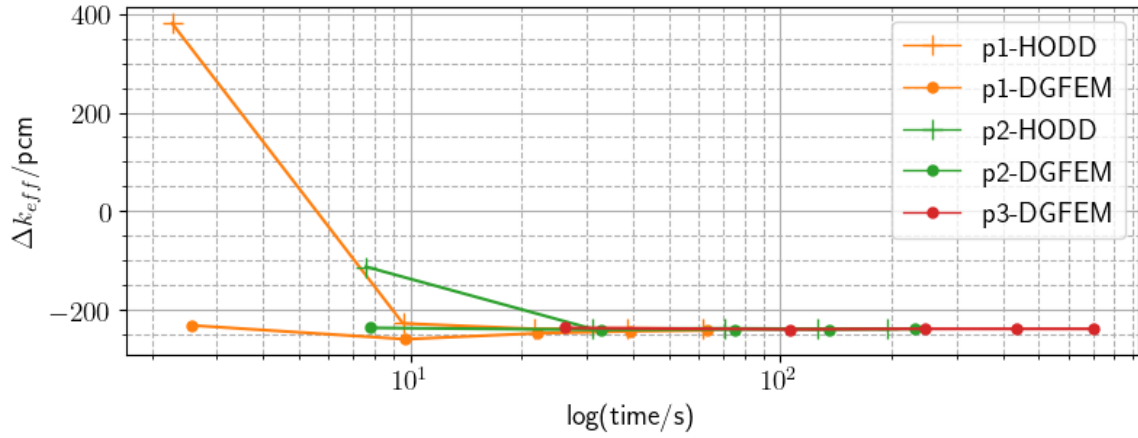
(b) Without HODD-0.

Figure 3.10 2D-AIC benchmark: S_6 convergence rates for the k_{eff}, as a function of CPU time. The second plot forgoes HODD-0 for better clarity.

given that the computational times seem to be consistently faster for HODD, we think that this might be the method, and not these variations.

Finally, we thought it would be interesting to look at surface plots of the scalar plots. These were generated using the open-source data-analysis and visualisation application, ParaView [54]. Figure 3.11 shows these surface plots for S_8 HODD/DG-1 with a mesh refinement of 2, as well as the reference calculation. There is no discernible differences between HODD and DG when looking at only these. And comparing with the reference solution, both do very well approximating the solution with the lower resolution.

With ParaView, we can also subtract between two data sets to find the differences as long as

the underlying mesh is the same. Hence, Figure 3.12 shows the differences between the scalar fluxes for HODD and DGFEM from Figure 3.11. The largest variation is in the lowest energy group but this variation is mostly confined to the poison pins. In fact, this is mostly true across all groups. This is not entirely surprising as it was expected for HODD to struggle more in anisotropic regions. Still, this variation is only about 1%.

Table 3.3 Cross-section data for the 2D-AIC benchmark. All fission neutrons are emitted in group $g = 1$. The energy group limits are at 53, 4 and 0.353 eV.

g	Mix	Σ^g (cm^{-1})	$\Sigma_{s0}^{g\to1}$ (cm^{-1})	$\Sigma_{s0}^{g\to2}$ (cm^{-1})	$\Sigma_{s0}^{g\to3}$ (cm^{-1})	$\Sigma_{s0}^{g\to4}$ (cm^{-1})	$\Sigma_{s1}^{g\to1}$ (cm^{-1})	$\Sigma_{s1}^{g\to2}$ (cm^{-1})	$\Sigma_{s1}^{g\to3}$ (cm^{-1})	$\Sigma_{s1}^{g\to4}$ (cm^{-1})	$\nu\Sigma_{f}^g$ (cm^{-1})
1	1	0.5316	0.4973	3.134E-2*	2.304E-3	2.228E-4	0.2735	1.410E-2	3.050E-4	1.245E-5	0.0
	2	0.4097	0.3908	8.546E-4	0.0	0.0	0.04793	-2.640E-4	0.0	0.0	0.01632
	3	0.4072	0.3885	8.562E-4	0.0	0.0	0.04981	-2.624E-4	0.0	0.0	0.01652
	4	0.4240	0.3738	2.439E-4	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	1	0.9352	0.0	0.6570	0.2528	2.412E-2	0.0	0.4253	0.1203	3.373E-3	0.0
	2	0.6286	0.0	0.4407	8.057E-3	0.0	0.0	5.419E-3	-2.453E-3	0.0	0.09259
	3	0.6223	0.0	0.4387	8.591E-3	0.0	0.0	3.100E-3	-2.610E-3	0.0	0.09136
	4	1.0476	0.0	0.4102	7.370E-4	0.0	0.0	-6.161E-2	0.0	0.0	0.0
3	1	0.9931	0.0	0.0	0.7176	0.2707	0.0	0.0	0.4286	6.426E-2	0.0
	2	0.4594	0.0	0.0	0.3785	1.203E-2	0.0	0.0	9.014E-3	-2.572E-3	0.1129
	3	0.4599	0.0	0.0	0.3780	1.229E-2	0.0	0.0	1.142E-2	-2.604E-3	0.1143
	4	2.7503	0.0	0.0	0.4956	3.248E-3	0.0	0.0	-2.326E-1	0.0	0.0
4	1	1.5931	0.0	0.0	1.163E-2	1.5650	0.0	0.0	5.064E-3	0.4384	0.0
	2	0.7508	0.0	0.0	5.318E-3	0.3983	0.0	0.0	-7.338E-4	8.769E-3	0.6508
	3	0.7504	0.0	0.0	5.236E-3	0.3984	0.0	0.0	-7.231E-4	8.764E-3	0.6500
	4	11.194	0.0	0.0	4.985E-3	0.2996	0.0	0.0	0.0	0.0	0.0

*: 3.134E-2 should be read as 3.134×10^{-2}.

Table 3.4 Summarised results for the 2D-AIC benchmark. Λ denotes the polynomial order, and *subm.* the submeshing. Three significant figures are given, except for k_{eff}.

S_N	Λ	*subm.*	DD					DG				
			$k_{\text{eff}}^{\dagger}$	Δk_{eff} (pcm)	$\epsilon_{\max}$ (%)	$\bar{\epsilon}$ (%)	Time (s)	$k_{\text{eff}}^{\dagger}$	Δk_{eff} (pcm)	$\epsilon_{\max}$ (%)	$\bar{\epsilon}$ (%)	Time (s)
4	0	1	0.9469103	3240	21.2	5.1	0.367	-	-	-	-	-
		2	0.9128222	-170	9.83	2.97	1.63	-	-	-	-	-
		3	0.9107411	-379	9.5	3.04	3.26	-	-	-	-	-
		4	0.9105441	-398	9.07	3.06	5.67	-	-	-	-	-
		5	0.9105403	-399	8.87	3.02	9.16	-	-	-	-	-
	1	1	0.9167795	225	7.36	1.74	1.24	0.9103752	-415	10.1	3.75	1.2
		2	0.9105386	-399	8.58	2.93	5.18	0.9102691	-426	8.82	3.09	5.14
		3	0.91043	-410	8.59	2.98	11.2	0.9103753	-415	8.65	3.02	11.7
		4	0.9104243	-410	8.59	2.99	20.2	0.9103971	-413	8.62	3	21.6
		5	0.9104274	-410	8.6	2.99	31.2	0.9104031	-412	8.61	3	35.1
	2	1	0.9113917	-314	8.33	2.76	4.04	0.9105515	-398	8.6	2.87	4.37
		2	0.9104236	-410	8.59	2.99	15.8	0.9104217	-410	8.59	2.98	17.1
		3	0.9104274	-410	8.6	2.99	36.1	0.9104132	-411	8.6	2.99	41
		4	0.9104271	-410	8.6	2.99	65.2	0.9104182	-411	8.6	2.99	75.2
		5	0.9104228	-410	8.6	2.99	99.4	0.9104192	-411	8.6	2.99	125
	3	1	-	-	-	-	-	0.9104783	-405	8.58	2.97	14
		2	-	-	-	-	-	0.9104142	-411	8.6	2.99	57.5
		3	-	-	-	-	-	0.9104202	-411	8.6	2.99	130
		4	-	-	-	-	-	0.9104206	-411	8.6	2.99	231
		5	-	-	-	-	-	0.9104206	-411	8.6	2.99	370
6	0	1	0.948131	3360	21	5.11	0.674	-	-	-	-	-
		2	0.9150842	55.7	5.82	1.54	3.13	-	-	-	-	-
		3	0.9124346	-209	5.74	1.84	6.18	-	-	-	-	-
		4	0.9121562	-237	5.4	1.81	11.5	-	-	-	-	-
		5	0.9121342	-239	5.2	1.76	18	-	-	-	-	-
	1	1	0.918336	381	3.68	1.21	2.27	0.9122064	-232	6.86	2.57	2.56
		2	0.9122439	-228	4.81	1.63	9.55	0.9119289	-260	5.24	1.81	9.7
		3	0.9121355	-239	4.87	1.64	21.6	0.9120476	-248	5	1.71	22.1
		4	0.9121441	-238	4.88	1.64	38.6	0.912086	-244	4.93	1.68	39.2
		5	0.9121396	-239	4.88	1.64	61.5	0.9121041	-242	4.91	1.66	63.7
	2	1	0.9133948	-113	4.21	1.45	7.59	0.9121546	-237	4.87	1.59	7.8
		2	0.9121409	-239	4.88	1.64	31	0.9121141	-241	4.88	1.66	32.9
		3	0.9121445	-238	4.88	1.64	70.5	0.9121271	-240	4.88	1.65	75.1
		4	0.9121394	-239	4.89	1.64	126	0.9121315	-240	4.89	1.64	136
		5	0.9121381	-239	4.88	1.64	195	0.9121333	-239	4.89	1.64	232

Table 3.4 *(continued)*

			DD					DG				
S_N	Λ	*subm.*	$k_{\text{eff}}^{\dagger}$	Δk_{eff} (pcm)	$\epsilon_{\max}$ (%)	$\bar{\epsilon}$ (%)	Time (s)	$k_{\text{eff}}^{\dagger}$	Δk_{eff} (pcm)	$\epsilon_{\max}$ (%)	$\bar{\epsilon}$ (%)	Time (s)
3		1	-	-	-	-	-	0.9121557	-237	4.86	1.67	26.1
		2	-	-	-	-	-	0.9121296	-240	4.88	1.64	106
		3	-	-	-	-	-	0.912136	-239	4.88	1.64	245
		4	-	-	-	-	-	0.9121364	-239	4.88	1.64	436
		5	-	-	-	-	-	0.9121361	-239	4.89	1.64	700
8	0	1	0.9477884	3330	21.1	4.97	1.08	-	-	-	-	-
		2	0.9155012	97.4	3.54	1.06	4.47	-	-	-	-	-
		3	0.9131584	-137	3.46	1.21	10	-	-	-	-	-
		4	0.912832	-170	3.25	1.15	18.2	-	-	-	-	-
		5	0.91283	-170	3.07	1.07	29.2	-	-	-	-	-
	1	1	0.9188837	436	4.06	1.06	3.66	0.9130214	-151	4.9	1.87	3.79
		2	0.9129794	-155	2.66	0.927	15.1	0.9126708	-186	3.14	1.12	15.6
		3	0.9128875	-164	2.77	0.927	34.5	0.9127844	-174	2.89	1.01	35.7
		4	0.9128937	-163	2.77	0.933	60.2	0.9128295	-170	2.83	0.976	63.7
		5	0.9128892	-164	2.77	0.936	99	0.9128516	-168	2.8	0.96	103
	2	1	0.9142289	-29.8	2.01	0.764	12.4	0.9128666	-166	2.71	0.901	13
		2	0.9128928	-163	2.78	0.928	50.5	0.9128571	-167	2.77	0.95	51.8
		3	0.9128887	-164	2.77	0.937	112	0.9128776	-165	2.77	0.94	122
		4	0.9128849	-164	2.77	0.938	201	0.9128816	-164	2.77	0.939	218
		5	0.9128858	-164	2.77	0.938	325	0.9128825	-164	2.77	0.939	354
	3	1	-	-	-	-	-	0.9128822	-164	2.74	0.973	42.4
		2	-	-	-	-	-	0.9128827	-164	2.78	0.938	172
		3	-	-	-	-	-	0.9128833	-164	2.77	0.939	392
		4	-	-	-	-	-	0.9128833	-164	2.77	0.939	711
		5	-	-	-	-	-	0.9128848	-164	2.77	0.938	1090

$\dagger$ Reference k_{eff} is 0.9145269, obtained using an S_{18} HODD-2 method with a submeshing of 5.

3.3.3 Four-group 3D small FNR core, Takeda Model 2: 3D-TAK2

The 3D-TAK2 benchmark is a three-dimensional isotropic four-group five-mixture eigenvalue problem, documented by Takeda and Ikeda [16]. There are two variations of this benchmark, one with the control rod completely withdrawn, and one where it is half inserted. We have chosen to do the latter as it presents more complexity. The quoted literature values for the k_{eff} are given in Table 3.5.

(a) S_8 HODD-1 *subm*.2 (b) S_8 DG-1 *subm*.2 (c) Ref. calc., (S_{18} HODD-2 *subm*.5)
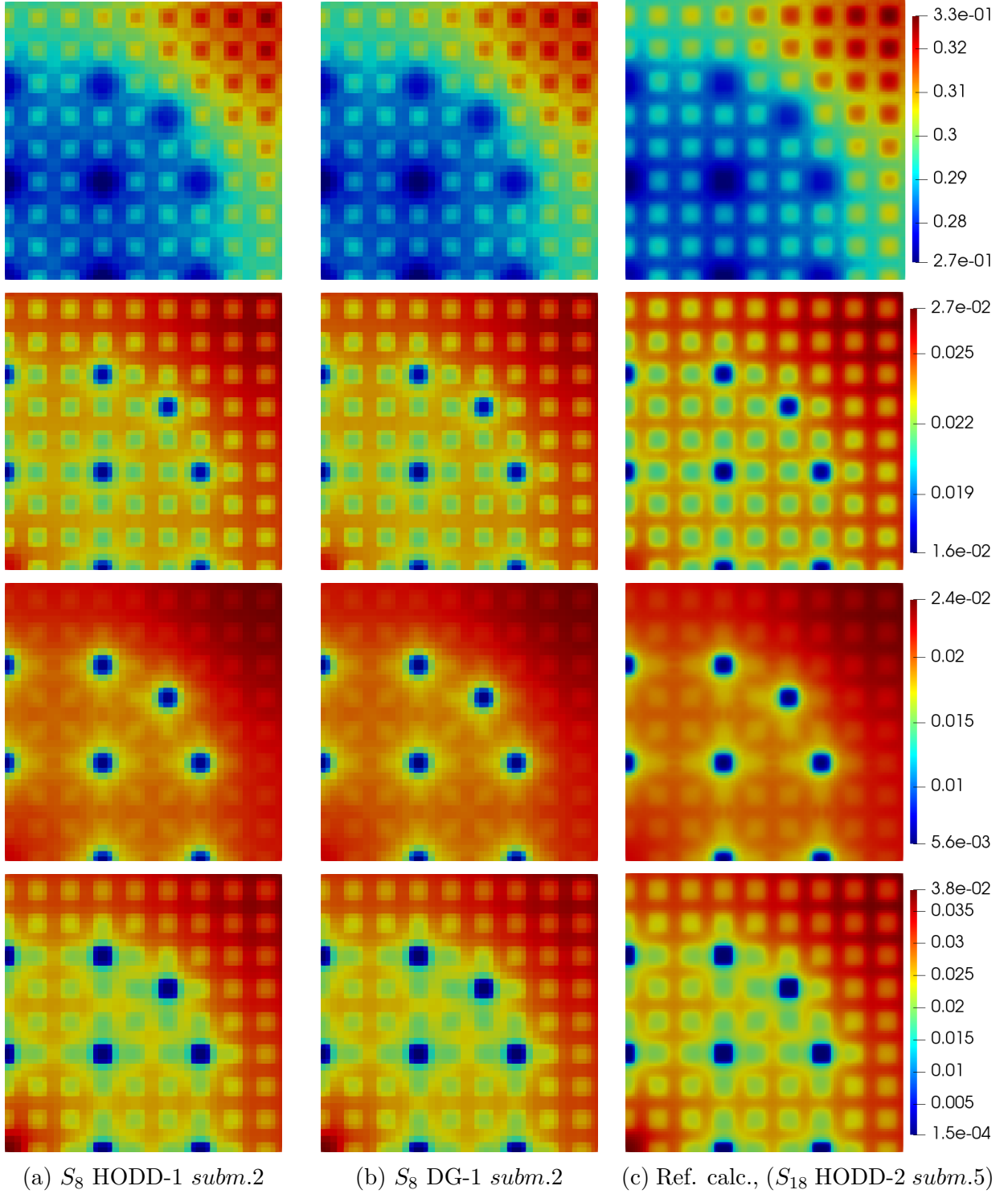
Figure 3.11 2D-AIC benchmark: scalar flux surface plots for energy groups 1 to 4 given in that order, with energy group 1 at the top. These contrast HODD and DGFEM (at lower discretisation levels), as well as the reference calculation.
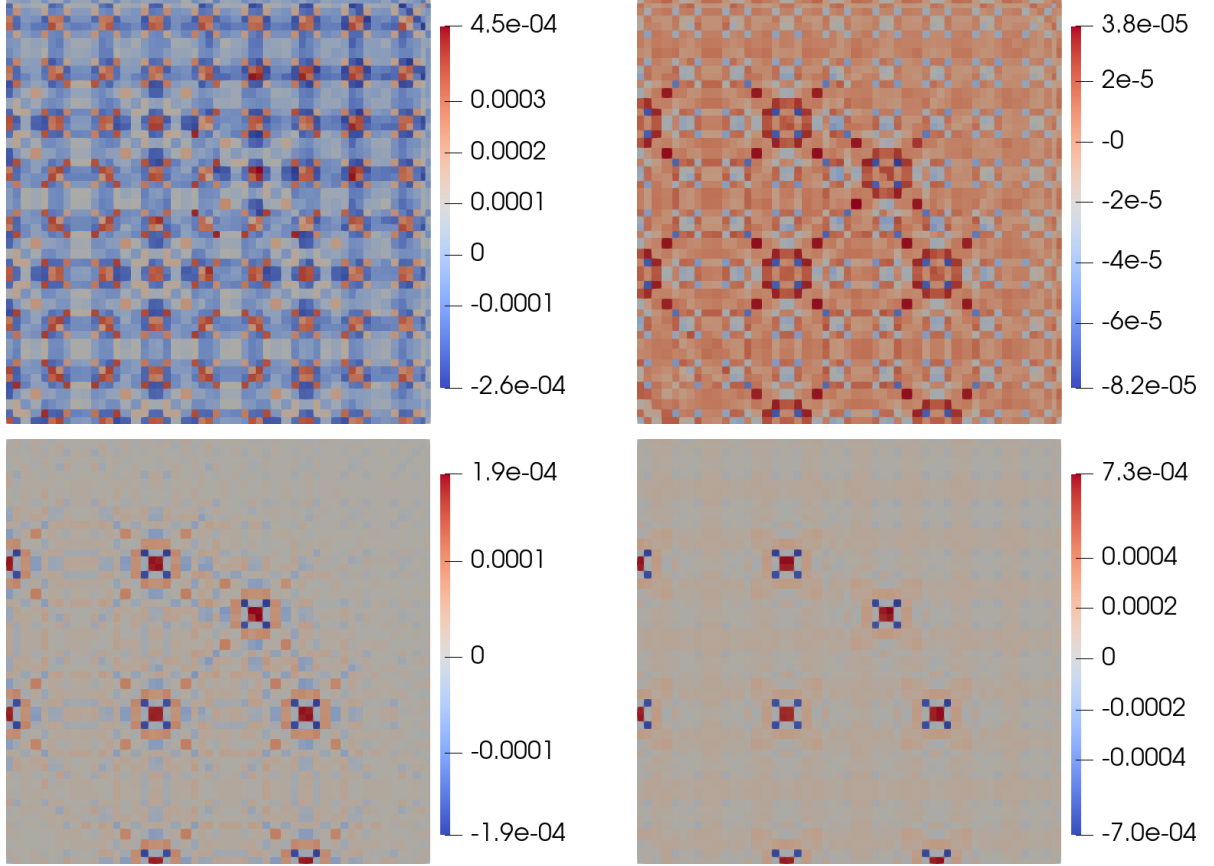
Figure 3.12 2D-AIC benchmark: surface plots of the difference between the two scalar flux maps of HODD and DGFEM from Figure 3.11, for the four energy groups. From left to right, top to bottom: group 1, group 2, group 3 and group 4.

Table 3.5 Quoted literature values for various methods for the 3D-TAK2 benchmark, reproduced from Takeda and Ikeda [16].

Method		k_{eff}	precision
Monte Carlo	‖	0.9589	±0.0002
S_4	‖	0.9594	±0.0001
S_8	‖	0.9593	±0.0002
P_7	‖	0.9647	-

A quarter of the core is shown in Figure 3.13, as well as the initial computational mesh. While this initial mesh is not the one recommended by Takeda and Ikeda [16], the highest mesh refinement used in the calculations roughly approximates it. The cross sections have not been reproduced in this document but can be found in [16].

Since we only have the k_{eff} in the literature, we chose to also run a *calculated reference* using an S_{10} HODD-2 discretisation with a submeshing of 4. With this level of refinement, it actually very closely resembles Takeda's recommended mesh. The results are summarised in Table 3.6, and we also provide convergence plots (Figures 3.14 and 3.15), as well as scalar fluxes surface plots (Figures 3.16 and 3.17).

Overall, the results are very much in line with what we expect, and have seen so far with 2D cases. The calculated reference k_{eff} is respectively within 35 pcm (or a precision of 0.00035) and 75 pcm from the S_8 and Monte Carlo quoted the literature values – which is quite acceptable.

We do observe from the summarised results and convergence plots (Figures 3.14 and 3.15) that for this isotropic test case, HODD-1 (and even to some extent HODD-0) outperforms DG-1. This is in line with the expectation that with a smoother solution, HODD shines better than DG.

With the surface plots of the scalar flux (Figure 3.16) and the difference between HODD and DG (Figure 3.17), we find that the scalar fluxes are again very similar, with differences centred around the control rod again. In this region, while DG seems to offer a smoother flux, HODD seems to overestimate in one mesh, and then underestimate in the neighbouring one so as to compensate.

(a) Top view. (b) Side view. (c) Overall view.
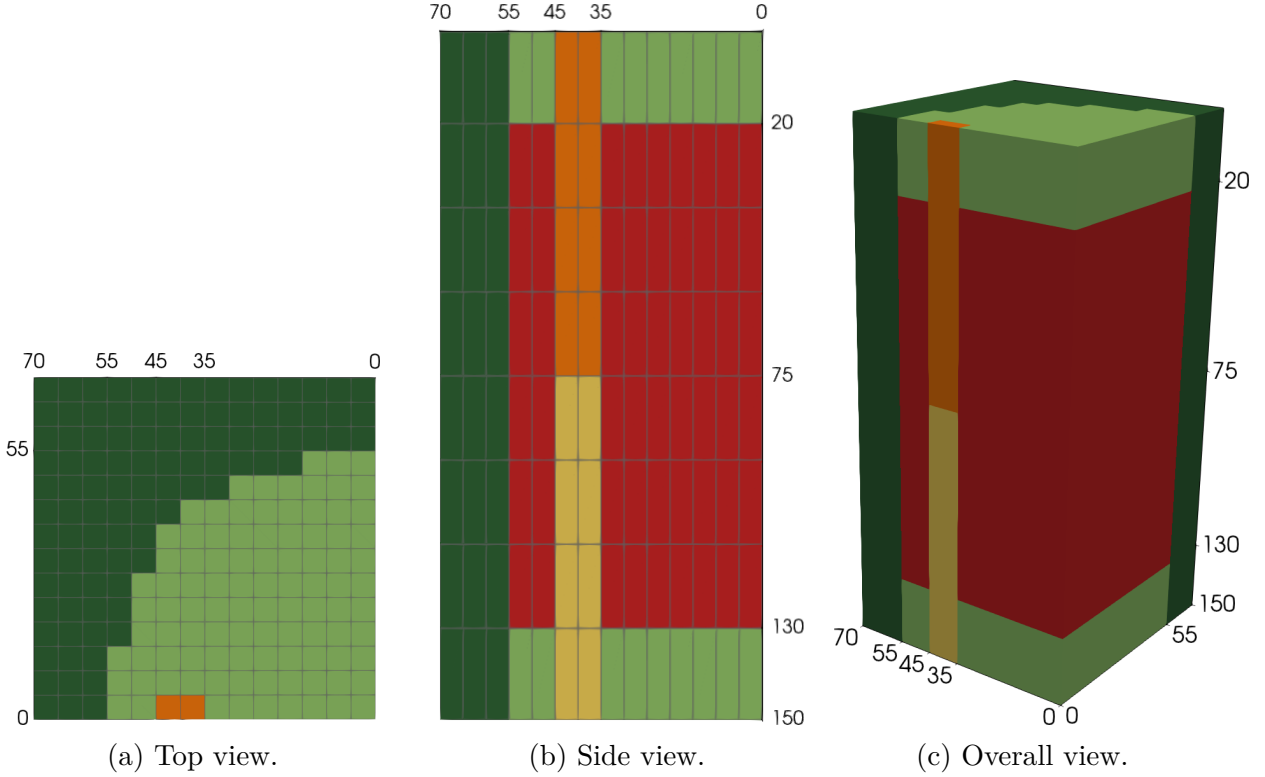
Figure 3.13 Domain of the 3D-TAK2 benchmark at various angles. Dimensions are in cm. Reflective boundary conditions are used on the inner sides (right and bottom on (a), and out-of-page on (c)) while vacuum boundary conditions are applied to the top, bottom, and outer sides. The initial computational mesh before subsequent submeshing is shown on (a) and (b). Red is the core, light green the axial blanket, dark green the radial blanket, orange the control rod and yellow the sodium-filled Control Rod Position (CRP).
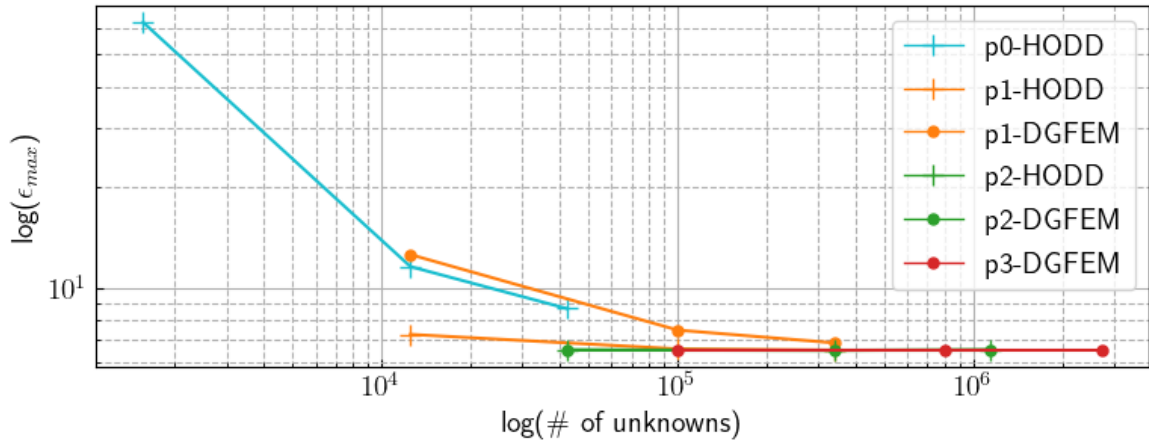
Figure 3.14 3D-TAK2 benchmark: S_4 convergence rates for the maximum error, ϵ_{max}, as a function of the number of unknowns.
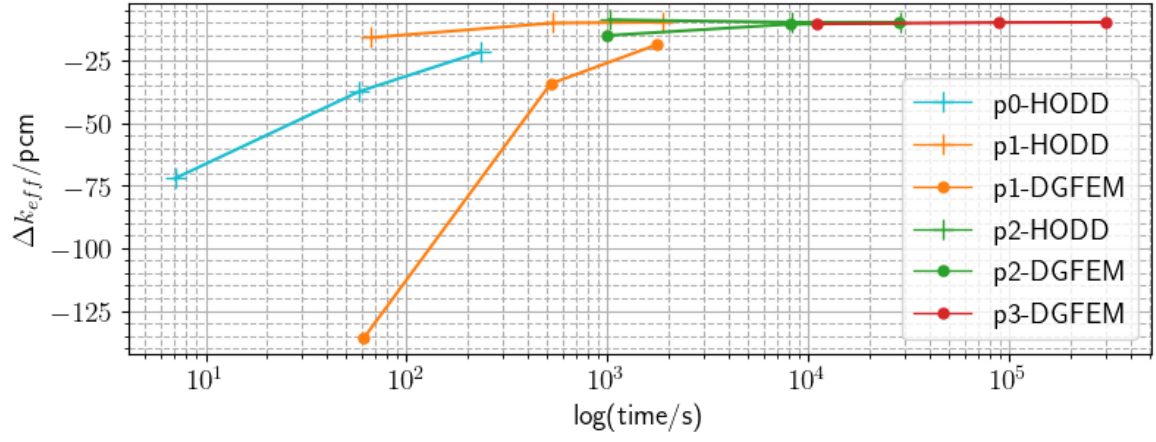
Figure 3.15 3D-TAK2 benchmark: S_4 convergence rates for the k_{eff}, as a function of CPU time.

(a) S_4 HODD-1 $subm.1$ (b) S_4 DG-1 $subm.1$ (c) Ref. calc., (S_{10} HODD-2 $subm.4$)
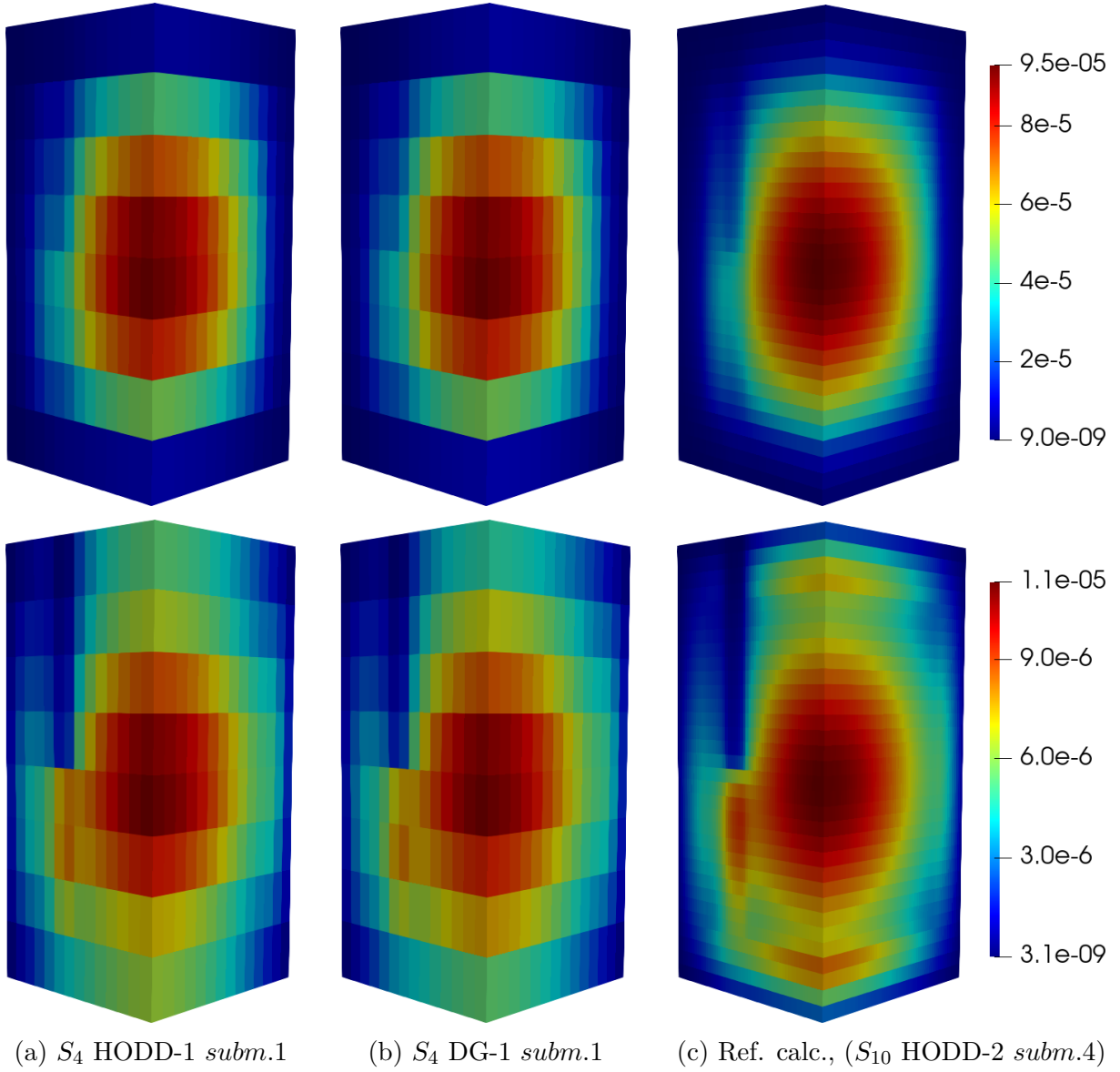
Figure 3.16 3D-TAK2 benchmark: scalar flux surface plots for energy groups 1 and 4 given in that order, with energy group 1 at the top. These contrast HODD and DGFEM (at lower discretisation levels), as well as the reference calculation. The orientation of the domain is the same as that used in the overall representation of Figure 3.13.
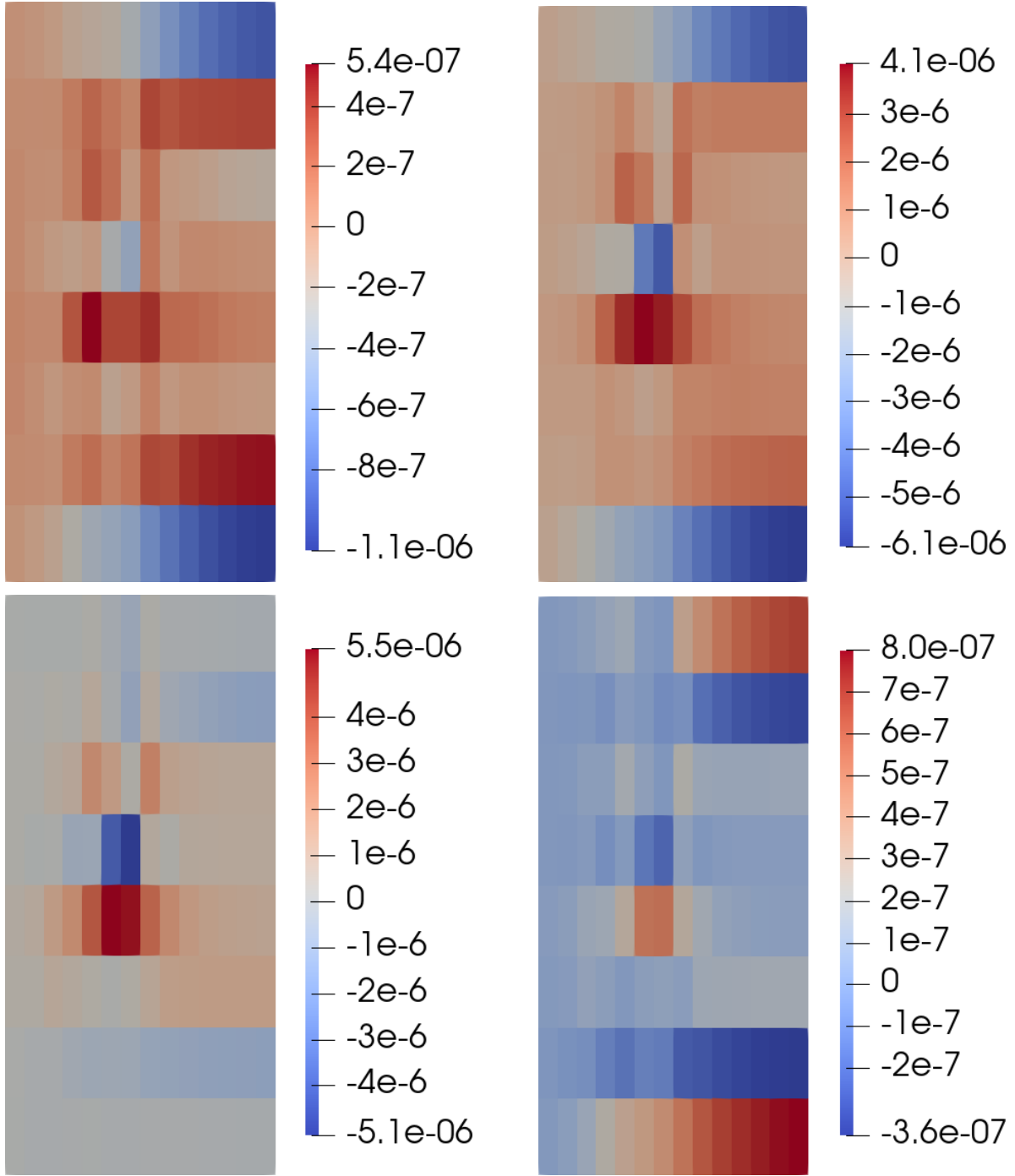
Figure 3.17 3D-TAK2 benchmark: surface plots of the difference between the two S_4 scalar flux maps of HODD-1 and DG-1 from Figure 3.16, for the four energy groups. From left to right, top to bottom: group 1, group 2, group 3 and group 4. This side orientation was chosen for clarity and because it shows most of the salient details. It is the same as the side orientation shown in Figure 3.13.

Table 3.6 Summarised results for the 3D-TAK2 benchmark. Λ denotes the polynomial order, and *subm.* the submeshing. Three significant figures are given, except for k_{eff}.

			DD					DG				
S_N	$O.$	*subm.*	$k_{\text{eff}}^{\dagger}$	Δk_{eff} (pcm)	$\epsilon_{\max}$ (%)	$\bar{\epsilon}$ (%)	Time (s)	$k_{\text{eff}}^{\dagger}$	Δk_{eff} (pcm)	$\epsilon_{\max}$ (%)	$\bar{\epsilon}$ (%)	Time (s)
2	0	1	0.9584814	-117	105	14.9	2.48	-	-	-	-	-
		2	0.9589372	-71.5	90.4	9.22	20.9	-	-	-	-	-
		3	0.9590641	-58.8	86.7	8.49	81.1	-	-	-	-	-
	1	1	0.9591543	-49.8	86.5	7.72	22.5	0.9577716	-188	92	9.5	20.7
		2	0.9591693	-48.3	87.8	7.9	184	0.9589058	-74.7	90.1	8.13	182
		3	0.9591702	-48.2	88.1	7.9	697	0.9590738	-57.9	89.2	7.98	638
	2	1	0.9591698	-48.3	87.8	7.92	355	0.9591303	-52.2	89.1	7.89	331
		2	0.9591707	-48.2	88.2	7.9	2750	0.9591627	-49	88.4	7.9	2760
		3	0.9591705	-48.2	88.3	7.9	9720	0.9591676	-48.5	88.3	7.9	9320
	3	1	-	-	-	-	-	0.9591612	-49.1	88.4	7.91	3630
		2	-	-	-	-	-	0.9591692	-48.3	88.3	7.9	29100
		3	-	-	-	-	-	0.95917	-48.2	88.3	7.9	98900
4	0	1	0.9589314	-72.1	62.4	8.11	7.03	-	-	-	-	-
		2	0.9592796	-37.3	11.6	2.05	58.4	-	-	-	-	-
		3	0.9594365	-21.6	8.69	1.35	233	-	-	-	-	-
	1	1	0.9594949	-15.8	7.27	1.22	66.5	0.9582973	-136	12.6	2.7	61.1
		2	0.9595526	-10	6.58	0.874	536	0.959309	-34.3	7.49	1.12	520
		3	0.959555	-9.7	6.52	0.882	1880	0.959465	-18.7	6.86	0.958	1750
	2	1	0.9595656	-8.7	6.54	0.919	1020	0.9595026	-15	6.54	0.861	988
		2	0.9595556	-9.7	6.51	0.884	8290	0.9595474	-10.5	6.54	0.886	8080
		3	0.9595554	-9.7	6.55	0.885	28900	0.9595525	-10	6.54	0.885	28300
	3	1	-	-	-	-	-	0.9595493	-10.3	6.55	0.892	10900
		2	-	-	-	-	-	0.9595543	-9.8	6.55	0.884	88000
		3	-	-	-	-	-	0.9595549	-9.7	6.55	0.884	300000
6	0	1	0.9589735	-67.9	61.4	7.46	17	-	-	-	-	-
		2	0.9593314	-32.1	6.12	1.45	120	-	-	-	-	-
		3	0.9594978	-15.5	3.75	0.793	455	-	-	-	-	-
	1	1	0.9595516	-10.1	5.17	0.785	129	0.95837	-128	11.9	2.16	122
		2	0.9596105	-4.2	2.92	0.33	1060	0.959371	-28.1	2.87	0.559	1020
		3	0.9596136	-3.9	2.89	0.333	3860	0.9595249	-12.8	2.74	0.404	3520
	2	1	0.9596297	-2.3	3.98	0.402	2090	0.9595611	-9.1	2.9	0.32	1970
		2	0.9596143	-3.8	2.89	0.335	16900	0.959606	-4.6	2.92	0.337	16100
		3	0.9596141	-3.8	2.89	0.336	57000	0.9596112	-4.1	2.9	0.336	55800

Table 3.6 *(continued)*

S_N	$O.$	*subm.*	$k_{\text{eff}}^\dagger$ (DD)	Δk_{eff} (pcm)	$\epsilon_{\max}$ (%)	$\bar{\epsilon}$ (%)	Time (s)	$k_{\text{eff}}^\dagger$ (DG)	Δk_{eff} (pcm)	$\epsilon_{\max}$ (%)	$\bar{\epsilon}$ (%)	Time (s)
3	1	-	-	-	-	-	0.9596089	-4.4	2.99	0.349	21900
	2	-	-	-	-	-	0.9596129	-3.9	2.9	0.335	176000
	3	-	-	-	-	-	-	-	-	-	-

$\dagger$ Calculated reference value k_{eff} is 0.9596524, obtained using an S_{10} HODD-2 method with *subm.* 4.

3.4 Concluding Remarks

In this chapter, we have reviewed the theory and methodology behind the HODD and DGFEM methods. While the latter is not a new method, it was novel for DRAGON5. As such, we have briefly detailed how it was implemented in the DRAGON5 code with regards to its sweep and algorithm. We also explained our reasoning behind our choice of basis functions. To verify our implementation, we tested our implementation on two 2D benchmarks, and found very good agreement between the two methods. Finally, we positively validated both methods against the Takeda Model 2 Case 2 benchmark.

CHAPTER 4   HEXAGONAL GEOMETRY IMPLEMENTATION

The motivation behind hexagonal geometries in Fast Neutron Reactors (FNRs) has already been outlined in Chap. 1. Therefore, here, we focus solely on methodologies and the implementation in DRAGON5. Indeed, a brief outline of various approaches is given, before turning to the choice that was made in DRAGON5 as well as some implementation details. Finally, similarly to the previous chapter, a number of verification and validation benchmark results are presented and analysed, for both HODD and DGFEM. We also take the opportunity to compare the two on this geometry.

4.1   Potential Avenues for Hexagonal Representation

Hexagonal geometries – and how to properly model them – have been of interest for some time in the reactor physics community. Indeed, quite often, in core simulations (*i.e.*, the second step in a two-level scheme), hexagonal assemblies are homogenised to reduce the required computational resources and time for simulations. That being said, it is known that in some cases (for *e.g.*, [16]), the flux within an assembly can vary quickly within the hexagonal plane, making one mesh element per hexagon too coarse.

Various methods and schemes have been devised. The simplest might just be to develop finite difference equations (similar to the Diamond Difference (DD) scheme equations in Sec. 3.1.1) along the three axes of symmetry of the hexagon.

These equations can be developed either along the faces or along the vertices, and one hexagonal element can even be submeshed into smaller overlapping hexagonal elements. This was developed and implemented by Yamasaki *et al.* [55]. While the the use of the DD method meant that the code was relatively rapid, it was also shown to present instabilities for development along the faces. Considering the fluxes at the vertices was much more stable and produced good results.

A somewhat more "exotic" method involves using specially-adapted high-order basis functions on the whole hexagonal element, such as Gout's Wachspress finite elements [56]. It is expected that these high-order polynomials, adapted to the six-sided polygon, will be faster than having to submesh the hexagonal element – all the while being able to capture enough of the rapidly-changing flux without having to give up too much of the accuracy of the solution. This is currently being implemented in an in-house prototype code at the Alternative Energies and Atomic Energy Commission (*French:* Commissariat à l'énergie atomique et aux énergies alternatives) (CEA) [56] and is showing promising results.

Another method is the subdivision of each hexagonal mesh into six equilateral triangles. This is extremely popular because a lot of finite element codes are already conventionally programmed to work with triangular elements. Hence, within such a paradigm, not many or even no additional tweaks need be made to the code to ensure it can accurately model these geometries. As an example, a couple of codes that do use this method are the CEA solver MINARET [20], and the Texas A&M University (TAMU) code XUTHUS [57].
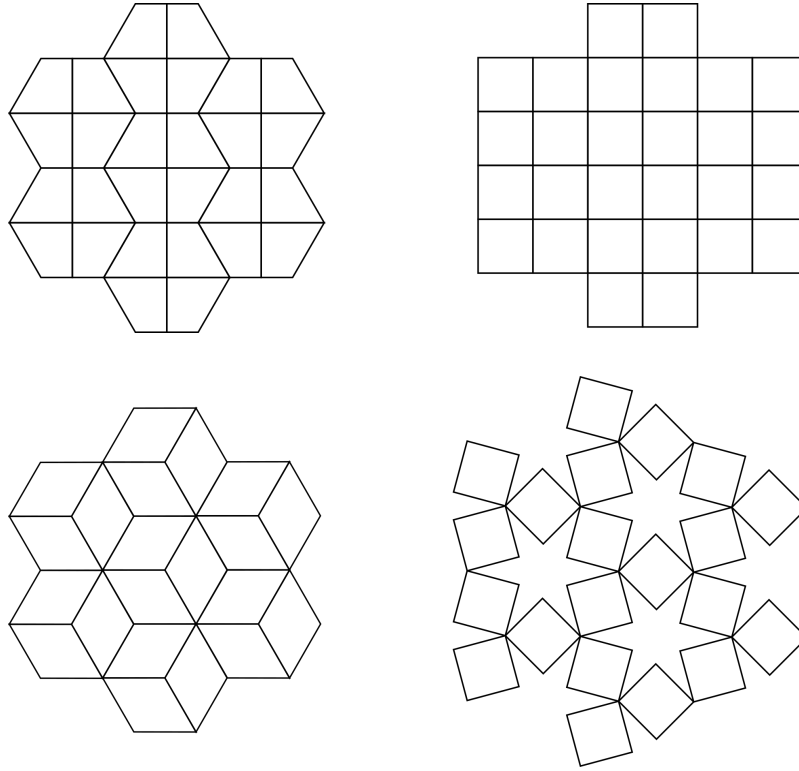
Figure 4.1 Two ways of splitting up the hexagon into quadrilaterals. Top: splitting into trapezia. Bottom: splitting into lozenges. The calculation geometry is shown on the left with the corresponding reference geometry given on the right. This specific arrangement of the reference elements (especially for the lozenges) was only chosen to facilitate with the visual representation.

It is also possible to cut up the hexagon into other shapes – beyond dividing into arbitrary shapes and using an unstructured mesh solver that is. Indeed, it is possible to partition the hexagon into either trapezia[1] or lozenges. These shapes can then be mapped onto the reference square element by using a transformation. These are both represented in Fig. 4.1. There are two ways of doing these transformations: a *conformal map* which conserves angles

---

[1]Note that the trapezium is generally termed 'trapezoid' in North American English and, in British English, both 'trapeziums' and 'trapezia' are accepted forms for the plural.

and shapes at the infinitesimal scale, but not the general size or curvature and an *affine transformation* which conserves lines and parallelism but not distances or angles.

Schneider [58] investigated these for the diffusion equation discretised using a dual-space finite element representation. As affine transformations, he considered the bilinear and Piola[2] mappings for trapezia and lozenges respectively, while for conformal transformations, he looked at the *Schwarz-Christoffel* transformation [59, 60] for both shapes. He found that affine transformations generally resulted in better results with a slight advantage for lozenges.

Lozenge-based submeshing was also quite successfully implemented for the discrete-ordinates transport equation in the CEA solver ERANOS [61] as well as some work by Valle and Mund [62]. The latter used a Gordon blending technique [63] for the transformation where a relationship is found between the reference element and each lozenge. This then allows for the direction cosines in the transport equation to be modified, as well as find scaled values for the material cross-section and source terms. Private communication with the first author of [61] allowed us to ascertain that they used a transformation of the differential operators and vectors similar to that outlined in [64].

4.2 Handling of the Hexagonal Geometry in DRAGON5

For this work, it was chosen to divide the hexagons into lozenges similar to the work done by Le Tellier *et al.* [61]. This method brings about a certain elegance and ease to the implementation. Indeed, as the High Order Diamond Difference (HODD) and Discontinuous Galerkin Finite Element Method (DGFEM) solvers had already been set up in Cartesian geometry, it was convenient to use this method as it made the code implementation much more convenient and straight-forward.
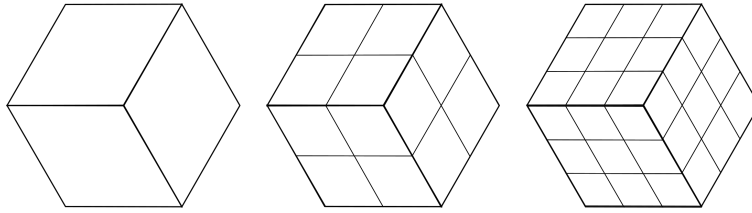
Figure 4.2 Lozenge submesh and its associated refinement in the hexagonal geometry.

As we have seen, the hexagon is first split into three lozenges and each individual lozenge can then be further sub-meshed if needed, as shown in Fig. 4.2. These lozenges can then

---

[2]Schneider [58] used the Piola transformation as he was using a dual-space finite element representation and he needed to ensure continuity of the current vector between lozenges. This is not present in the S_N method for the transport equation and hence, a simpler affine transformation is used – as we will see later.

be mapped onto reference square elements (denoted by $\hat{\xi}$) by using a transformation F_U, as shown in Fig. 4.3, where U denotes the lozenge under consideration. The associated Jacobian matrices, also shown in Fig. 4.3, can be obtained by finding

$$\mathbb{J}_{\hat{\xi},U} = \left(\frac{\partial x_i}{\partial u_i}\right)_{i,j},$$

having determinant $|\mathbb{J}_{\hat{\xi},U}|$. These matrices represent the shear and/or rotation that the reference element undergoes.
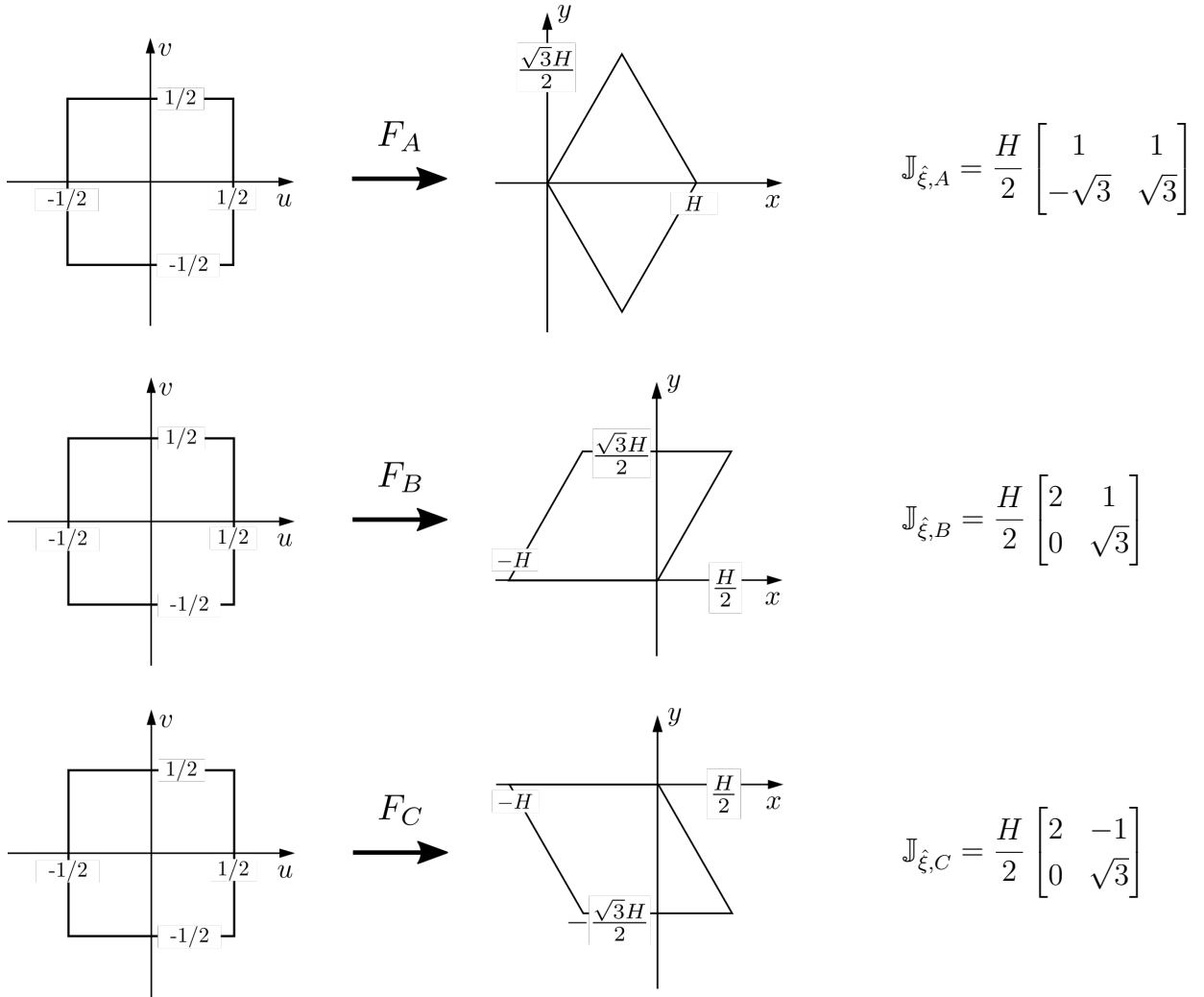
Figure 4.3 Affine transformations for each of the lozenges (top to bottom: A, B, and C) making up the hexagon, and the associated Jacobian matrices. The translation is not taken into account in the Jacobian and is not really important to the procedure. It is only shown to represent the relative position of the lozenges within a hexagon centred at the origin. H is the length of one side of the hexagon or the lozenge.

These transformations are all fairly straight-forward save for one thing. If we consider the left and right sides of the reference element to be the x sides, and the top and bottom the y sides, we can find corresponding x and y sides for the lozenges. There is no ambiguity for lozenges B and C, as it is only a shear transformation. If F_A is viewed as a $\pi/3$ anticlockwise rotation followed by a shear, then the North-West and South-East sides of the lozenge are the x sides, and the others y. This is all to point out that the x side of lozenge C is connected to the y side of lozenge A. This will be important later on.

We now begin with the DGFEM formulation of the transport equation obtained previously in Sec. 3.1.4, and reproduced below:

$$\int_{\hat{\xi}} \left(\boldsymbol{\Omega}_n \cdot \hat{\nabla}\psi_{n,h} + \Sigma\psi_{n,h} \right) w_h d\hat{s} = \int_{\hat{\xi}} Q_n w_h d\hat{s} + \int_{\partial\hat{\xi}_-} (\hat{\boldsymbol{n}} \cdot \boldsymbol{\Omega}_n) \, [\![\psi_{n,h}]\!] w_h^+ d\hat{l} \ , \qquad (4.1)$$

where the hat $(\hat{\cdot})$ notation is used to explicitly represent that it is defined on the reference element, $\hat{\xi}$. It is possible to have the same equation defined on the lozenge, $i.e.$,

$$\int_{U} \left(\boldsymbol{\Omega}_n \cdot \nabla\psi_{n,h} + \Sigma\psi_{n,h} \right) w_h ds = \int_{U} Q_n w_h ds + \int_{\partial U_-} (\boldsymbol{n} \cdot \boldsymbol{\Omega}_n) \, [\![\psi_{n,h}]\!] w_h^+ dl \ , \qquad (4.2)$$

where the changes are on the operators, vectors and integration limits. We can then find the equivalence between Eqn. 4.1 and Eqn. 4.2 by using the transformed operators and vectors,

$$\begin{aligned}
\nabla &= \mathbb{J}_{\hat{\xi},U}^{-T}\hat{\nabla} \\
ds &= |\mathbb{J}_{\hat{\xi},U}|d\hat{s} \\
dl &= \|\mathbb{J}_{\hat{\xi},U}\hat{\boldsymbol{t}}\|d\hat{l} \ ,
\end{aligned} \qquad (4.3)$$

and obtain,

$$\int_{\hat{\xi}} \left(\boldsymbol{\Omega}_n \cdot (\mathbb{J}_{\hat{\xi},U}^{-T}\hat{\nabla}\psi_{n,h}) + \Sigma\psi_{n,h} \right) w_h \ |\mathbb{J}_{\hat{\xi},U}|d\hat{s}$$
$$= \int_{\hat{\xi}} Q_n w_h \ |\mathbb{J}_{\hat{\xi},U}|d\hat{s} + \int_{\partial\hat{\xi}_-} (\boldsymbol{n} \cdot \boldsymbol{\Omega}_n)[\![\psi_{n,h}]\!] w_h^+ \ \|\mathbb{J}_{\hat{\xi},U}\hat{\boldsymbol{t}}\|d\hat{l} \ , \quad (4.4)$$

where $\|\cdot\|$ is the Euclidean norm in $\mathbb{R}^2$ and $\hat{\boldsymbol{t}}$ is the unit tangent vector to the inflow boundary, $\partial\hat{\xi}_-$, of the reference element. It should be noted that while $\hat{\boldsymbol{n}}$ is the unit normal to $\hat{\xi}$, $\boldsymbol{n}$ is the unit normal to the lozenge element, U.

Of additional particular interest here is that while the Jacobian matrix is different for each of the three lozenges, the determinant and $\|\mathbb{J}_{\hat{\xi},U}\hat{\boldsymbol{t}}\|$ are in fact the same for all of them; this

greatly helps with the implementation. They are given by

$$|\mathbb{J}_{\hat{\xi},U}| = \frac{\sqrt{3}H^2}{2}$$
$$\|\mathbb{J}_{K,U}\hat{\boldsymbol{t}}\| = H \ .$$

$$(4.5)$$

For a general Jacobian matrix, $\mathbb{J}_{\hat{\xi},U}$, whose elements are given by

$$\mathbb{J}_{\hat{\xi},U} = \begin{bmatrix} A & B \\ C & D \end{bmatrix} \ ,$$

if we make the x and y dependence explicit (as well as expand $\boldsymbol{\Omega}$ into μ and η), akin to Eqn. 3.36, we can see somewhat more clearly that the changes to the equation are not extensive:

$$
\begin{aligned}
&(\mu_n D - \eta_n B) \int_{\hat{\xi}} du \ dv \ \widetilde{P}_\alpha(u)\widetilde{P}_\beta(v) \ \frac{\partial}{\partial u}\psi_{n,i,j}(u,v)+ \\
&(-\mu_n C + \eta_n A) \int_{\hat{\xi}} du \ dv \ \widetilde{P}_\alpha(u)\widetilde{P}_\beta(v) \ \frac{\partial}{\partial v}\psi_{n,i,j}(u,v)+ \\
&\frac{\sqrt{3}H^2}{2} \int_{\hat{\xi}} du \ dv \ \widetilde{P}_\alpha(u)\widetilde{P}_\beta(v)\Sigma- \\
&(\mu_n D - \eta_n B) \int_{\xi_{r-}} du \ \widetilde{P}_\alpha(u) \left(\psi_{n,i,j}(u,\xi_{r-})|^+ - \psi_{n,i,j}(u,\xi_{r-})|^- \right) - \\
&(-\mu_n C + \eta_n A) \int_{\xi_{r-}} dv \ \widetilde{P}_\beta(v) \left(\psi_{n,i,j}(\xi_{r-},v)|^+ - \psi_{n,i,j}(\xi_{r-},v)|^- \right) = \\
&\frac{\sqrt{3}H^2}{2} \int_{\hat{\xi}} du \ dv \ \widetilde{P}_\alpha(u)\widetilde{P}_\beta(v)Q_{n,i,j}(u,v) \ ,
\end{aligned}
$$

$$(4.6)$$

where, effectively, we end up with modified direction cosines given by

$$
\begin{aligned}
\mu_n^H &= \mu_n D - \eta_n B \\
\eta_n^H &= -\mu_n C + \eta_n A \ .
\end{aligned}
$$

$$(4.7)$$

Similar equations can be derived in 3D hexagonal-z geometry using the same approach.

4.3   Solution Algorithm and Implementation Details

This section describes some of the differences and subtleties that have been to be taken into account when implementing the sweep in hexagonal geometry. To do so, we will be mostly contrasting with the 2D Cartesian sweep that we described in Chap. 3 as this greatly helps to illustrate the point.

4.3.1 Number of sweep directions

One of the biggest differences that one encounters almost immediately when starting to implement the sweep in hexagonal geometry is the number of sweep directions. Indeed, as we discussed previously in Sec. 3.2.4, for the 2D Cartesian geometry, there are *four* major sweep directions, as had been exemplified in Fig. 3.3. In hexagonal geometry, however, there are *six* major directions, as shown in Fig. 4.4.
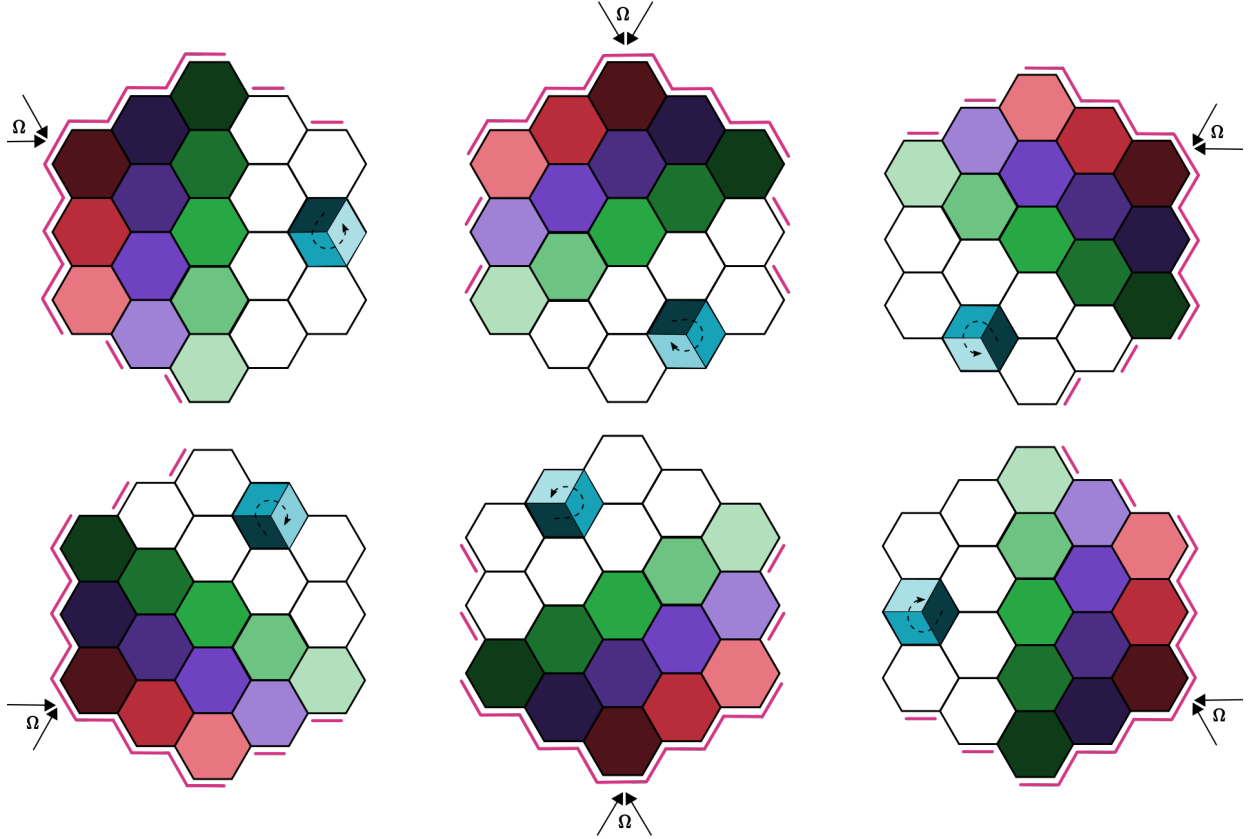
Figure 4.4 The six different ways that the hexagonal columnar sweep can proceed depending on the direction of neutron travel. The orientation of the domain is the same throughout. The incoming sides where the flux is assumed to be known at the beginning of each sweep is shown in pink. The sweep progression is then as follows: it starts with the darkest red colour and proceeds with the fading red colours; it then moves on to the next colour, from darkest to lightest and so on. The blue colours indicate the lozenge sweep within the hexagon, again from darkest to light; this is highlighted with the dashed arrow.

We show there how the sweep proceeds depending on the general direction, for a "columnar" style sweep.[3] The sweep starts with the dark red hexagon; it then proceeds from darkest

---

[3]It is also possible to have a wavefront-style sweep for hexagons but we will look at that in more detail

to lightest colour, moving from colour to colour. It should be noted that the sweep on the lozenges also changes depending on direction; it is represented in blue, again from darkest to lightest within each domain.

4.3.2 Outgoing fluxes

For the Cartesian algorithm, we mentioned how it was straightforward to sweep the geometry *implicitly.* Indeed, there, the columns are always of the same size, and there are two outgoing sides for each element with each side always connecting to the same relative columns.

With the hexagonal plane, it is slightly more complicated. The columns are of differing lengths. While there are consistently three outgoing sides for the hexagons, these only correspond to two columns, so one would have to be mindful of that. Additionally, while the hexagons themselves consistently have three outgoing sides, the solver deals with the lozenges, so these would have to correspond correctly.

Hence, instead of trying to find an implicit way through appropriate loops and arrays to sweep through the domain, we decided to compute a *connectivity matrix* early on, for each direction. This was similar to a graph in that it allowed us to know the following: for each of the six cardinal sweeping direction, each side of each element was connected to the side of another element – which side (x or y), which lozenge (A, B or C) and which hexagon (within the numbering system of the `GEO:` module of DRAGON5) is it? This was implemented in a new subroutine, `SNGRPH`, within the S_N tracking module, `SNT:`.

4.3.3 Outgoing fluxes between lozenges A and C

Passing the outgoing flux from lozenge C to lozenge A (or vice-versa) is a very important aspect. Indeed, as mentioned previously in Sec. 4.2, the x side of lozenge C connects to the y side of lozenge A. This is shown in Fig. 4.5 where the relevant edges have been highlighted in red and green, for both the lozenges and the corresponding reference elements.

Now consider the reference frame of the reference elements where dummy boundary fluxes have been drawn on the red edge. This is shown in Fig. 4.6 where a three-dimensional representation has been used – the u and v axes on the xy-plane and ψ on the z-axis. We can see that when the outgoing flux of the x side is passed from lozenge C to A, the red flux that was sloping upwards with increasing v becomes a green flux that now slopes downwards with increasing u. Simply put, the gradient has been reversed. Hence, for odd Legendre moments of the boundary fluxes, one should be exceptionally careful to make this adjustment.

---

in Chap. 6.
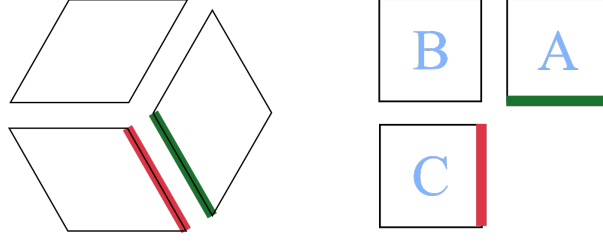
Figure 4.5 Highlighted edges, on lozenges and corresponding reference elements, where the x side of lozenge C connects to the y side of lozenge A.
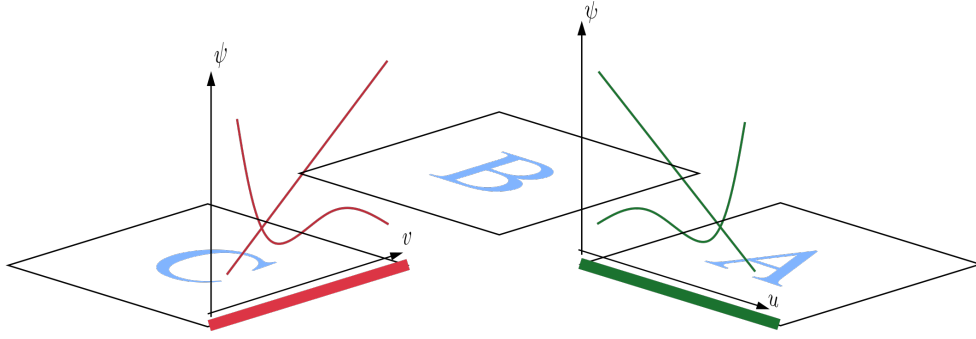
Figure 4.6 This image should be interpreted in conjunction with Fig. 4.5. A 3D representation is used: the xy-plane correspond to the u- and v-axes while the z-axis is the flux, ψ. The corresponding reference elements of all three lozenges are drawn to help with the visual orientation. This image shows how dummy fluxes represented on the x side of lozenge C are mirrored onto the y side of lozenge A such that the gradients are reversed.

4.3.4 Overall algorithm

Having discussed the previous points, the overall algorithm is given in Alg. 2. As with the Cartesian 2D case, while quite detailed, it is not exhaustive. This is done in an effort to keep it readable. We wish to highlight the following points:

- As this geometry was implemented with only vacuum boundary conditions, the outgoing fluxes on the domain boundaries are not of consequence.

- As inputs are the matrices `HEXSWP` and `LOZSWP` given the sweep within the domain for different directions; this corresponds to Fig. 4.4. There is also `CONNEC` which indicates to which side of which lozenge of which hexagon, each side of each lozenge of hexagon is connected to, using the numbering system of the `GEO:` module in DRAGON5.

- $\varphi_{n,BC}^{[\alpha]}$ and $\varphi_{n,BC}^{[\beta]}$ correspond to the outgoing boundary fluxes in between the hexagons.

- There are now four loops to sweep over the whole domain: loops over hexagons, over

lozenges and over the x and y submeshes of the lozenges.

4.4 Numerical Results

Three benchmarks will be presented in this section: a one-group anisotropic 2D simple benchmark (2D-SNA), a one-group anisotropic 3D simple benchmark (3D-SNA) and a four-group 3D small FNR (3D-TAK4). The first two were intended for verification and validation, while the last one was drawn from the Takeda benchmarks [16] to really test the solver.

Similarly to the previous chapter, the same parameters are investigated and the results presented are k_{eff} values, errors on the absorption rate distribution, and the computational times. The overall discussion will, however, be done after all the results are presented.

No acceleration methods were used yet for all calculations in this section and a Level Symmetric (LS) quadrature was used in all computations. While the latter is not necessarily optional for the hexagonal geometry (for *e.g.*, not rotationally invariant on the hexagonal plane with respect to $\pi/3$ increments), it was expected that it would do fine in cases where the whole geometry is considered. Finally, the convergence criterion was again 10^{-5}.

4.4.1 One-group benchmarks: 2D-SNA and 3D-SNA

Having implemented the methods described in the previous two sections, we verified and validated the solver with each of a 2D and 3D one-group benchmark. While not realistic, these were devised by Hébert [65] in an effort to sharply increase the transport and anisotropic effects. Also available in the same paper were the k_{eff} values from another discrete-ordinates solver, SNATCH. This would hence allow for a larger verification of the solver.

SNATCH [61] is a DGFEM-based S_N solver, with a lozenge-based submeshing too. However, it features different basis functions which are hierarchical and based on both Legendre and Lagrange basis functions. Also, it uses a product quadrature, with n directions on the azimuthal plane and m directions on the polar half plane. This is represented as $HQ_{n,m}$ giving a total of $12 \times n \times m$ directions on the unit sphere. This means that while both $HQ_{1,2}$ and S_4 have 24 directions over the unit sphere, $HQ_{4,5}$ has 240 directions and S_{14} has 224.

The 2D benchmark, 2D-SNA, is described in Fig. 4.7. The cross-sections are the same as for the 2D-CNS benchmark – see Tab. 3.3. The SNATCH reference results are given in Tab. 4.1 while the results obtained from DRAGON5 are in Tab. 4.2. We did not have access to the raw data from the SNATCH computations so each set of calculations (from SNATCH and DRAGON5) used their own reference calculation for the computation of the Δk_{eff}, ϵ_{max} and $\bar{\epsilon}$. These are clearly marked in the tables.

Algorithm 2: Representation of the sweep for the 2D hexagonal case.

input : $Q_{l,i,j}^{m,[\alpha,\beta]}$, `HEXSWP`($N_{\text{hex}}$,6), `LOZSWP`(3,6), `CONNEC`(3, $N_{\text{hex}} \times 2$,6)

output: $\phi_{l,i,j}^{m,[\alpha,\beta]}$

local : $\varphi_{n,BC}^{[\alpha]}$, $\varphi_{n,BC}^{[\beta]}$, `JAC`(4,3)

```
/* Initialise flux.                                                      */
```
1 $\phi_{l,i,j}^{m,[\alpha,\beta]} = 0.0$

```
/* Loop over directions in unit sphere.                                  */
```
2 **for** $n = 1$ **to** $N(N+2)/2$ **do**

3   Determine cardinal direction, i_{DIR}.

```
    /* Loop over hexagons in domain.                                     */
```
4   **for** $j_H = 1$ **to** N_{hex} **do**   `/* `N_{hex}` = # of hexagons */`

5         $i_H = $ `HEXSWP`(j_H,i_{DIR})

```
        /* Loop over lozenges within each hexagon.                       */
```
6   **for** $j_L = 1$ **to** 3 **do**

7             $i_L = $ `LOZSWP`(j_L,i_{DIR})

8   Compute μ_n^H and η_n^H using `JAC`(4,i_L).

```
            /* Loop over submeshes within each lozenge.                  */
```
9   **for** $i = 1$ **to** m_x **do**   `/* `m_x` = # of submesh/lozenge   */`

10                 $\varphi_{n,i-}^{[\beta]} \leftarrow \varphi_{n,BC}^{[\beta]}/\varphi_{n,i+}^{[\beta]}$

11   **for** $j = 1$ **to** m_y **do**   `/* `$m_y = m_x$` */`

12                     $\varphi_{n,j-}^{[\alpha]} \leftarrow \varphi_{n,BC}^{[\alpha]}/\varphi_{n,j+}^{[\alpha]}$

13   Solve system of equations $\mathbb{T}\Psi_{n,i,j} = \mathbb{Q}_{n,i,j}$.

14   Compute the outgoing fluxes, $\varphi_{n,i+}^{[\beta]}$ and $\varphi_{n,j+}^{[\alpha]}$.

15   **if** $i_L = 1$ *or* $i_L = 3$ **then**

16   Multiply odd Legendre moments by -1.

17   Sum angular fluxes over directions to obtain scalar flux such that

18                     $\phi_{l,i,j}^{m,[\alpha,\beta]} = \phi_{l,i,j}^{m,[\alpha,\beta]} + 2\omega_n\psi_{l,i,j}^{m,[\alpha,\beta]}$.

```
                /* Store outgoing fluxes on lozenge y-side.              */
```
19   **if** $j == m_y$ **then**

20   Store $\varphi_{n,BC}^{[\alpha]}$ from $\varphi_{n,j+}^{[\alpha]}$ using `CONNEC`.

```
            /* Store outgoing fluxes on lozenge x-side.                  */
```
21   **if** $i == m_x$ **then**

22   Store $\varphi_{n,BC}^{[\beta]}$ from $\varphi_{n,j+}^{[\beta]}$ using `CONNEC`.
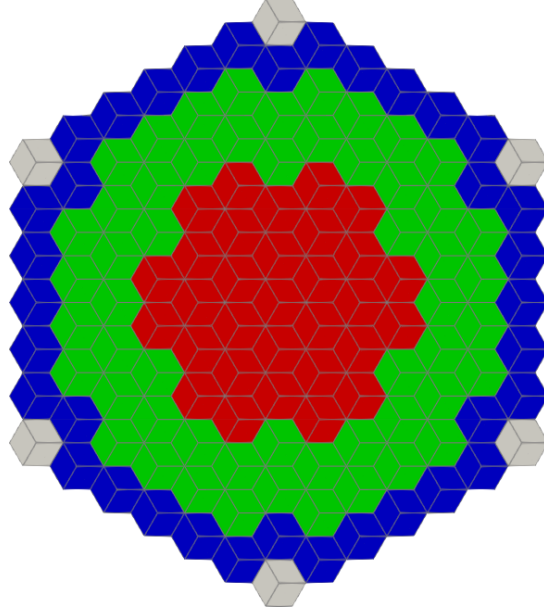
Figure 4.7 Description of the one-group hexagonal 2D benchmark, 2D-SNA. Each hexagon side is 19 cm. Vacuum boundary conditions are applied on the whole edge of the domain. See Table 3.3 for the cross-section data. Red corresponds to mixture 1, green to 2, blue to 3, and grey indicates a void. The initial un-refined computational mesh is also shown.

Table 4.1 Quoted literature values for 2D-SNA using SNATCH, reproduced from [65].

$HQ_{n,m}$	$O.$	$subm.$	k_{eff}	Δk_{eff} (pcm)	ϵ_{max} (%)	$\bar{\epsilon}$ (%)
$HQ_{1,2}$	1	1	1.000606	-170.7	5.11	0.98
		2	1.000898	-141.5	3.76	0.94
		3	1.000960	-135.3	3.61	0.95
	2	1	1.000969	-134.4	3.44	0.94
		2	1.000999	-131.4	3.57	0.95
		3	1.001005	-130.8	3.61	0.95
$HQ_{4,5}$	1	1	1.002094	-21.9	1.67	0.43
		2	1.002281	-3.2	0.28	0.07
		3	1.002304	-0.9	0.09	0.03
	2	1	1.002313	0.1	0.08	0.02
		2	1.002314	0.1	0.00	0.00
		3	1.002313	SNATCH ref.		

Table 4.2 Summarised results for the 2D-SNA benchmark. Λ denotes the polynomial order, and *subm.* the submeshing. Three significant figures are given, except for k_{eff}.

			DD					DG				
S_N	Λ	*subm.*	k_{eff}	Δk_{eff} (pcm)	$\epsilon_{\max}$ (%)	$\bar{\epsilon}$ (%)	Time (s)	k_{eff}	Δk_{eff} (pcm)	$\epsilon_{\max}$ (%)	$\bar{\epsilon}$ (%)	Time (s)
4	0	1	1.001075	-125	7.74	1.71	0.053	-	-	-	-	-
		2	1.001898	-43.1	3.83	0.679	0.185	-	-	-	-	-
		3	1.002044	-28.5	3.3	0.546	0.371	-	-	-	-	-
	1	1	1.002175	-15.4	3.81	0.664	0.138	1.001958	-37.1	3.26	0.667	0.13
		2	1.002176	-15.2	2.58	0.432	0.513	1.002143	-18.5	3.05	0.458	0.521
		3	1.002174	-15.5	2.95	0.464	1.16	1.002165	-16.4	3.01	0.463	1.16
	2	1	1.002168	-16.1	2.57	0.444	0.441	1.002175	-15.4	3.13	0.474	0.426
		2	1.002175	-15.4	2.95	0.465	1.64	1.002175	-15.4	2.94	0.463	1.69
		3	1.002176	-15.3	2.91	0.461	3.76	1.002175	-15.3	2.92	0.46	3.75
	3	1	-	-	-	-	-	1.002174	-15.5	2.93	0.464	1.28
		2	-	-	-	-	-	1.002175	-15.4	2.91	0.46	5.24
		3	-	-	-	-	-	1.002176	-15.3	2.91	0.46	11.8
14	0	1	1.001212	-112	5.85	1.33	0.458	-	-	-	-	-
		2	1.002054	-27.4	0.878	0.273	1.54	-	-	-	-	-
		3	1.002202	-12.7	0.497	0.133	3.19	-	-	-	-	-
	1	1	1.002330	0.1	1.18	0.333	1.17	1.002106	-22.3	1.68	0.458	1.28
		2	1.002330	0.1	0.0628	0.0153	4.72	1.002293	-3.6	0.297	0.0839	4.57
		3	1.002329	0	0.0212	0.00333	10.1	1.002317	-1.2	0.104	0.0291	10.1
	2	1	1.002325	-0.4	0.151	0.0199	3.72	1.002326	-0.2	0.132	0.029	3.84
		2	1.002329	0	0.028	0.00282	14.6	1.002328	-0.1	0.0208	0.00281	14.5
		3	1.002329	0	0.0126	0.00114	33.1	1.002329	**DRAGON5 ref.**			32.8
	3	1	-	-	-	-	-	1.002327	-0.2	0.0341	0.00677	12
		2	-	-	-	-	-	1.002329	0	0.00653	0.000609	47.6
		3	-	-	-	-	-	1.002329	0	0.0126	0.00102	105

Comparing the results from DRAGON5 with those of SNATCH, it can be seen that when the results have converged (both angularly and spatially), there is only a difference of 1.6 pcm between the two – which is an excellent agreement between the two solvers. Moreover, the results from DRAGON5 are able to approach the reference k_{eff} much faster, even at S_4.

In a similar fashion, the domain for 3D benchmark, 3D-SNA, is given in Fig. 4.8, and the cross sections in Tab. 3.3. This test case is made of 3 distinctive layers with the $0 \leq z < 50$ cm layer being exactly the same as the 2D-SNA benchmark. A reflective boundary condition is

applied at $z = 0$ cm while vacuum is present on the rest. Only layer 1 has its submeshing further refined when the *subm.* parameter is increased; layers 2 and 3 are left as is. The literature values of SNATCH are given in Tab. 4.3 while the results from DRAGON5 are summarised in Tab. 4.4. As with the 2D-SNA benchmark, each set of calculation has its own reference calculation clearly marked in the tables.

The results for the 3D-SNA benchmark from DRAGON5, on their own, are very similar to what we have seen so far, both for the Cartesian (2D and 3D) and hexagonal 2D cases: very good agreement between HODD and DGFEM but with DGFEM struggling a bit more with less refined meshes. However, this time, there is a much larger difference between the two solvers – about 430 pcm. While this might have indicated an issue with our solver, we decided to test with a benchmark that had quoted literature Monte Carlo values.

(a) Layer 1: $0 \leq z < 50$ cm   (b) Layer 2: $50 \leq z < 60$ cm   (c) Layer 3: $60 \leq z < 70$ cm
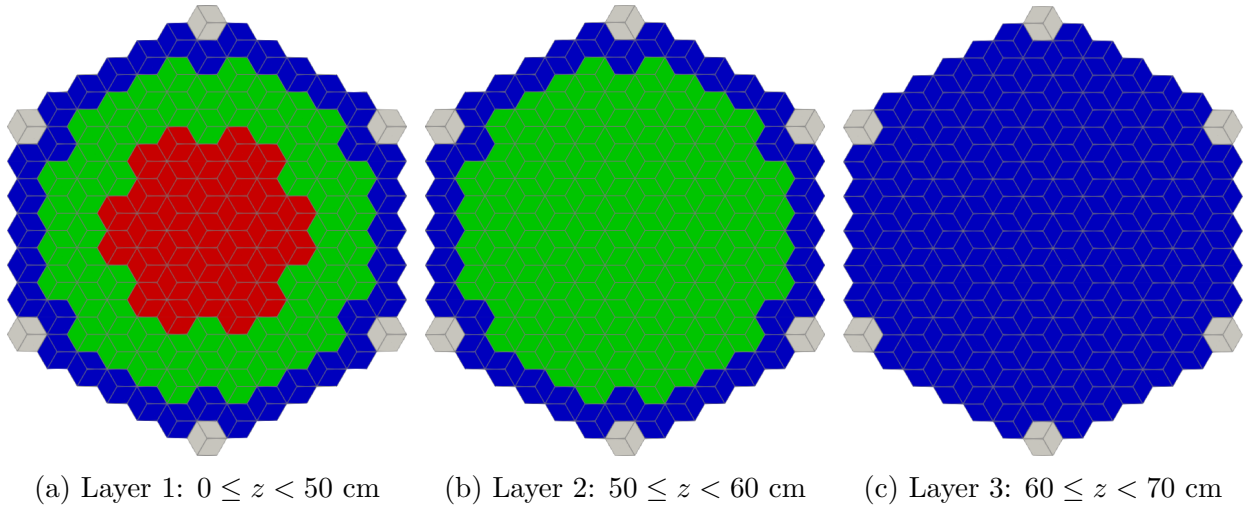
Figure 4.8 Description of the simple one-group hexagonal 3D benchmark, 3D-SNA. Each hexagon side is 19 cm. Reflective boundary conditions are applied at the plane $z = 0$ cm and vacuum is applied to the rest of the domain. See Table 3.3 for the cross-section data. Red corresponds to mixture 1, green to 2, and blue to 3. The grey region indicates a void. The initial un-submeshed computational mesh is also shown.

Table 4.3 Quoted literature values for 3D-SNA using SNATCH, reproduced from [65].

$HQ_{n,m}$	$O.$	$subm.$	k_{eff}	Δk_{eff} (pcm)	ϵ_{max} (%)	$\bar{\epsilon}$ (%)
$HQ_{1,2}$	1	1	0.752581	-139.0	11.01	2.42
		2	0.754311	34.0	11.72	2.44
		3	0.754807	83.6	11.79	2.53
	2	1	0.754117	14.6	12.24	2.57
		2	0.754603	63.2	12.02	2.48
		3	0.754983	101.2	11.88	2.57
$HQ_{4,5}$	1	1	0.752114	-185.7	2.65	0.66
		2	0.753531	-44.0	0.68	0.13
		3	0.753890	-8.1	0.08	0.03
	2	1	0.753404	-56.7	1.88	0.37
		2	0.753746	-22.5	0.61	0.10
		3	0.753971	**SNATCH ref.**		

Table 4.4 Summarised results for the 3D-SNA benchmark. Λ denotes the polynomial order, and *subm.* the submeshing. Three significant figures are given, except for k_{eff}.

			DD					DG				
S_N	Λ	*subm.*	k_{eff}	Δk_{eff} (pcm)	$\epsilon_{\max}$ (%)	$\bar{\epsilon}$ (%)	Time (s)	k_{eff}	Δk_{eff} (pcm)	$\epsilon_{\max}$ (%)	$\bar{\epsilon}$ (%)	Time (s)
4	0	1	0.7510752	-724	31.2	6.86	0.547	-	-	-	-	-
		2	0.7556428	-267	26.1	5.99	3.01	-	-	-	-	-
		3	0.7564842	-183	25.2	5.87	9.35	-	-	-	-	-
	1	1	0.7569891	-133	26.2	3.11	3.4	0.7561832	-213	22.9	3.63	3.37
		2	0.7570121	-130	25.3	3.05	21.2	0.7568968	-142	23.4	3.57	20.9
		3	0.7570118	-130	25.4	3.05	67.1	0.7569786	-134	23.7	3.56	66.9
	2	1	0.7570086	-131	24.7	3.15	44.3	0.7570125	-130	24.7	3.14	43.3
		2	0.7570123	-130	24.6	3.15	273	0.7570124	-130	24.7	3.14	271
		3	0.7570131	-130	24.5	3.15	840	0.7570125	-130	24.7	3.14	825
	3	1	-	-	-	-	-	0.7570118	-130	24.7	3.15	426
		2	-	-	-	-	-	0.7570125	-130	24.6	3.15	2600
		3	-	-	-	-	-	0.7570129	-130	24.6	3.15	7790
14	0	1	0.7525744	-574	11.8	3.49	5.07	-	-	-	-	-
		2	0.7571055	-121	9.28	2.9	27.2	-	-	-	-	-
		3	0.7579356	-38.1	8.96	2.85	86.1	-	-	-	-	-
	1	1	0.7582908	-2.6	1.04	0.402	31.7	0.7575588	-75.8	2.04	0.731	30.5
		2	0.7583100	-0.6	0.725	0.229	189	0.7582338	-8.2	1.58	0.501	185
		3	0.7583105	-0.6	0.725	0.229	626	0.7583118	-0.4	1.59	0.503	628
	2	1	0.7583149	-0.1	0.488	0.0939	415	0.7583157	-0.1	0.13	0.0233	400
		2	0.7583168	0.1	0.386	0.0862	2600	0.7583160	0	0.0206	0.00268	2470
		3	0.7583167	0	0.373	0.0862	8010	0.7583163	**DRAGON5 ref.**			7630
	3	1	-	-	-	-	-	0.7583162	0	0.219	0.0557	5540
		2	-	-	-	-	-	-	-	-	-	-
		3	-	-	-	-	-	-	-	-	-	-

4.4.2 Four-group 3D small FNR core, Takeda Model 4: 3D-TAK4

This subsection presents the results for the Takeda Model 4 benchmark [16] based on the KNK-II fast reactor core. There are 3 cases of this model which place the control rod at different points: completely out, half in and completely in. We chose to model and present the results for the control rod completely in case (commonly referred to as 'Case 3') as this

is the one that can present the most difficulty for solvers to accurately portray.

That being said, in and of itself, this benchmark can be notoriously difficult to model. It is not a pure FNR as it also contains regions with moderator, meaning there are thermal effects as well. The thermal and fast fluxes both need to be properly resolved. The quoted literature values [16] are reproduced in Tab. 4.5.

Table 4.5 Quoted literature values for various methods for the 3D-TAK4 benchmark, reproduced from Takeda and Ikeda [16].

Method	k_{eff}	precision
Monte Carlo	0.8799	± 0.0003
S_8	0.8927	± 0.0110
P_7	0.8819	± 0.0100

It can be seen that there is quite the discrepancy between the Monte Carlo and the S_8 results – about 1280 pcm worth in fact. Takeda and Ikeda [16] note, in fact, that this "is because the flux varies rapidly with positions in the hexagonal plane, making one mesh per hexagon too coarse to be accurate" for the S_N solver. The paper states that there were two S_N solvers used to compute this benchmark: one was a DD solver Bando *et al.* [66] (possibly on the whole hexagon) and the other is just cited as "private communication" but it could be safe to assume it was also DD considering the codes available at the time. We, therefore, expected that with the lozenge submeshing and high-order polynomials, the results should be much better.

The domain is drawn in Fig. 4.9 with a description of the various regions in Tab. 4.6. Also, while the benchmark uses 36 meshes axially, we went with 38 as it came about naturally from the subdivision of our layers when inputting the data. These axial meshes do match up pretty well though and we do not expect it to make much of a difference. When refining the mesh, only the hexagonal plane mesh was subdivided; *i.e.*, the axial mesh remained constant throughout. Considering the size of the problem, we did not run a high S_N order (> 12) calculation as reference, choosing instead to focus on the k_{eff} values. Some of the calculations run took nearly a week to complete and the compute time available on the Digital Research Alliance of Canada (DRAC) clusters was indeed limited to a week.

The results are summarised in Tab. 4.7. It can be observed that as the calculation is properly resolved both angularly and spatially (for *e.g.*, at S_{10} DD/DG-1 *subm*.3), the results agree very well with the Monte Carlo reference value – around 50 pcm for DG and a mere 2 pcm for DD. This verifies and validates our hexagonal solver in DRAGON5.
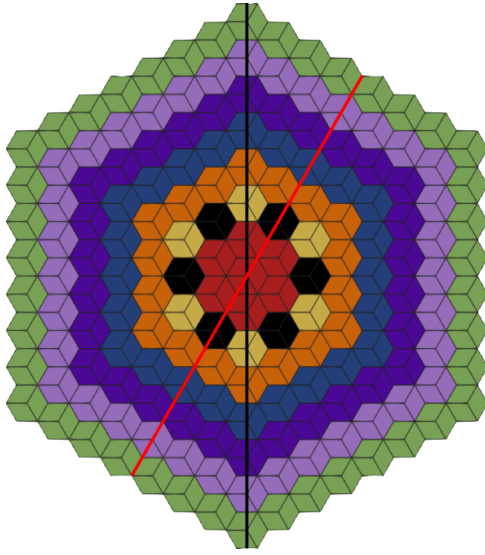
(a) Cut-through top view at $z = 95$ cm.
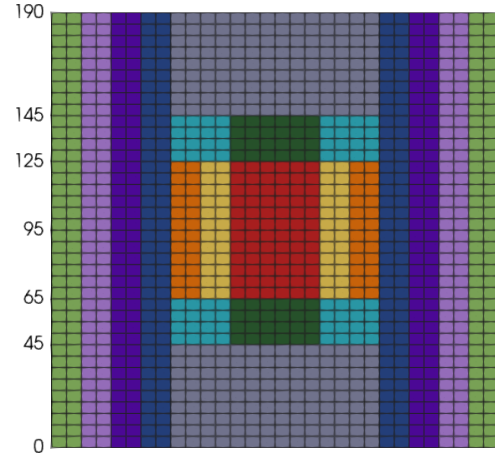
(b) Cut-through side view 1: through black line.
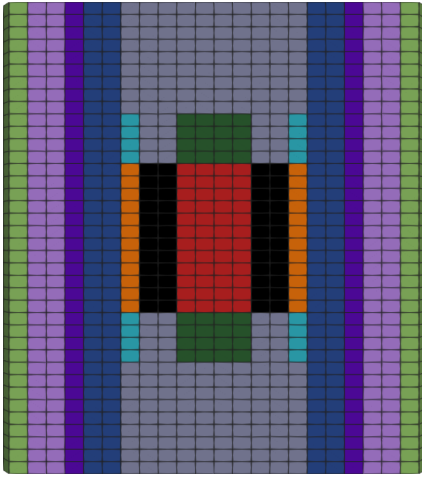
(c) Cut-through side view 2: through red line.
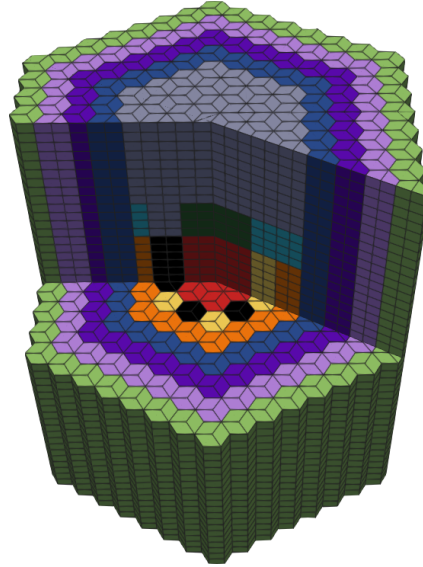
(d) Overall view with part removed.

Figure 4.9 Domain of the 3D-TAK4 benchmark at various angles. Dimensions are in cm; side of one hexagon is 7.5 cm. Vacuum boundary conditions are applied to the whole outer edge of the domain. The initial computational mesh before subsequent mesh refinement is shown in all subfigures. Refer to Tab. 4.6 for a description of the materials.

Table 4.6 Description of the colour representation in the 3D-TAK4 benchmark (see Fig. 4.9).

Region	Description
	Steel
	Axial blanket
	Axial reflector
	Test zone
	Driver without moderator
	Driver with moderator
	Reflector without moderator
	Reflector with moderator
	KNK-1 reflector
	Sodium steel zone
	Control rod

Surface plots of the scalar flux are also given in Fig. 4.10 as well as error surface plots for DG and DD in Fig. 4.11. The surface plots demonstrates how the flux indeed varies very rapidly even within one hexagonal element. Even if it is only the average fluxes that are plotted in each element (the flux is not actually reconstructed using all the degrees of freedoms (d.o.f.s)), Fig. 4.10 shows clearly how much smoother the flux is when using a *subm.* of 4.

Finally, the error surface plots in Fig. 4.11 brings an idea of where DD and DG perform differently. This was plotted for the linear order at S_{10}. So, while DD was within a $\tilde{1}00$ pcm of the reference k_{eff}, DG was more than 650 pcm away. Comparing with Fig. 4.10, we find that the largest differences tend to be where the flux is actually average. For the faster groups, these tend to be the control rods and the driver-without-moderator region while for the slower groups, there are the driver-with-moderator and reflectors regions. Still, even the largest differences are only about 1% or less of the actual flux.

4.5   Concluding Remarks

The S_N solver was expanded to hexagonal geometry in 2D and 3D for both the HODD and DGFEM methods. The theory was outlined as well as the more subtle points that we think one should be aware of during implementation. A pseudo-code algorithm was given to summarise everything. Benchmark results were then presented. These included test cases with results from another reliable solver from a national research laboratory in France, and a well-known test case from the literature. The results obtained with DRAGON5 were very much in line with the quoted values, allowing us to validate our solver. Again, we found that HODD performed comparably, if not better than DGFEM most of the time.

(a) S_{10} DG-1 $subm$.1

(b) S_{10} DG-1 $subm$.4
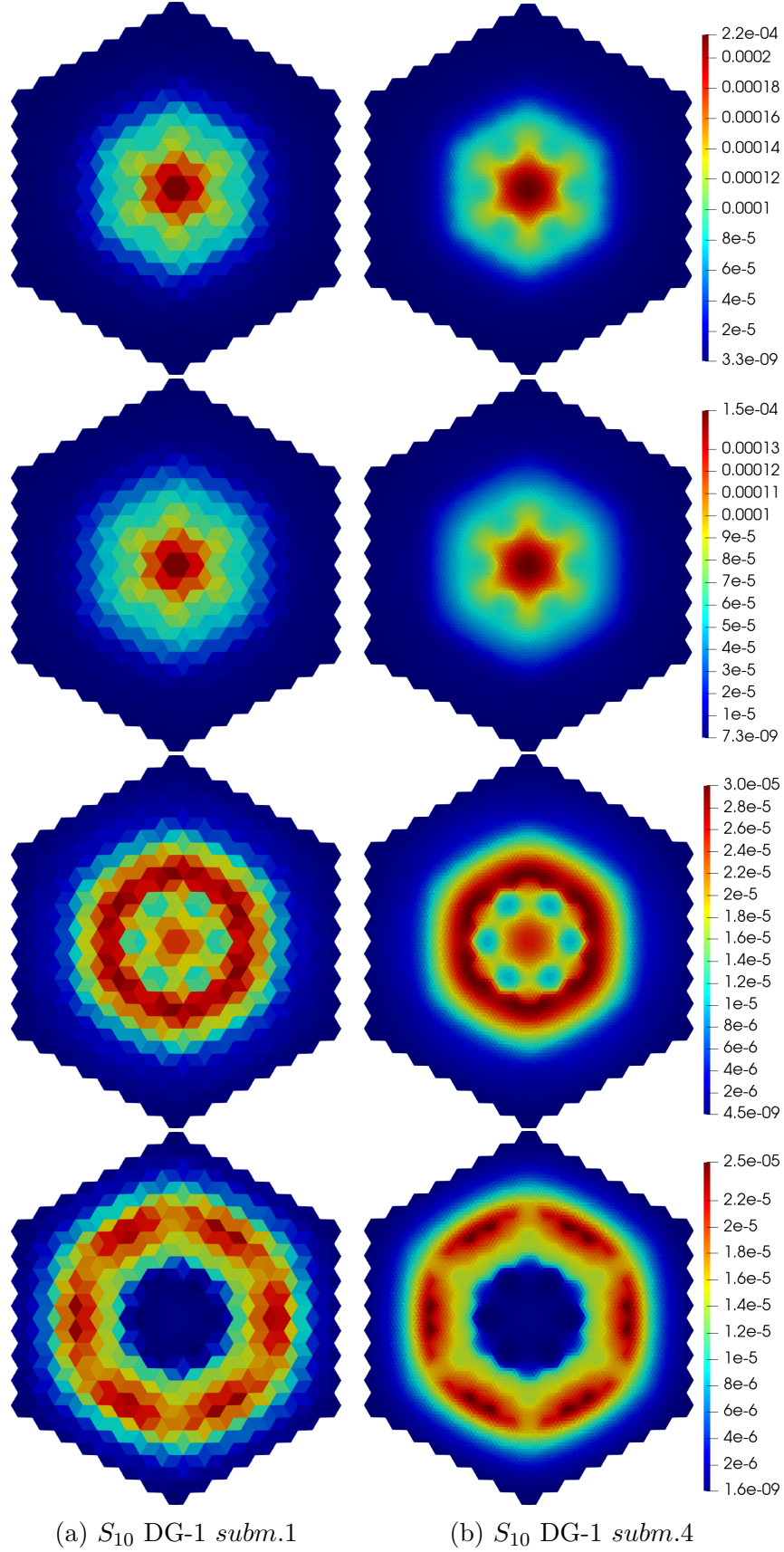
Figure 4.10 3D-TAK4 benchmark: scalar flux surface plots for energy groups 1 to 4, with group 1 at the top. These contrast the unrefined and $subm$. 4 meshes. The view and orientation are the same as Fig. 4.9a.
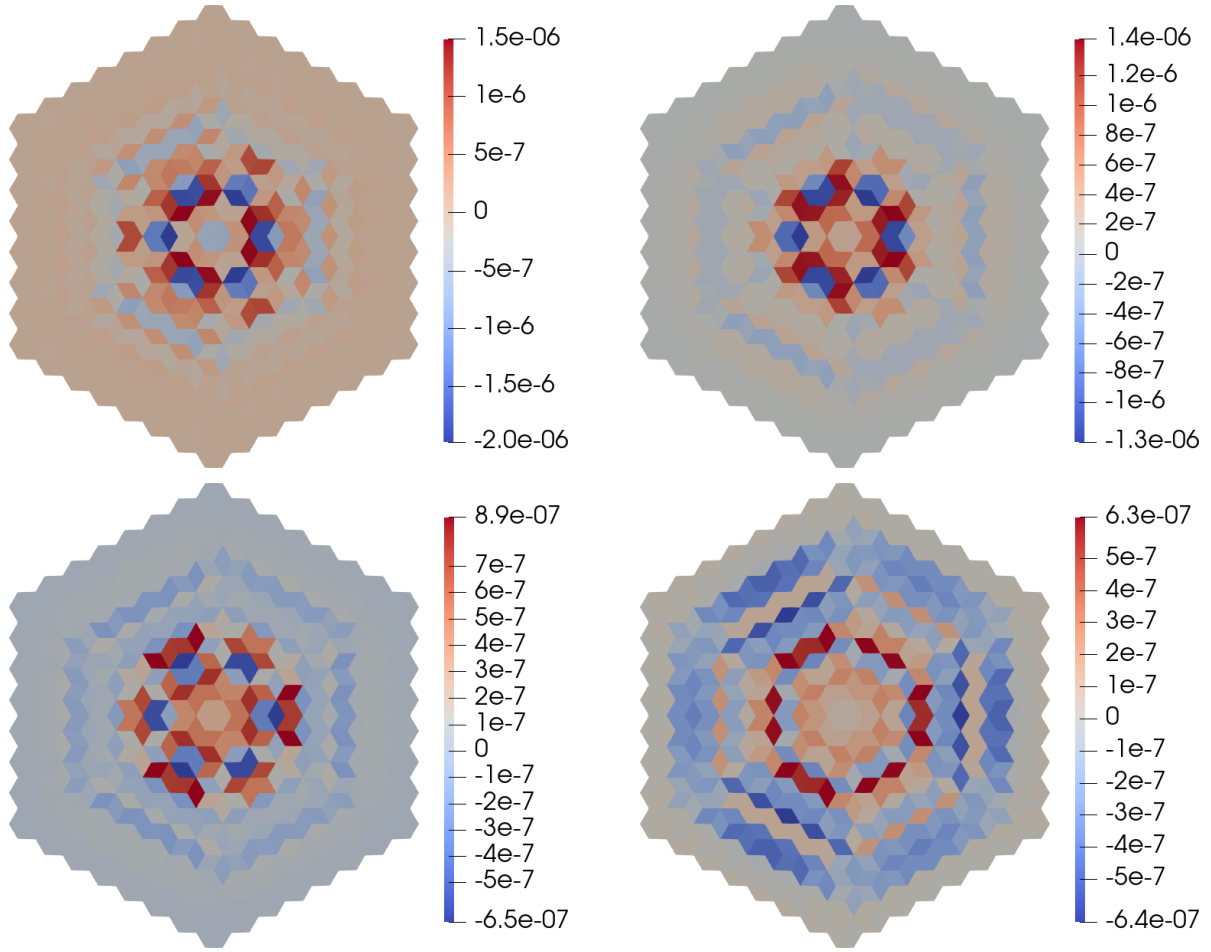
Figure 4.11 3D-TAK4 benchmark: surface plots of the difference between the two scalar flux maps of HODD and DGFEM for S_{10} *subm*.1, for the four energy groups. From left to right, top to bottom: group 1, group 2, group 3 and group 4.

Table 4.7 Summarised results for the 3D-TAK4 benchmark. Λ denotes the polynomial order, and *subm.* the submeshing. Three significant figures are given, except for k_{eff}.

S_N	Λ	*subm.*	DD k_{eff}	DD Δk_{eff} (pcm)	DD Time (s)	DG k_{eff}	DG Δk_{eff} (pcm)	DG Time (s)
2	0	1	0.8634277	1650	61.1	-	-	-
		2	0.8737065	619	249	-	-	-
		3	0.8750459	485	580	-	-	-
	1	1	0.8777618	214	421	0.8694615	1040	434
		2	0.8767351	316	1760	0.8751626	474	1810
		3	0.8766908	321	4010	0.8760252	387	4100
	2	1	0.8766767	322	6460	0.8764135	349	6220
		2	0.8766865	321	25100	0.8766070	329	24700
		3	0.8766868	321	57000	0.8766532	325	55400
	3	1	-	-	-	0.8765851	331	62900
		2	-	-	-	0.8766687	323	251000
		3	-	-	-	0.8766788	322	567000
6	0	1	0.8665104	1340	339	-	-	-
		2	0.8767435	316	1430	-	-	-
		3	0.8780870	181	2640	-	-	-
	1	1	0.8807698	-87	1630	0.8728631	704	1650
		2	0.8797427	15.7	7050	0.8783911	151	7270
		3	0.8797078	19.2	16400	0.8791765	72.3	17400
	2	1	0.8796850	21.5	37300	0.8795780	32.2	36300
		2	0.8797044	19.6	146000	0.8796658	23.4	142000
		3	0.8797033	19.7	326000	0.8796878	21.2	316000
	3	1	-	-	-	0.8796571	24.3	371000
10	0	1	0.8667960	1310	840	-	-	-
		2	0.8769145	299	3250	-	-	-
		3	0.8782655	163	8080	-	-	-
	1	1	0.8809062	-101	6500	0.8731214	678	5750
		2	0.8799127	-1.27	22500	0.8785824	132	23000
		3	0.8798766	2.34	55600	0.8793559	54.4	54800
	2	1	0.8798544	4.56	93600	0.8797545	14.6	91400
		2	0.8798733	2.67	366000	0.8798349	6.51	352000

CHAPTER 5   SYNTHETIC ACCELERATION (SA)

This chapter is the first way in which an attempt is made at trying to reduce the calculation time. Synthetic Accelerations (SAs) have long been used in the industry and have proven to be effective. However, they are seldom stable for all cases while at the same time being easy of implementation. Here, we look at a different form of synthetic acceleration as applied to HODD and DGFEM.

We begin by defining and exemplifying the idea of synthetic acceleration before moving on to the theory behind it. We then present an overview of the Diffusion Synthetic Acceleration (DSA), which is perhaps the most well-known and used SA. A synopsis of the Fourier Analysis (FA) method is also given. The implementation within the DRAGON5 code is then outlined, followed by Fourier analyses for the acceleration method as well as how tweaking some parameters of said method affected the resolution speed. Finally, the method is applied to some of the benchmarks seen thus far to see how much time can be gained.

5.1   Introduction

As mentioned previously, in Sec. 2.2, solving the neutron transport equation involves the well-established iterative scheme known as the *Source Iteration (SI)*. However, for diffusive media, *i.e.*, scattering-dominated[1], this converges very slowly. Hence, the idea was the following: accelerate the overall resolution by using a less computationally intensive calculation – also termed *low-order* – in between each transport (*high-order*) iteration.

As a result, this would yield a correction to the input flux that had been initially given to the low-order computation. The hope is that the sum of the original input plus the correction will allow for a faster convergence on the following transport iteration. While the the Synthetic Acceleration (SA) method is an additive (linear) scheme, the correction can also be multiplicative in other methods. Depending on the acceleration scheme used and the problem at hand, this method can work extremely well.

Numerous acceleration schemes have been devised over the years, a lot of which has been thoroughly described in a seminal review article by Adams and Larsen [67]. Examples include

- using the Coarse Mesh Rebalance (CMR) method where the problem is solved on a coarser spatial mesh;

---

[1]Scattering-dominated regions are sometimes also termed *optically thick*.

- newer variants on the CMR method such as the Spatially Variant Rebalance (SPV) or Response Matrix Acceleration (RMA) methods as described by Ford [68];

- using a transport equation of lower angular order, *i.e.*, when solving in S_{10} for example, an S_2 transport iteration is used as the intermediate step;

- using the diffusion equation as the low-level operation, leading to the Diffusion Synthetic Acceleration (DSA); and

- the *KP Method* which is a family of methods where a number of low-order level operations with differential operators of different orders are used. In this simplest form, this reduces to the DSA.

The last three are synthetic accelerations and known in the mathematical community as preconditioning methods, and the SI scheme can also be shown to be equivalent to a *Richardson matrix iteration scheme* [67]. Perhaps the most well-known schemes are the DSA and its derivatives, which have been applied with quite a lot of success. [57, 69, 70, 71]

In this work, however, we aim to use the SP_n equation discretised using the Raviart-Thomas (RT) method, as the low-level approximation for the synthetic acceleration. This will allow us to test a few things:

- Is the RT discretisation of the SP_n equation *consistent* with the High Order Diamond Difference (HODD) or Discontinuous Galerkin (DG) methods, insofar that the acceleration scheme is unconditionally stable? If not, when is it unstable and how much does that affect benchmark cases?

- The isotropic SP_1 equations are equivalent to the isotropic diffusion equation. However, do higher-order SP_n equations or considering an anisotropic scattering source offer any advantages?

Basic functionalities of this SA (SP_n equation discretised using the Raviart-Thomas (RT) method) had been previously implemented in DRAGON5 [36] for the HODD spatial discretisation. However, it was unstable with the higher-polynomial orders of HODD or with reflective boundary conditions. New more stable algorithms were devised and it was extended to the Discontinuous Galerkin Finite Element Method (DGFEM) as well as hexagonal geometries. Fourier analyses were carried out and the number of iterations and computational times were quantified for benchmark cases in an attempt to answer the aforementioned questions. We will henceforth refer to the implemented method as RT-SP_nSA or SP_nSA.

5.2 Theoretical Background and Prior Work

This section starts by presenting the formalism behind the Synthetic Acceleration (SA). The general DSA formulation is then outlined, as well as some description of previous studies. While we do not investigate a DSA in this work, we think it is important to understand the method as it is the closest point of comparison with SP_nSA– not to mention the equivalence at lowest order for isotropic cases. It also sheds light on the idea of consistency. Finally, the Fourier Analysis (FA) method is described. The theory presented here has been somewhat adapted from the most salient points of Adams and Larsen [67].

5.2.1 Source iteration and synthetic acceleration

We reproduce below the isotropic scattering transport equation with an external source,

$$\boldsymbol{\Omega} \cdot \nabla \psi(\boldsymbol{r}, \boldsymbol{\Omega}) + \Sigma(\boldsymbol{r}) \psi(\boldsymbol{r}, \boldsymbol{\Omega}) = \frac{\Sigma_s(\boldsymbol{r})}{4\pi} \phi(\boldsymbol{r}) + Q_{\text{ext}}(\boldsymbol{r}, \boldsymbol{\Omega}) \,, \tag{5.1}$$

where

$$\phi(\boldsymbol{r}) = \phi_0(\boldsymbol{r}) = \int_{4\pi} d^2\Omega \ \psi(\boldsymbol{r}, \boldsymbol{\Omega}) \,, \tag{5.2}$$

and the notation is the same as in Chap. 2. In operator notation, we have

$$\mathcal{L}\psi = \mathcal{S}\psi + Q_{\text{ext}} \,, \tag{5.3}$$

where the explicit spatial and angular dependencies are dropped to lighten the notation.

The scattering source depends on the angular flux – which is what is being solved for. Because of this cyclic dependency, an iterative scheme is needed. This is the Source Iteration (SI) scheme, and it can explicitly shown by

$$\mathcal{L}\psi^{(\kappa+1)} = \mathcal{S}\psi^{(\kappa)} + Q_{\text{ext}} \ \text{ where } \kappa \geq 0, \tag{5.4}$$

where κ is the iteration index and the initial flux can be random, uniform or an educated guess. When the difference between successive scalar fluxes is less than a pre-set convergence criterion, the iterations are stopped.

As the SI scheme is notoriously slow to converge in systems where neutrons undergo a high number of collisions, we want to represent an accelerated scheme. To analyse this, consider step $(\kappa+1)$ being redefined to $(\kappa+1/2)$ to represent the source iteration before the correction such that,

$$\mathcal{L}\psi^{(\kappa+1/2)} = \mathcal{S}\psi^{(\kappa)} + Q_{\text{ext}} \ \text{ where } \kappa \geq 0. \tag{5.5}$$

Step $(\kappa+1)$ will now represent the flux after the low-order correction. After subtracting Eqn. 5.5 from Eqn. 5.3, and expanding:

$$\begin{aligned}
\mathcal{L}(\psi - \psi^{(\kappa+1/2)}) &= \mathcal{S}(\psi - \psi^{(\kappa)}) \\
&= \mathcal{S}(\psi - \psi^{(\kappa+1/2)}) + \mathcal{S}(\psi^{(\kappa+1/2)} - \psi^{(\kappa)}) \; ,
\end{aligned} \tag{5.6}$$

we observe,

$$\psi = \psi^{(\kappa+1/2)} + (\mathcal{L} - \mathcal{S})^{-1}\mathcal{S}(\psi^{(\kappa+1/2)} - \psi^{(\kappa)}) \; , \tag{5.7}$$

which defines the full exact solution ψ. However, this still requires the inversion of the full transport operator $(\mathcal{L} - \mathcal{S})^{-1}$ – which would be quite inefficient. Therefore, $(\mathcal{L} - \mathcal{S})^{-1}$ is replaced by a 'low-order' operator, $\mathcal{M}$, such that $\mathcal{M} \approx (\mathcal{L} - \mathcal{S})^{-1}$. This is the *synthetic acceleration*. Eqn. 5.7 is then written as

$$\psi^{(\kappa+1)} = \psi^{(\kappa+1/2)} + \mathcal{M}\mathcal{S}(\psi^{(\kappa+1/2)} - \psi^{(\kappa)}) \; , \tag{5.8}$$

where the $(\kappa + 1)$ step has been explicitly shown.

5.2.2 Prior DSA formulations

Starting with Eqn 5.1 above and using Larsen's four-step approach [39], it is possible to derive the equations to the Diffusion Synthetic Acceleration (DSA) method. In fact, this is outlined in Appendix C. The DSA scheme is given by:

$$\boldsymbol{\Omega} \cdot \nabla \psi^{(\kappa+1/2)}(\boldsymbol{r}, \boldsymbol{\Omega}) + \Sigma(\boldsymbol{r})\psi^{(\kappa+1/2)}(\boldsymbol{r}, \boldsymbol{\Omega}) = \frac{\Sigma_s(\boldsymbol{r})}{4\pi}\phi^{(\kappa)}(\boldsymbol{r}) + Q_{\text{ext}}(\boldsymbol{r}, \boldsymbol{\Omega}) \; ; \tag{5.9a}$$

$$-\nabla \cdot \frac{1}{3\Sigma(\boldsymbol{r})}\nabla F_0^{(\kappa+1)}(\boldsymbol{r}) + (\Sigma_t(\boldsymbol{r}) - \Sigma_s(\boldsymbol{r}))\, F_0^{(\kappa+1)}(\boldsymbol{r}) = \Sigma_s(\boldsymbol{r})\left(\phi_0^{(\kappa+1/2)}(\boldsymbol{r}) - \phi_0^{(\kappa)}(\boldsymbol{r})\right) \; ; \tag{5.9b}$$

$$\phi_0^{(\kappa+1)}(\boldsymbol{r}) = \phi_0^{(\kappa+1/2)}(\boldsymbol{r}) + F_0^{(\kappa+1)}(\boldsymbol{r}) \; . \tag{5.9c}$$

The first equation is the usual transport source iteration; the second equation yields a correction, $F_0^{(\kappa+1)}(\boldsymbol{r})$, to the scalar flux using a diffusion equation and the output from the source iteration; the final equation is simply an additive correction to obtain the new flux.

However, quite clearly, this is only analytical. For deterministic numerical simulations, this needs to be discretised. An issue of crucial importance when it comes to DSAs is the search for a *consistent* spatial discretisation scheme for both transport and diffusion equations. Indeed, in 1971, Reed [72] demonstrated that the method was unstable for cases where the cells had widths greater than one Mean Free Path (MFP). An MFP is the average distance travelled

by a neutron in between interactions and is given by $\frac{1}{\Sigma}$.

A breakthrough came with Alcouffe's paper in 1977 [73]. Indeed, he showed that the choice of the difference method for the diffusion equation is important to ensure a convergent solution and that it was possible to have stability for all mesh sizes. His realisation was that the discretisations of the transport and diffusion equations had to be intimately linked for them to be consistent. Using this, he was able to derive a discretised diffusion equation that yielded a stable DSA. He further demonstrated that the instability obtained by Reed [72] were due to this very fact – or rather, the lack thereof.

Then in 1982, Larsen [39] showed that if the discretised diffusion equation is derived from the discretised transport equation, this always ensures a *completely consistent* DSA, *i.e.*, stable and convergent for all mesh sizes. This is known as the *standard linear DSA*. However, it also results in an algebraically complex equation – which can be difficult to solve for. It was applied to the discontinuous Finite Element Method (FEM) and implemented for the 3D discrete-ordinate transport equation with isotropic scattering, on a mesh of unstructured tetrahedral elements, by Warsa *et al.* [71] in 2002. And it indeed yielded a computationally ineffective DSA because complex mixed P_1 equations[2] based on a DGFEM discretisation needed to be solved at every iteration. "Partially consistent" still outperformed it under most circumstances, except in highly diffusive media at high S_N orders. And that is the hope when devising partially consistent schemes: that the reduction in complexity is not offset by worse performance and/or instability.

Khalil [74] tried to get around this issue of complexity by applying the same discretisation method used for the transport equation directly to the analytical diffusion equation. Starting from that point (instead of the discretised transport equation) resulted in an algebraically much simpler process. This was highly successful in 1D but its performance in higher dimensions was unclear [75], and there does not seem to have been much work done on it since. Yet another popular DSA method is the *Modified Four Step (M4S) method* by Adams and Martin [75]. However, it is also well-known that this is unstable at intermediate mesh cell sizes (in terms of MFP).

More recently, in 2009, Wang [57] developed a DSA scheme similar to Warsa *et al.* [71], based on the P_1 equations. It was only partially consistent as only the zeroth moment of the transport equation was retained. This was applied to adaptive mesh refinement code for unstructured meshes. It was quite successful but was shown to lose some effectiveness for void boundaries and strong material discontinuities. Also, Févotte [69] designed a DSA for

---

[2]P_1 equations are the zeroth and first moments of the transport equation but without the second and higher order moments, as a linearly anisotropic flux is assumed.

use in piecewise domain calculations that was shown to perform as well as regular partially consistent DSAs.

5.2.3 Fourier Analysis (FA)

To properly characterise the performance of the SI scheme and any DSA scheme used to accelerate it, the convention in the community has been to use Fourier Analysis (FA). Most novel DSA schemes are profiled with one as they are implemented. There are various ways of describing as well as implementing it, and in this section, we take inspiration from the works of Févotte [69] and, Adams and Larsen [67].

The idea behind the technique is to decompose the error into frequencies (or modes) that are represented by Fourier wavenumbers, and then study how each frequency is dampened through the iterative process. The more the errors are suppressed, the better the method, and the faster the convergence. It is important that a broad range of error modes be tested. The mode that is slowest to dampen will eventually characterise the iterative process if present in a particular problem. Its dampening rate will be the lowest and is known as the *spectral radius*. If ever it is higher than 1, the scheme is unstable for that particular case. Fourier analyses are usually performed for simple homogeneous test cases or periodic doubly heterogeneous ones.

Considering an infinite one-dimensional homogeneous slab geometry, represented by a domain $\mathcal{D} = [0, L]$ with reflective boundary conditions, the transport equation with isotropic scattering reduces to,

$$\mu\frac{\partial\psi^{(\kappa+1)}}{\partial x}(x, \mu) + \Sigma\psi^{(\kappa+1)}(x, \mu) = \frac{\Sigma_s}{2}\int_{-1}^{-1} d\mu'\psi^{(\kappa)}(x, \mu') + Q(x, \mu) \ , \qquad (5.10)$$

with boundary conditions,

$$\psi^{(\kappa+1)}(0, \mu) = \psi^{(\kappa+1)}(0, -\mu)$$
$$\psi^{(\kappa+1)}(L, \mu) = \psi^{(\kappa+1)}(L, -\mu) \ .$$

Introducing the scattering ratio, $c = \Sigma_s/\Sigma$, and using the 1D equivalent of Eqn. 5.2, this becomes,

$$\mu\frac{\partial\psi^{(\kappa+1)}}{\partial x}(x, \mu) + \Sigma\psi^{(\kappa+1)}(x, \mu) = c\Sigma\phi^{(\kappa)}(x) + Q(x, \mu) \ . \qquad (5.11)$$

Applying the same procedure as in Sec. 5.2.1 (when the general formulation of the synthetic

acceleration was derived), Eqn. 5.10 is subtracted from its *exact* equivalent, to give,

$$\mu \frac{\partial f^{(\kappa+1)}}{\partial x}(x,\mu) + \Sigma f^{(\kappa+1)}(x,\mu) = c\Sigma F^{(\kappa)}(x) \ . \tag{5.12}$$

where F is defined as previously and $f^{(\kappa)} = \psi(x,\mu) - \psi^{(\kappa)}(x,\mu)$; these represent the errors after the κth iteration on the scalar and angular fluxes respectively. The above demonstration shows that analysing the convergence to a given flux ψ for a system with an arbitrary source $Q(x,\mu)$ is essentially the same as analysing the convergence to zero (*i.e.*, no error) for system with the source set to 0.

For initial scalar error given by

$$F^{(0)}(x) = \cos\left(\omega\Sigma x\right) \ , \tag{5.13}$$

where $\omega = \frac{2\pi}{\Sigma L}$ and $\omega\Sigma$ is the Fourier wavenumber denoting the frequency for this initial error, the first iteration yields,

$$f^{(1)}(x,\mu) = \frac{c\,\mu\,\omega\sin(\omega\Sigma x) + c\cos(\omega\Sigma x)}{2(\mu^2\omega^2 + 1)} \ . \tag{5.14}$$

Integrating to find the new scalar error gives,

$$\begin{aligned} F^{(1)} &= \int_{-1}^{-1} f^{(1)}(x,\mu)d\mu \\ &= \frac{c\arctan\omega}{\omega}F^{(0)}(x) \ . \end{aligned} \tag{5.15}$$

It should be noted that the same equation is obtained when initialising the error with an odd mode (*i.e.*, using sin()).

A recursive relation can then be obtained such that

$$F^{(\kappa+1)} = \left(\frac{c\arctan\omega}{\omega}\right)^{\kappa} F^{(0)}(x) \ , \tag{5.16}$$

which implies that errors of the form, $F^{(0)}(x)$ are eigenfunctions of the SI operator, with eigenvalues $\frac{c\arctan\omega}{\omega}$. Therefore, for reasonable initial errors,

$$\begin{aligned} \left\|F^{(\kappa+1)}\right\| &\le \left\|\frac{c\arctan\omega}{\omega}\right\|^{\kappa}\left\|F^{(0)}(x)\right\| \\ &\le \rho_{SI}^{\kappa}\left\|F^{(0)}(x)\right\| \end{aligned} \tag{5.17}$$

where

$$\rho_{SI} = \frac{c \arctan \omega}{\omega} \ . \tag{5.18}$$

A similar analysis can be carried for the diffusion equation [69] to obtain the eigenvalue for the DSA scheme as

$$\rho_{DSA} = \frac{\omega^2 \ \rho_{SI}(\omega) + 3\rho_{SI}(\omega) - 3c}{\omega^2 - 3c + 3} \ . \tag{5.19}$$

Now that the eigenvalue has been obtained as a function of the error frequency, Eqns. 5.18 and 5.19 can be plotted, as shown in Fig. 5.1, for a value of $c = 1$. It can clearly be seen that the largest eigenvalue, $i.e.$, the spectral radius, is $c = 1$ for in the case of only source iteration and occurs at $\omega = 0$. This explains why SI can be painfully slow in highly optically thick regions. On the other hand, for DSA, the spectral radius is roughly 0.2247 for $\omega \approx 2.5$. This shows that the DSA is much more efficient at suppressing the error modes.

5.3   RT-SP_n Synthetic Acceleration (RT-SP_nSA)

As previously outlined in the introduction of this chapter, we make use of the SP_n equations (summarised in Sec. 2.2.4) for the synthetic acceleration in this work. As previously mentioned, only isotropic-source SP_1 equations are equivalent to the diffusion equation in cases where the P_1 scattering terms are vanishing. With RT-SP_nSA, the correction step (Eqn. 5.9b) of the SA scheme changes to

$$\frac{l}{2l + 1}\boldsymbol{\nabla} \cdot \boldsymbol{F}_{l-1}^{(\kappa+1)}(\boldsymbol{r}) + \frac{l + 1}{2l + 1}\boldsymbol{\nabla} \cdot \boldsymbol{F}_{l+1}^{(\kappa+1)}(\boldsymbol{r}) + \Sigma(r)F_l^{(\kappa+1)}(\boldsymbol{r}) = Q_l^{(\kappa+1/2)}(\boldsymbol{r}) \ , \ l \text{ even}$$

$$\frac{l}{2l + 1}\boldsymbol{\nabla} F_{l-1}^{(\kappa+1)}(\boldsymbol{r}) + \frac{l + 1}{2l + 1}\boldsymbol{\nabla} F_{l+1}^{(\kappa+1)}(\boldsymbol{r}) + \Sigma(r)\boldsymbol{F}_l^{(\kappa+1)}(\boldsymbol{r}) = \boldsymbol{Q}_l^{(\kappa+1/2)}(\boldsymbol{r}) \ , \ l \text{ odd} \ , \tag{5.20}$$

where $F_l(\boldsymbol{r})$ is as previously defined and $\boldsymbol{F}_l^{(\kappa+1)}(\boldsymbol{r})$ is the correction on the current, given by

$$\boldsymbol{F}_l^{(\kappa+1)}(\boldsymbol{r}) = \Phi_l^{(\kappa+1)}(\boldsymbol{r}) - \Phi_l^{(\kappa+1/2)}(\boldsymbol{r}) \ . \tag{5.21}$$

Moreover,

$$Q_l^{(\kappa+1/2)}(\boldsymbol{r}) = \sum_{l=0}^{L} \frac{2l + 1}{4\pi}\Sigma_{s,l}(\boldsymbol{r}) \sum_{m=-l}^{l} R_l^m(\boldsymbol{\Omega}) \ (\phi_l^{m,(\kappa+1/2)}(\boldsymbol{r}) - \phi_l^{m,(\kappa)}(\boldsymbol{r})) \ , \tag{5.22}$$

and similarly for $\boldsymbol{Q}_l^{(\kappa+1/2)}(\boldsymbol{r})$, the source using the current correction.

Eqn. 5.20 is simply the SP_n equations seen earlier (Eqn. 2.22), but defined with the correction in fluxes. These are then discretised using the Raviart-Thomas (RT) spatial discretisation,
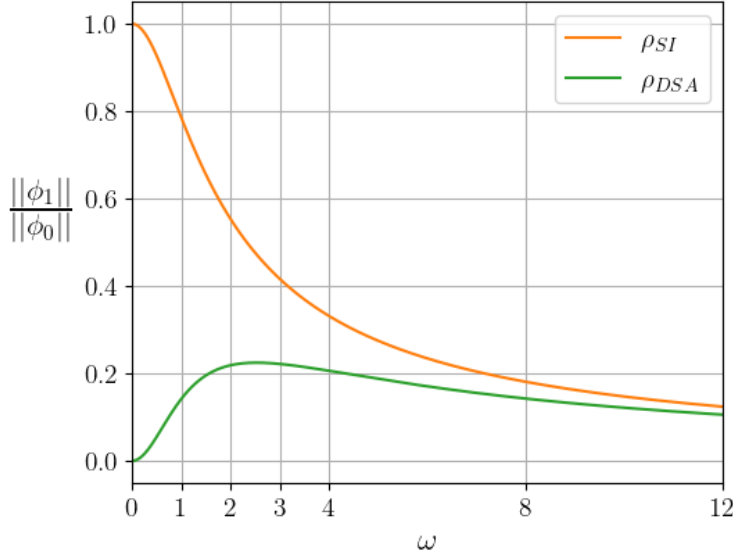
Figure 5.1 Variation of the eigenvalue, ρ in terms of the error frequency, ω.

a mixed dual space finite element method. This was thoroughly described in an article by Hébert [76]. For this reason, this will not be detailed here.

The motivation behind the use of the RT-discretised SP_n as an SA was the finding of a perfect numerical agreement of results [77] in 1D between the HODD flat order (*i.e.*, HODD-0) of the S_N equation and the RT flat order[3] (*i.e.*, RT0) of the P_n equation. As we discussed in Sec. 2.2.4, the P_n and SP_n equations are equivalent in 1D. This then allowed for the speculation that there would perhaps be some sort of partial consistency between these two spatial discretisations – at least in 1D. This was then assumed heuristically for higher dimensions.

The RT-discretised SP_n solver Hébert [76] had already been implemented in DRAGON5. An isotropic SP_1 solver with RT-k spatial discretisation (where k is the same order as the HODD order for the S_N transport equation) was tested as a synthetic acceleration. The results were promising but with our tests, the acceleration was unstable with high-order transport equations and reflective boundary conditions.

This work goes further in the implementation and investigation. First, new algorithms were devised for the correction of fluxes as well as the treatment of reflective boundary conditions. These are described below. Also, this SA was studied with DGFEM. And while the afore-mentioned perfect numerical agreement of results does not hold true in this case, considering how close the results are for HODD and DGFEM, it was expected to perform well.

In the next section, we present a numerical Fourier Analysis (FA) of the improved RT-SP_nSA.

---

[3]The flux is flat order, the current is first order.

This investigates the stability of the acceleration method: if the spatial discretisations are consistent, one should observe unconditional stability for all mesh sizes. Moreover, the fact that SP_n equations are being used allows us to investigate whether higher-order SP_n (for *e.g.*, SP_3) or source anisotropy improves the acceleration method. These questions are examined, and we present numerical results for some of the benchmarks already seen thus far.

5.3.1 Flux correction

The scalar flux in either of the HODD or DGFEM methods and the even-parity flux in RT are both expanded using Legendre polynomials, *i.e.*, in 2D,

$$\phi_l^m(u,v) = \sum_\alpha^\Lambda \sum_\beta^\Lambda \widetilde{P}_\alpha(u)\widetilde{P}_\beta(v) \; \phi_l^{m,[\alpha,\beta]} \; . \tag{5.23}$$

As such, if both the S_N and the SP_n problems are properly converged, one might expect them to be equivalent. And when running separate calculations on a test case, the moments are indeed very similar.

However, when linear polynomials (or indeed any higher order) are used to expand the flux for the transport problem, using the RT-1 discretisation as the acceleration could result in a substantially slower convergence or no convergence altogether for certain test cases. Admittedly, these were particular demanding featuring high scattering ratios and anisotropy (the 2D-AIC benchmark).

That being said, this was not observed when using the HODD-0 discretisation with RT-0. As such, a solution was found in applying the zeroth moment correction as a scaled fraction to the higher moments. This is perhaps better represented as,

$$\phi^{[\alpha,\beta](\kappa+1)} = \phi^{[\alpha,\beta](\kappa+1/2)} \times \left(1 + \frac{F^{[0,0](\kappa+1/2)}}{\phi^{[0,0](\kappa+1/2)}}\right) \; , \tag{5.24}$$

where α and β cannot both be zero. The $\cdot_l^m$ indices were dropped to lighten the notation. It should be noted that, while this amounts to a linear correction for the zeroth moment, it is non-linear for all higher moments.

A similar proportionality calculation was carried out to obtain the corrections to the anisotropic fluxes. Indeed, while the mixed-dual Raviart-Thomas method calculates the odd-parity flux ($l = 1$) intrinsically, these are edge fluxes, and not centre fluxes, as in HODD/DFE. Not to mention, the current (equivalent to the $l = 1$ flux here) is set to zero when dealing with reflective boundary conditions in this method.

5.3.2 Treatment of reflective boundary conditions

As seen in Sec. 3.2.4, when reflective (or albedo) boundary conditions are present, the outgoing boundary surface fluxes for each direction, $\boldsymbol{\Omega}_n$, are stored in between each transport sweep, and used to initialise the next sweep. Therefore, when a synthetic acceleration method is applied, these surface fluxes also need to be corrected. If not, again, depending on the test case, one might end up with a slower convergence or none at all.

Recall, from Sec. 2.2.4, the scalar flux is given from the quadrature rule by

$$\phi_{l,i,j}^{m,[\alpha,\beta]} = \sum_{n=1}^{M} \omega_n R_l^m(\boldsymbol{\Omega}_n) \psi_{n,i,j}^{[\alpha,\beta]} \ , \tag{5.25}$$

where M is the number of directions over the unit sphere, and we have made explicit the spatial discretisation through the $[\alpha,\beta]$ superscript. The difficulty here is: while every spatial moment of each angular moment, *i.e.* $\psi_{n,i,j}^{[\alpha/\beta]}$, of the surface flux (corresponding to each direction) needs to be corrected, the only correction available and usable from the SA is for the isotropic integrated volumetric flux, *i.e.* $\phi_{l,i,j}^{m,[\alpha,\beta]}$ with $l = m = 0$.

We describe below a method to deal with this. While it is relatively straightforward, we were unable to find any such description in a review of current literature. The method is outlined below for the right side of a two-dimensional reference square element; the remaining sides or a 3D test case follow by analogy. Therefore,

1. Obtain the uncorrected scalar surface flux, $\phi(1/2, v)^{(\kappa+1/2)}$, by evaluating the uncorrected scalar volumetric flux on the boundary,

$$\phi(1/2, v)^{(\kappa+1/2)} = \sum_{\alpha=0}^{\Lambda} \sum_{\beta=0}^{\Lambda} \sqrt{2\alpha + 1} \ \widetilde{P}_\beta(v) \ \phi^{[\alpha,\beta](\kappa+1/2)} \ . \tag{5.26}$$

This allows us to get the uncorrected surface flux moments as

$$\phi^{[*,\beta](\kappa+1/2)} = \sum_{\alpha=0}^{\Lambda} \sqrt{2\alpha + 1} \ \phi^{[\alpha,\beta](\kappa+1/2)} \ , \ \forall \beta = [0, \Lambda] \ . \tag{5.27}$$

2. Boundary surface fluxes can also be viewed as outgoing/incoming currents, and they can fall in one of four categories for a square reference element: $\mu_n > 0$, $\mu_n < 0$, $\eta_n > 0$, and $\eta_n < 0$, where $\boldsymbol{\Omega}_n = (\mu_n, \eta_n)$.

   Hence, a different value for spatial moments of the uncorrected scalar surface flux can be acquired by integrating the angular surface fluxes over only the relevant half of the S_N quadrature. For example, in the case of the right $(x+)$ boundary, instead of Eqn.

5.25, we have

$$\phi_{BF}^{[*,\beta](\kappa+1/2)} = \sum_{\substack{n=1 \\ \mu_n>0}}^{M} \omega_n \, \psi_n^{[*,\beta](\kappa+1/2)} \, , \ \forall\beta = [0,\Lambda] \, , \tag{5.28}$$

where we have used the subscript BF to indicate that this is obtained from the surface *Boundary Fluxes*, and we consider only the isotropic flux such that $R_l^m(\mathbf{\Omega}_n) = 1$. $\phi^{[*,\beta](\kappa+1/2)}$ and $\phi_{BF}^{[*,\beta](\kappa+1/2)}$ will not be equal as the order of operations (evaluation at the boundary and integration/quadrature formula) does matter in this case. However, this allows for the obtention of a ratio of the two,

$$R^{[\beta]} = \frac{\phi_{BF}^{[*,\beta](\kappa+1/2)}}{\phi^{[*,\beta](\kappa+1/2)}} \, , \ \forall\beta = [0,\Lambda] \, . \tag{5.29}$$

3. Step 1 is then repeated with the accelerated scalar volumetric flux, to give the accelerated surface flux moments, $\phi^{[*,\beta](\kappa+1)}$.

4. An estimate of $\phi_{BF}^{[*,\beta](\kappa+1)}$ is then calculated using $\phi^{[*,\beta](\kappa+1)}$ and $R^{[\beta]}$,

$$\phi_{BF}^{[*,\beta](\kappa+1)} = \phi^{[*,\beta](\kappa+1)} \times R^{[\beta]} \, , \ \forall\beta = [0,\Lambda] \, . \tag{5.30}$$

5. Finally, each corrected angular moment of the surface flux is calculated using

$$\psi_n^{[*,\beta](\kappa+1)} = \frac{\psi_n^{[*,\beta](\kappa+1/2)}}{\phi_{BF}^{[*,\beta](\kappa+1/2)}} \times \phi_{BF}^{[*,\beta](\kappa+1)} \, , \ \forall\beta = [0,\Lambda] \, . \tag{5.31}$$

5.4 Parametric Study

We first assess RT-SP_nSA through a one-dimensional Fourier analysis and a test case we found particularly difficult to resolve during our testing. It is investigated whether the angular order of the SP_n equation or the anisotropy of the source scattering has any effect on the acceleration. Also, with the way the flux correction was implemented, it made it possible to use non-matching RT orders for the SA. Essentially, this means that, for example, RT-0 can be applied to accelerate HODD/DG-1 or RT-1 with HODD/DG-2.

5.4.1 Numerical Fourier analysis using DRAGON5

Fourier analyses are usually carried out analytically by starting with an error of the shape given by Eqn. 5.13 and propagating this through the transport and synthetic acceleration equations to obtain the eigenvalue equation and subsequently the spectral radius. However,

for this work, this was done computationally using the DRAGON5 code. The procedure is essentially the same and is tedious only insofar as the amount of data to process. The results presented in this section were run with the initial error containing even modes only (refer to Eqn. 5.13); however similar results were obtained for odd modes (not presented).

This is carried out for a homogeneous one-dimensional test problem with domain, $\mathcal{D} = [0, L]$ where $L = 10$ cm, with vacuum on one end and reflective boundary conditions on the other. A constant unitary source was present throughout and the scattering ratio, c, was set to 0.99, unless otherwise specified.

The range of Fourier wavelengths investigated was $\omega\Sigma = [0, \frac{2\pi}{L})$. As this was done computationally, the range of values was discrete. We used 60 equally spaced discrete values in that range. In our testing, increasing this discretisation did not yield different results.

For mesh elements with a width greater than 1 MFP, the number of elements was kept constant. The total cross-section, Σ, was changed from 1 to 100000 cm^{-1} to vary the optical thickness. However, for elements with spacing smaller than 1 MFP, Σ was kept constant at 1 cm^{-1}, and the mesh was steadily refined down. This was done in an effort to keep leakage from affecting the results. This simulation setup was based on the work of Wang [57].

Finally, the calculation was allowed to run for four iterations for each Fourier wavelength, and the eigenvalue calculated using

$$\rho_{\omega\Sigma} = \sqrt{\frac{\left\|\phi^{(\kappa)} - \phi^{(\kappa-1)}\right\|}{\left\|\phi^{(\kappa-2)} - \phi^{(\kappa-3)}\right\|}} \ , \tag{5.32}$$

where in this case $\kappa = 4$. Note that, in the literature, there are various ways of computing the eigenvalue numerically (see [75] and [69] for example) but Wang [57] seemed to suggest this equation and letting the calculation run for a few iterations would be a way to reduce numerical oscillations in the eigenvalue. And this is indeed what was observed in our simulations. The Numerical Spectral Radius (NSR) is then given by

$$\text{NSR} = \max(\rho_{\omega\Sigma}) \ . \tag{5.33}$$

The RT-SP_nSA equation initially used was an RT-0 SP_1 with an isotropic source. At that point, it was essentially equivalent to the diffusion equation and what was being tested was the consistency of the Raviart-Thomas (RT) discretisation with either HODD or DG.

The spectral radius for the SA is given in Fig. 5.2 for different angles for the HODD-0 spatial discretisation, as well as the spectral radius for the source iteration S_2 problem. The

unaccelerated NSR are exactly where we would expect with $c = 0.99$. For the accelerated problems, even though S_2 is wildly unstable, S_4, S_8 and S_{16} are all completely stable for all mesh sizes. While this was very encouraging, once c was increased to 0.9999, more or less all the test cases became unstable. For this reason, c was kept at 0.99 and the quadrature at S_8 in the rest of the presented numerical Fourier results.

Fig. 5.3 shows how the NSR varies when different transport spatial discretisation orders are tested. Various RT-k discretisations are tested for the same HODD/DG order, Λ. For HODD-0, the RT-0 acceleration is already stable and increasing k is not helpful at all. For the remaining orders, there seems to be a couple of trends emerging. First, RT-0 seems to offer more or less the same acceleration profile regardless of Λ. Also, except in the case of linear transport discretisation ($\Lambda = 1$), RT-2 seems to consistently offer a stable acceleration at all mesh sizes. We repeated the same experiment with SP_3, hopeful that the results would be improved. This was, unfortunately, not the case: the results, while not exactly similar, offered the same – and sometimes, even worse – instabilities around the same regions.

Finally, we were curious to see if using an anisotropic scattering source with the SP_n equation would make any difference when analysing an anisotropic problem. The problem investigated was identical to the one so far but with an additional anisotropic scattering cross-section, given by

$$\Sigma_{s,1} = \bar{\mu}\Sigma_{s,0} \; , \tag{5.34}$$

where $\bar{\mu} = 0.99$. The results are given in Fig. 5.4 and Fig. 5.5 where the SA is compared when using either an isotropic or anisotropic scattering source with the SP_1 equation. The use of the isotropic scattering source in this anisotropic test case results in wildly unstable acceleration. For example, in the case of HODD-0, the unstable peak is about 30 times higher than in the isotropic test (Fig. 5.3), and RT-2 is not stable at all anymore. Fig. 5.4 and Fig. 5.5 show that while using an anisotropic source did not bind these spectral radii to make the problem stable at all mesh sizes, the peaks were drastically reduced – as demonstrated on the right-hand side images which show a zoom on the y-axes.

5.4.2 2D-AIC test case

The benchmark that we were initially unable to make work with the synthetic acceleration (in its prior state) was the 2D-AIC one, described in Sec. 3.3.2. In fact, it was failing to converge, and that was what prompted us to review the whole algorithms for the flux correction and surface fluxes. It seemed only fitting to retest it and see how it fares. We also present here, for this more realistic test case, the impact, if any, of some of the parameters investigated
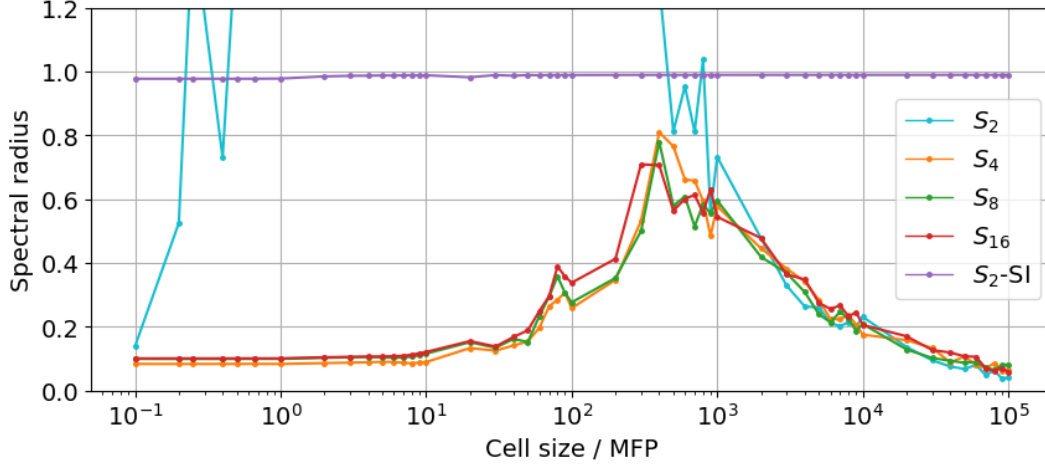
Figure 5.2 Variation of spectral radii with mesh size in terms of Mean Free Path (MFP) for the RT-SP_nSA scheme using RT-0 SP_1 isotropic source. Different S_N angles for the HODD-0 test case are shown, as well as the S_2 source iteration case.

for the numerical Fourier Analysis (FA). All results presented here are run using a quadratic ($\Lambda = 2$) spatial discretisation, and usually S_8 quadrature unless otherwise specified

Fig. 5.6 shows the reduction in calculation time and number of inner iterations from the unaccelerated to the accelerated simulation when using the 'basic' SP_1 isotropic source equation but with different RT-k schemes. All these plots have three data points for the accelerated calculations – each corresponding to when the SA starts being applied. Indeed, it was known from previous experiments that applying the SA after a few unaccelerated transport iterations could be very beneficial to the overall computation. We wanted to verify if that was still the case and sought to quantify for how many iterations waiting would be helpful.

Interestingly, even though from the Fourier analysis, it was RT-2 that was the most stable, we see here that RT-0 provides the most gains in computational times *and* number of iterations. While the higher time could be attributed to the higher computational cost of the RT-2 discretisation, one could have expected the number of iteration to be very similar. Instead, for DGFEM, there is a difference of around 20-25%. Also, when the quadrature is increased to S_{16} (see Fig. 5.7), the difference between RT-2 and RT-0 is much smaller as the SA now represents a smaller percentage of the computation time.

Also, SP_nSA seems to work better with DGFEM – at least in this case. We suspect that it has to do with how the reflective boundary surface flux correction was implemented. HODD and DGFEM define the flux differently on the boundaries. While HODD uses the closure relationships (Eqn. 3.13 and 3.14), Eqn. 5.26 is more representative of the flux for DGFEM.
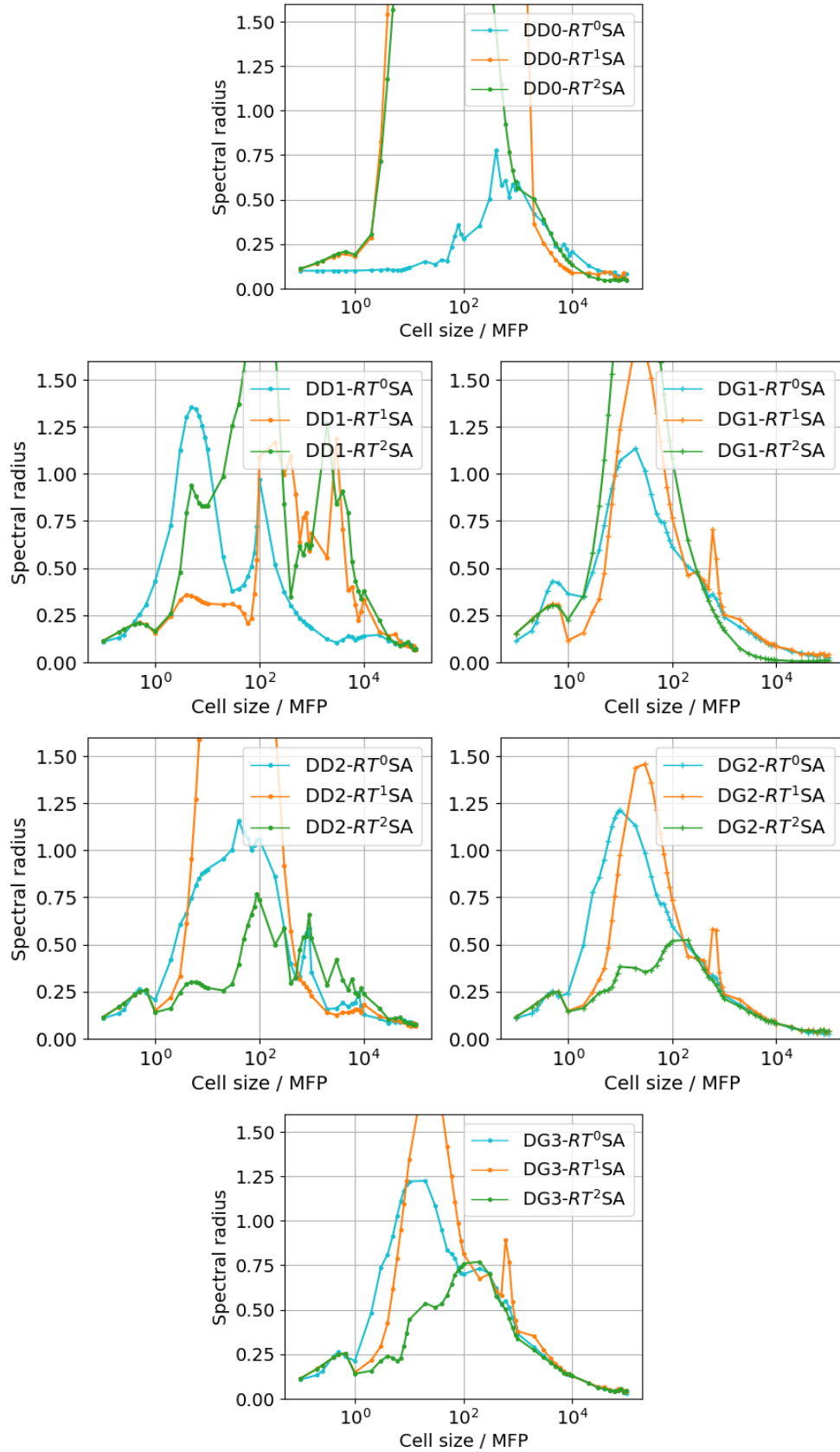
Figure 5.3 Variation of spectral radii with mesh size in terms of Mean Free Path (MFP) for the RT-SP_nSA scheme using RT-k SP_1 isotropic source. Each row top to bottom: HODD-0; HODD/DG-1; HODD/DG-2; DG-3.
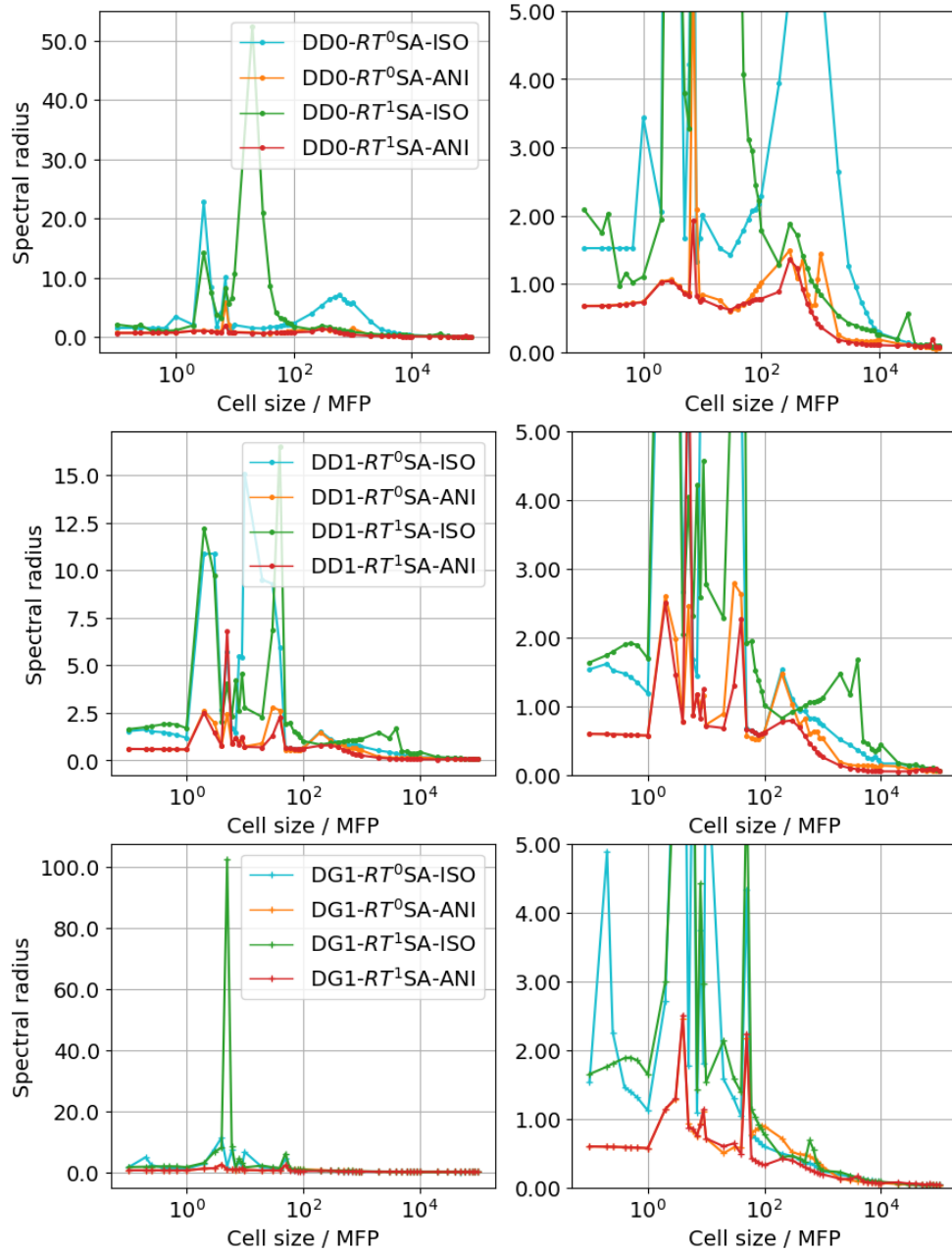
Figure 5.4 Variation of spectral radii with mesh size in terms of Mean Free Path (MFP) for an anisotropic test case using the RT-SP_nSA scheme with either an RT-k SP_1 isotropic or anisotropic source. Each row top to bottom: HODD-0; HODD-1; DG-1. The plots on the right are identical to those on the left but the y-axis is zoomed on, to better see the variation and improvement of using anisotropic scattering with the SP_n equation.
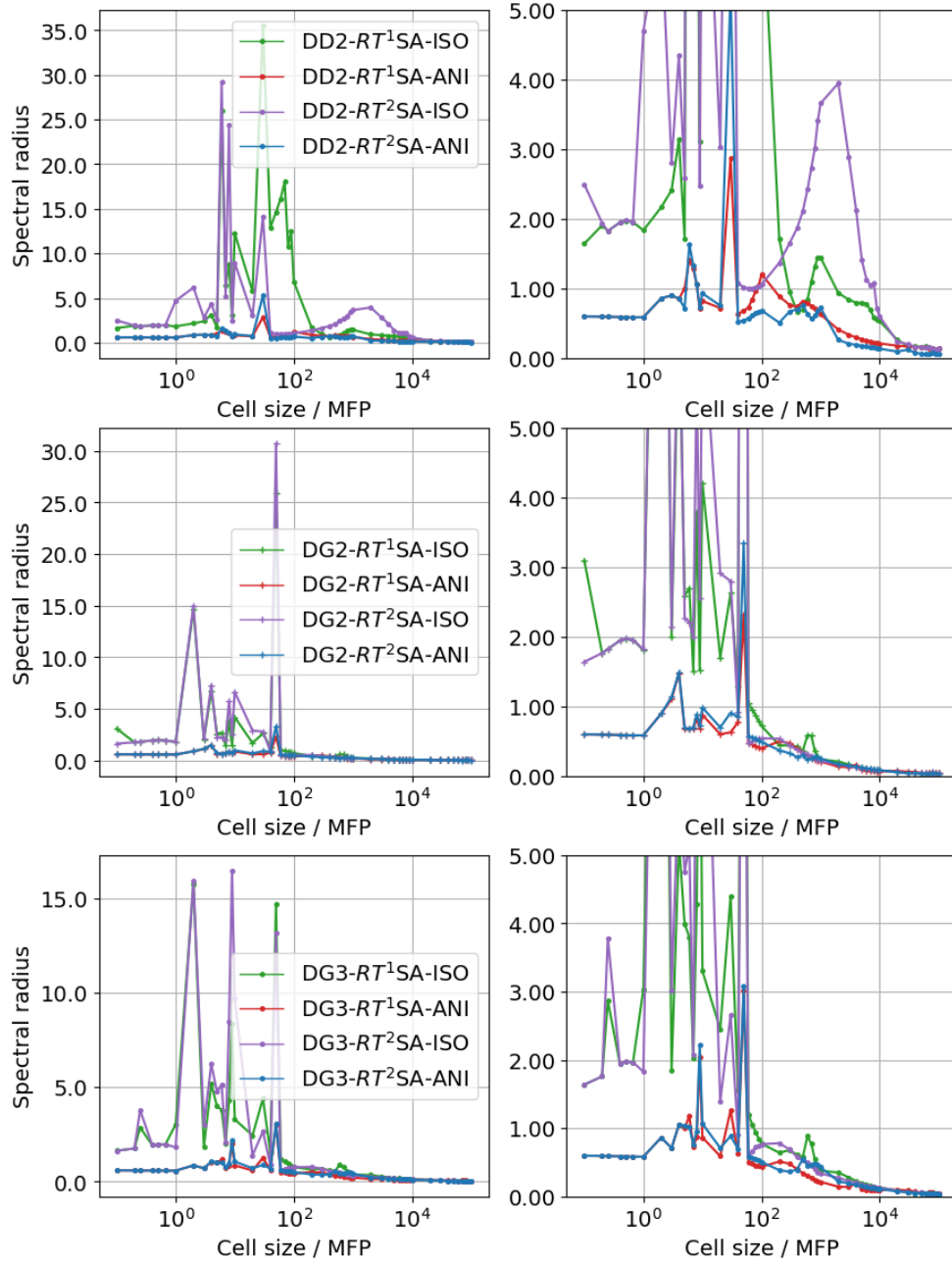
Figure 5.5 Variation of spectral radii with mesh size in terms of Mean Free Path (MFP) for an anisotropic test case using the RT-SP_nSA scheme with either an RT-k SP_1 isotropic or anisotropic source. Each row top to bottom: HODD-2; DG-2; DG-3. The plots on the right are identical to those on the left but the y-axis is zoomed on, to better see the variation and improvement of using anisotropic scattering with the SP_n equation.
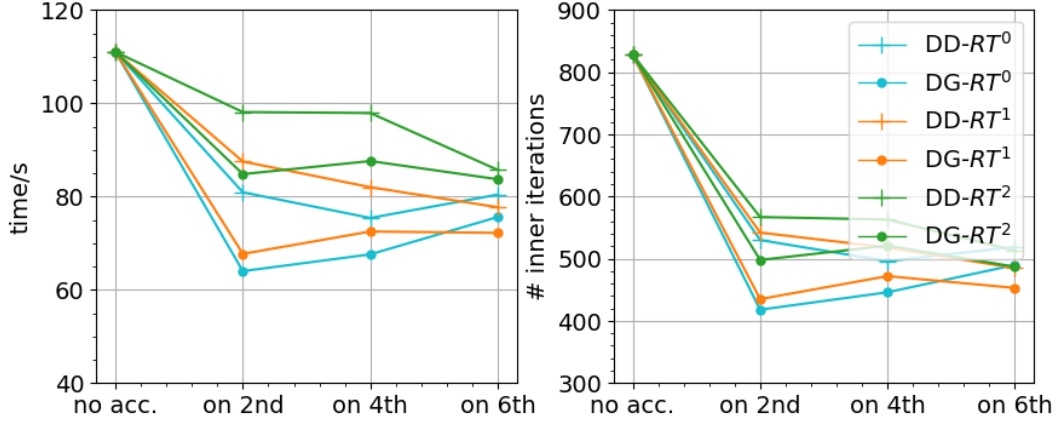
Figure 5.6 S_8 calculations with SP_1 isotropic source SA: plots of computational time and number of inner iterations for the unaccelerated and accelerated calculations. There are three cases for the latter: starting the SA on the 2nd, 4th and 6th transport iteration.

Overall, however, compared with the previous implementation where the RT order, k, was fixed to the transport discretisation order, Λ, there are big improvements – the decrease in computational times for RT-0 more than doubles compared with the gain for RT-2.

We investigate again the anisotropic source SP_nSA, shown in Fig. 5.8; there are slight improvements but nothing quite significant – which, again, is somewhat counter-intuitive to the results seen in the Fourier analysis. The SP_3 equation SA was also computed, both with an isotropic and anisotropic source, given in Fig. 5.9 for the S_8. The results degrade significantly unfortunately. As the S_N order was increased, we found this degradation to reverse course and actually turn into an improvement. This is not shown here as the plots are somewhat repetitive.

Finally, a very interesting 'trick' – for want of a better word – was suggested by Adams [78]: instead of just starting the SA a few iterations later, it could be applied every x number of iterations. This proved to be highly successful, as demonstrated in Fig. 5.10. In fact, there is not much, if any, difference now between isotropic and anisotropic source. Also, while the results for SP_3 here are quite worse than SP_1, it should be noted that these differences all but disappear at higher S_N orders.

Based on Fig. 5.10, we feel confident recommending that this particular SA, be used with an RT-0 SP_1 isotropic source configuration, while being applied every few (2 or 4) iterations.
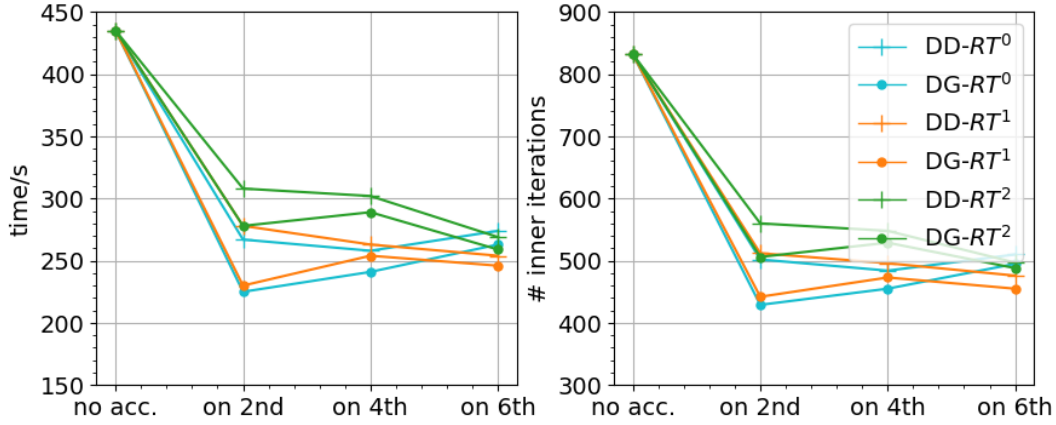
Figure 5.7 S_{16} calculations with SP_1 isotropic source SA: plots of computational time and number of inner iterations for the unaccelerated and accelerated calculations. There are three cases for the latter: starting the SA on the 2nd, 4th and 6th transport iteration.

5.5 Benchmark Results: 3D-TAK2 and 3D-TAK4

We now apply RT-SP_nSA to two three-dimensional benchmarks – one Cartesian: 3D-TAK2 and one hexagonal: 3D-TAK4– from Chap. 3 and Chap. 4. The specific configuration used is the one previously recommended: RT-0 SP_1 isotropic source every x iteration. x was {2,4} for the Cartesian case, {2,4,6} for the hexagonal one. The results are given in Tab. 5.1 and Tab. 5.2 respectively. Note that the differences in k_{eff} are not given as there was usually less than a couple pcm of difference between the unaccelerated and accelerated k_{eff} results.

For 3D-TAK2, with no mesh refinement, there is about a 50% reduction in computational times even if the number of iterations go down by about 75% (except in the HODD-0 scenario). However, as soon as the mesh is refined, the drop in computational time reflect the decrease in number of iterations. Overall, compared to the unaccelerated calculation, the SA is able to provide about a 75% reduction in time.

The results are very similar with 3D-TAK4 except for the fact that now, the best option for x is 4. Indeed, this is something that was observed across the board for hexagonal cases, applying the acceleration every 2 iterations resulted in an unstable scheme that sometimes even diverged.

5.6 Concluding Remarks

Novel algorithms were applied in the flux and surface flux corrections for a synthetic acceleration based on a Raviart-Thomas (RT) discretisation of the SP_n equations. This allowed
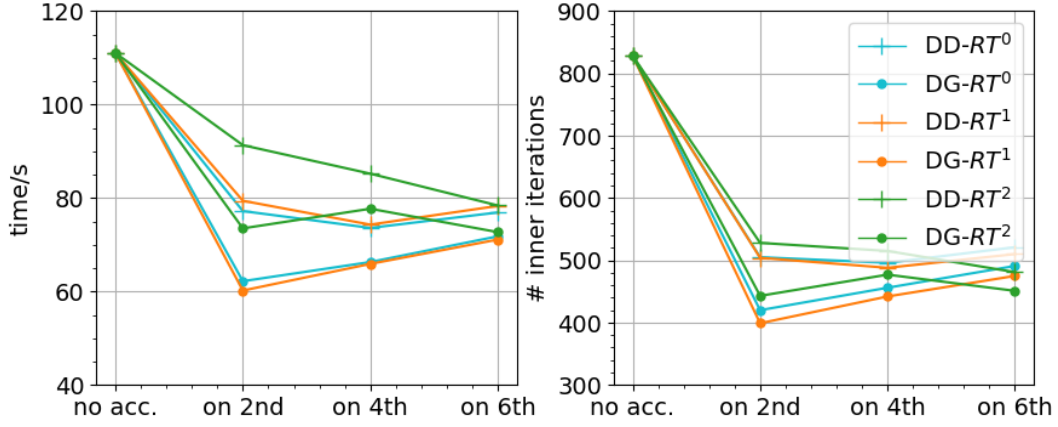
Figure 5.8 S_8 calculations with SP_1 anisotropic source SA: plots of computational time and number of inner iterations for the unaccelerated and accelerated calculations. There are three cases for the latter: starting the SA on the 2nd, 4th and 6th transport iteration.
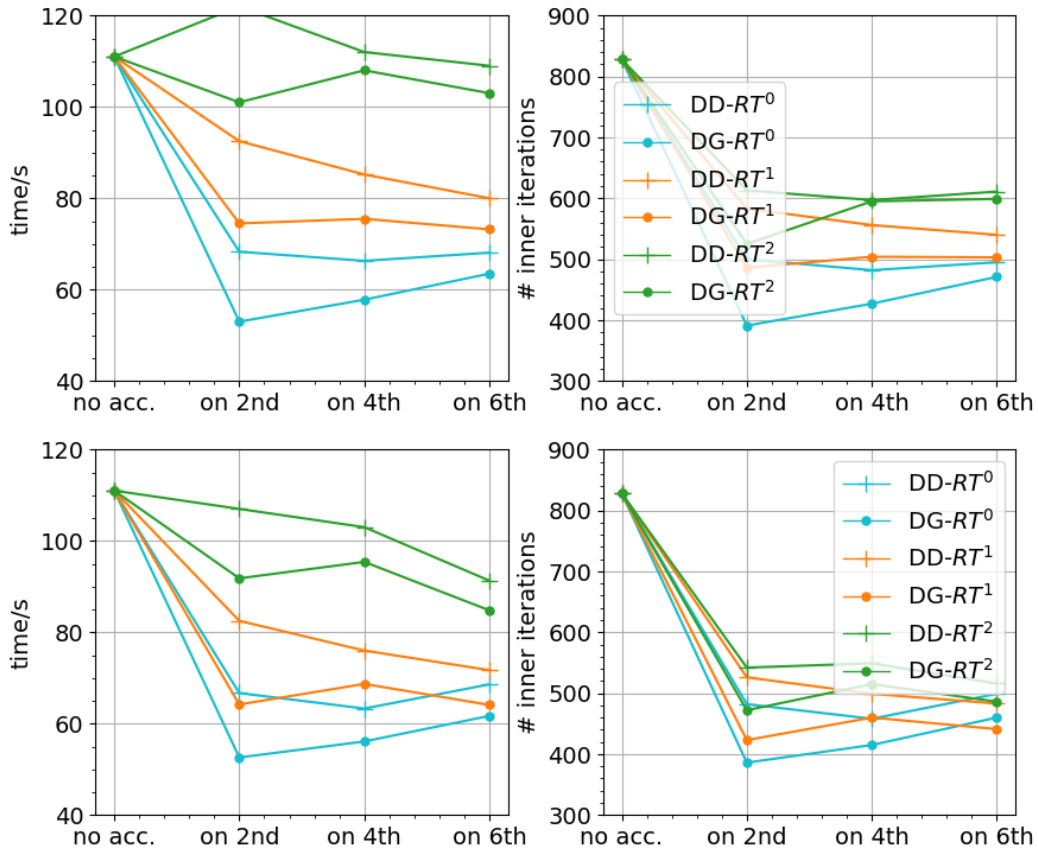
Figure 5.9 Top: S_8 calculations with SP_3 isotropic source SA. Bottom: S_8 calculations with SP_3 anisotropic source SA. Plots of computational time and number of inner iterations for the unaccelerated and accelerated calculations. There are three cases for the latter: starting the SA on the 2nd, 4th and 6th transport iteration.
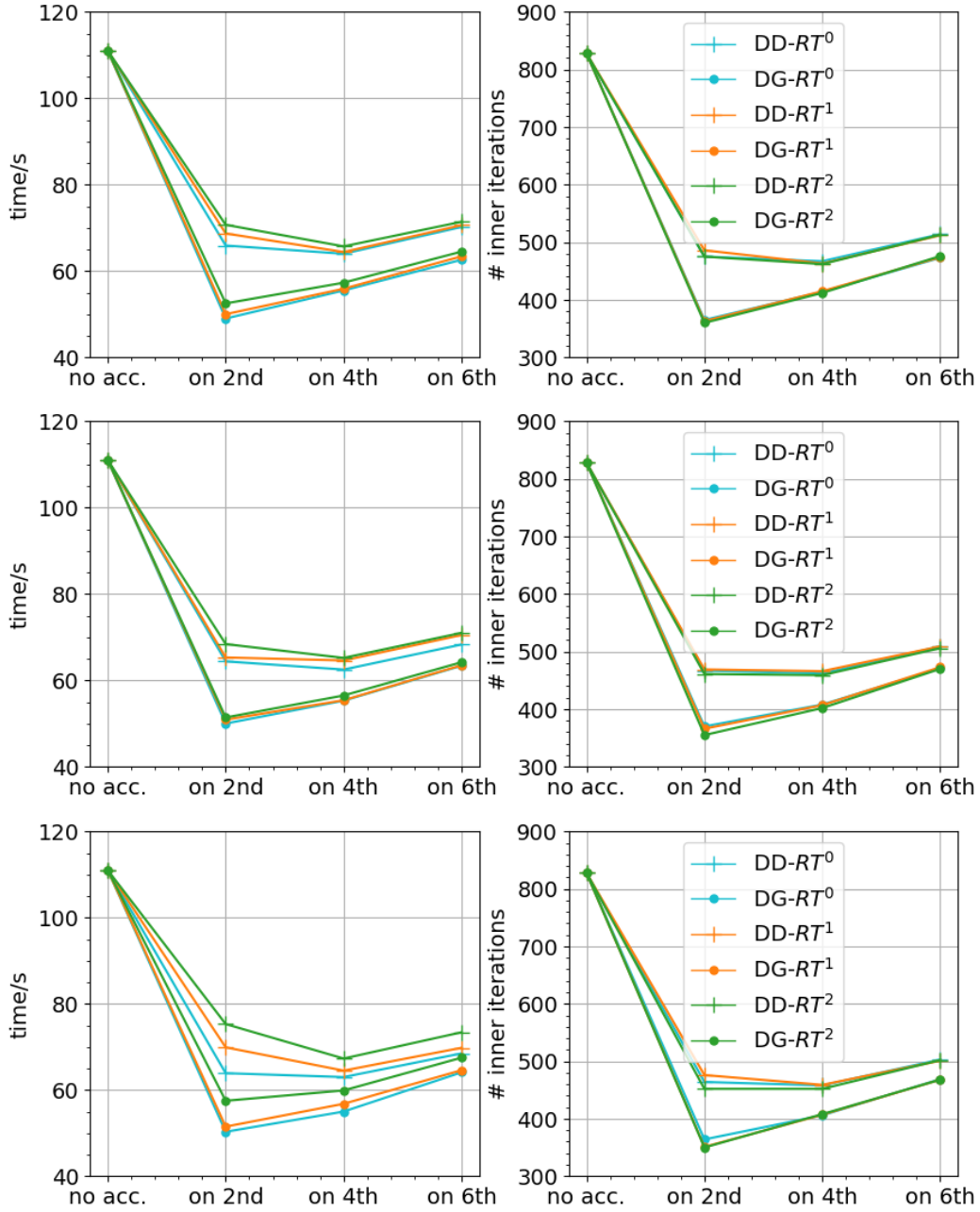
Figure 5.10 Top: S_8 calculations with SP_1 isotropic source SA. Middle: S_8 calculations with SP_1 anisotropic source SA. Bottom: S_8 calculations with SP_3 anisotropic source SA. Plots of computational time and number of inner iterations for the unaccelerated and accelerated calculations. Instead of only starting the SA after 2, 4 or 6 iterations, here, SP_nSA was applied every 2, 4 or 6 iterations.

Table 5.1 Results for 3D-TAK2 with S_6. Acceleration was RT-0 SP_1 isotropic. Λ is the polynomial order, *subm.* the submeshing, *accel.* the acceleration method used. Three significant figures given, except for k_eff.

Λ	*subm.*	*accel.*	DD k_eff	Time (s)	# iter.	DG k_eff	Time (s)	# iter.
0	1	no acc.	0.9589735	17	3122	-	-	-
		every 2nd	0.9589742	3.81	670	-	-	-
		every 4th	0.9589741	4.48	830	-	-	-
	2	no acc.	0.9593314	120	3254	-	-	-
		every 2nd	0.9593269	30.2	674	-	-	-
		every 4th	0.9593269	36.1	850	-	-	-
	3	no acc.	0.9594978	455	3208	-	-	-
		every 2nd	0.9595025	109	699	-	-	-
		every 4th	0.9595025	128	869	-	-	-
1	1	no acc.	0.9595516	129	3196	0.95837	122	3164
		every 2nd	0.9595544	57.6	1341	0.9583729	57	1370
		every 4th	0.9595548	60.8	1435	0.9583711	61.4	1493
	2	no acc.	0.9596105	1060	3216	0.959371	1020	3211
		every 2nd	0.9596144	291	835	0.9593751	302	913
		every 4th	0.9596145	331	978	0.9593751	329	1010
	3	no acc.	0.9596136	3860	3217	0.9595249	3520	3215
		every 2nd	0.9596179	857	741	0.9595291	857	775
		every 4th	0.9596179	1030	903	0.9595292	1010	913
2	1	no acc.	0.9596297	2090	3198	0.9595611	1970	3198
		every 2nd	0.9596334	941	1406	0.9595646	931	1420
		every 4th	0.9596338	960	1437	0.9595635	948	1447
	2	no acc.	0.9596143	16900	3217	0.959606	16100	3216
		every 2nd	0.9596183	4770	891	0.9596102	4640	874
		every 4th	0.9596184	5410	1001	0.9596103	5310	994
	3	no acc.	0.9596141	57000	3216	0.9596112	55800	3218
		every 2nd	0.9596183	14000	756	0.9596154	13700	755
		every 4th	0.9596183	16800	910	0.9596154	16300	904
3	1	no acc.	-	-	-	0.9596089	21900	3199
		every 2nd	-	-	-	0.9596119	9660	1393
		every 4th	-	-	-	0.9596105	10100	1462
	2	no acc.	-	-	-	0.9596129	176000	3215
		every 2nd	-	-	-	0.9596171	49500	889
		every 4th	-	-	-	0.959617	56000	1002

Table 5.2 Results for 3D-TAK4 with S_{10}. Acceleration was RT-0 SP_1 isotropic. Λ is the polynomial order, *subm.* the submeshing, *accel.* the acceleration method used. Three significant figures given, except for k_{eff}.

			DD				DG		
Λ	*subm.*	*accel.*	k_{eff}	Time (s)	# iter.		k_{eff}	Time (s)	# iter.
0	1	no acc.	0.866796	840	4284		-	-	-
		every 2nd		*diverged*			-	-	-
		every 4th	0.8667991	169	842		-	-	-
		every 6th	0.8667992	179	890		-	-	-
	2	no acc.	0.8769145	3250	4335		-	-	-
		every 2nd		*diverged*			-	-	-
		every 4th	0.8769171	656	909		-	-	-
		every 6th	0.8769172	907	955		-	-	-
	3	no acc.	0.8782655	8080	4311		-	-	-
		every 2nd	0.8782649	3000	1470		-	-	-
		every 4th	0.8782687	1620	917		-	-	-
		every 6th	0.8782688	1850	971		-	-	-
1	1	no acc.	0.8809062	6500	4417		0.8731214	5750	4452
		every 2nd		*diverged*				*diverged*	
		every 4th	0.8809069	1370	1015		0.8731231	1450	973
		every 6th	0.880907	1450	1065		0.8731228	1020	1029
	2	no acc.	0.8799127	22500	4322		0.8785824	23000	4458
		every 2nd	0.8799167	9140	1678			*diverged*	
		every 4th	0.8799163	4910	913		0.8785841	4760	878
		every 6th	0.8799162	5710	1023		0.8785841	6460	1001
	3	no acc.	0.8798766	55600	4322		0.8793559	54800	4459
		every 2nd	0.8798805	17900	1406		0.8793581	28100	2167
		every 4th	0.8798801	12100	926		0.8793575	11600	910
		every 6th	0.8798799	12400	1013		0.8793575	13100	1005
2	1	no acc.	0.8798544	93600	4419		0.8797545	91400	4421
		every 2nd	0.8798561	43200	2033			*diverged*	
		every 4th	0.8798554	21500	1013		0.8797554	21300	1021
		every 6th	0.8798555	22700	1072		0.8797554	22200	1072
	2	no acc.	0.8798733	366000	4324		0.8798349	352000	4323
		every 2nd	0.8798773	117000	1396		0.8798388	120000	1447
		every 4th	0.8798768	78100	929		0.8798383	75200	909
		every 6th	0.8798767	89000	1053		0.8798383	86900	1052
	3	no acc.	*estimated*	843000	4324		*estimated*	823000	4323
		every 2nd		*diverged*			0.879859	242000	1309

Table 5.2 *(continued)*

				DD				DG		
Λ	*subm.*	*accel.*		k_{eff}	Time (s)	# iter.		k_{eff}	Time (s)	# iter.
		every 4th		0.879873	173000	917		0.8798585	172000	929
		every 6th		0.8798728	199000	1050		0.8798584	185000	1000
3	1	no acc.		-	-	-		*estimated*	945000	4420
		every 2nd		-	-	-		*diverged*		
		every 4th		-	-	-		0.8798276	218000	1023

us to investigate various parameters such as the angular order of the SP_n equations, the presence of an isotropic or anisotropic source and the spatial order of the RT discretisation. Numerical Fourier analyses were presented. These showed that the scheme was conditionally stable depending on the S_N order and the level of scattering and anisotropy present. A case that was known to be previously difficult to accelerate with the earlier scheme was investigated as well. Based on these, we were able to recommend an RT-0 SP_1 isotropic source equation for the synthetic acceleration. Results were presented for the Takeda Models 2 and 4 benchmarks, showing a possible reduction of up to 75% in computational time.

CHAPTER 6   PARALLEL IMPLEMENTATION IN Wyvern

We knew that High Performance Computing (HPC) would be a part of the project from the very beginning even if we were not entirely certain which form it would take. For reasons explained in this chapter, we chose the Message Passing Interface (MPI) standard as the framework upon which to build our scheme. This implementation was done in a separate static library called WYVERN, which we expect will become part of the main DRAGON5 code library this year.

A brief introduction to some of the parallelisation solutions is first given to showcase the diversity of available strategies. We then briefly discuss the work that has been done for the S_N transport equation in the past. The Cartesian implementation of the parallelisation strategy in WYVERN is then outlined followed by the hexagonal one. Numerical results for the calculations for the Takeda benchmarks seen in preceding chapters are given to highlight the decrease in computational time. Finally, a full core mock breeder reactor is simulated.

6.1   Brief Introduction to Parallelisation Solutions

The use of parallelism in a solver can allow for tremendous gains in computational time. Larger and more complex problems can be solved in, frequently, vastly smaller timeframes. The current landscape in HPC has become quite diverse – both in terms of hardware and parallelisation paradigms.

With regards to hardware, there are two main types: multicore processors and Graphic Processing Units (GPUs). Multicore processors are essentially an extension of one Central Processing Unit (CPU) whereby several of the latter make up a multicore processor. They can be further grouped onto a *socket* – sometimes also called *CPU-chip*. These can be subsequently combined to make up a computing node, which can be viewed as one computer. Several of these computers together make up a computer cluster[1]. Each CPU usually runs one computation *thread* at the same time, although some CPUs are able to run two or more threads in parallel. Finally, each CPU also contains Single Instruction, Multiple Data (SIMD) (see below) units, with their own dedicated memory (called *registers*), able to carry out *vector* operations on data that can fit on the register. In this case, vector operations usually refer to the relatively simple operations[2] carried out synchronously on the whole set of data.

---

[1]The clusters used for this work are part of the Digital Research Alliance of Canada (DRAC).

[2]These include things like arithmetic, compare, data-type conversion or data and memory management operations amongst others. While the list is ever growing, it is a much smaller set of possible instructions

Now, the concept of parallelism is often classified according to what is known as Flynn's Taxonomy [79]. There are four main categories:

- Single Instruction, Single Data (SISD) – essentially, a sequential computer;

- Single Instruction, Multiple Data (SIMD) – all processing units execute the same instruction but using different data. This is suitable for problems with a high degree of regularity. It can be part of the hardware design in which case the units are in lockstep;

- Multiple Instruction, Single Data (MISD) – this is much more unusual, the most famous example was the Space Shuttle flight controller for fault tolerance;

- Multiple Instruction, Multiple Data (MIMD).

There are different ways in which each of these categories can be implemented. For example, as mentioned earlier, vectorisation capabilities are built right into CPUs. Auto-vectorisers present in compilers can take advantage of this, based on the level of optimisation flags at compile time. Unfortunately, this is notoriously unreliable for all but the simplest and most obvious of loops. Another method is through compiler vector intrinsics but it can degrade code readability. Optimised libraries for linear algebra such as INTEL MKL or EIGEN have also been developed.

There is also the INTEL SPMD Compiler (`ispc`) [80] which was specifically developed to address the lack of languages and compilers to take advantage of modern CPU hardware. `ispc` delivers high performance with speed increases of up to $35\times$ having been demonstrated on four-core systems.

Another parallelisation model is the *threads model* which is somewhat of an extension on the SIMD and MIMD, leading to a Single Program, Multiple Data (SPMD) model. It is a type of shared memory programming, where a main program creates a certain number of processes (threads) that are run concurrently and usually perform the same tasks on different data. In this case, because the memory is shared[3] between the threads, they all have access to the same information. The main process loads and acquires all the necessary system resources. This means that they can also overwrite information that might still be needed by other threads if not done correctly – leading to a *race condition.*

One implementation of this model is the OpenMP Application Programming Interface (API). It is an industry standard available in C/C++ and Fortran. It is a fork-and-join model employed mostly in parallel for-loops, although SIMD directives and task-based parallelism have

---

than the x86 set.

[3]*Shared memory* systems, in general, have the ability for all processors to access memory as global address space. Hence, while the processors might operate independently, they all share the same memory resources.

recently been added. Another similar but task-based standard is INTEL TBB (Threading Building Blocks). It is more powerful in that it is capable of creating graphs of dependent tasks. Unfortunately, it is only available in C++.

The last model to be addressed here is the Message Passing Interface (MPI) API. Vectorisation taps into the potential of a CPU; threads makes it possible to use the resources (processors and memory) of a computer or a whole compute node; MPI allows the use of several nodes at the same time, leading again to an SPMD model. MPI relies on a *distributed memory* paradigm whereby nodes have a communication network in between them. Each CPU has their own allocated memory and are unable to see the others'. Communication and exchange of information must be done explicitly by the developer.

Finally, of course, some of these methods can be combined to take full advantage of available resources – so long as the language, compiler and the methods themselves all support it!

6.2 Outline of Prior Parallelisation Work in S_N Neutron Transport

As briefly alluded to in Sec. 2.2.5, the sweep in discrete-ordinates transport does lend itself to parallelisation over the spatial domain. Moreover, as seen in Alg. 1, all directions within one octant of the unit sphere (and even over the whole sphere given no albedo boundary conditions) are independent of one another except for the quadrature summation at the end of the sweep. This means that they could potentially be parallelised as well. There is no dearth of work done on the parallelisation of the sweep for the discrete ordinates transport equation. In this section, we succinctly look at some of the most salient works in the literature.

6.2.1 The Koch-Baker-Alcouffe (KBA) method

Perhaps the most famous parallel sweep algorithm in S_N neutron transport is the Koch-Baker-Alcouffe (KBA) [81, 82, 83] method – named for the authors of the seminal paper.

Considering an example domain with a given direction, Ω, of neutron travel, as shown in Fig. 6.1a, the first indexed cell element that has both incoming sides known and can be computed is cell 1. After that, either of cells 2 or 6 can be computed; then either 3, 7 or 11; and so on. In fact, this is much better represented with a graph, drawn in Fig. 6.1b. This shows the dependencies and constraints of the cell computations. Because in this case, these dependencies are in one direction only and do not loop back, this is known as a Directed Acyclic Graph (DAG).

Now, the graph shows quite clearly that some cells can be computed simultaneously, leading

to an obvious parallelisation strategy. One method is to distribute the available processes over the width of the graph, as represented in Fig. 6.1c. Each processor then has a 'column' of tasks assigned to it. Tasks on the same level – or *wavefront* – of the DAG can be computed simultaneously. Initially, some processors are idle, until it reaches a point where all processors are busy; this would be the 4th step. After that, it reaches a point again (6th step) where not all the processors are working.

As an aside, it should be noted that a cell is not necessarily just one element in the computational mesh. In fact, it is usually more advantageous to group mesh cells together into a *macrocell*. This provides the arithmetic intensity[4] needed such that the computation time is not overshadowed by communication time. This is shown in Fig. 6.1a where the underlying computational mesh has been represented for the cell indexed 1.

Before going further, we will define some helpful metrics used to evaluate the performance of a parallelised code. The *speed-up* is defined as the ratio of the serial computation wall time[5] to the parallel computation wall time. *Parallel efficiency* is a closely related metric, in that it is the speedup divided by the number of processors. This gives the speed-up, S, and parallel efficiency, η_e, as

$$S = \frac{T_s}{T_p} \ , \ \eta_e = \frac{S}{N_{\text{proc.}}} \ , \tag{6.1}$$

where $N_{\text{proc.}}$ is the number of processors, and the indices s and p denote serial and parallel respectively. There are also the theoretical equivalents of the speed-up and parallel efficiency, respectively $S_{\text{theo.}}$ and the Parallel Computational Efficiency (PCE), which are essentially S and η_e in the absence of communication costs. These will be dependent on the algorithm implemented. In Fig. 6.1, $S_{\text{theo.}} = 3.125$ and PCE ≈ 0.78. The PCE has been shown to be about a third for an infinite cubic mesh [83] and represents an upper bound on the efficiency.

While we are on the topic of efficiency, it would be interesting to briefly talk about *scalability*. This refers to the ability of the software to make efficient use of the available computational resources. For example, it is rarely the case that doubling the amount of resources halves the computational time. There are a variety of underlying factors at play such as the nature of the methods or hardware used and the amount of code that is executed sequentially, amongst others. There are two main scalability tests: strong and weak scaling. In strong scaling, the problem size is fixed while the number of CPUs increases. In the ideal case, the computational time taken will reduce proportionally to the increase in CPUs. In weak scaling, the problem

---

[4]Although we sometimes use the term loosely, strictly, arithmetic intensity is defined as the ratio of the number of floating point operations to the main memory traffic (both read and write).

[5]Wall time is the 'regular' elapsed time regardless of the number of threads, CPUs or nodes running, for example, as experienced by a wall-mounted clock – hence the name. This differs from core time which takes into account how many cores are running.
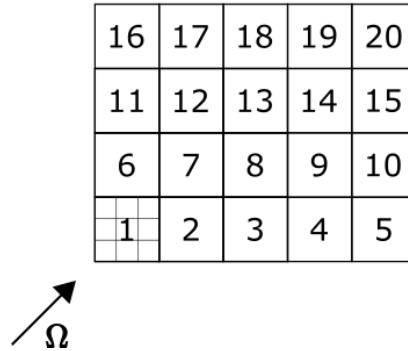
(a) Example of a 2D domain with a spatial grid of 4×5 macrocells – each macrocell being 3×3 elements of the computational mesh. The numbers in the cells merely represent a cell identifying index – *not* the computation order. The direction, $\mathbf{\Omega}$, of neutron travel is indicated.
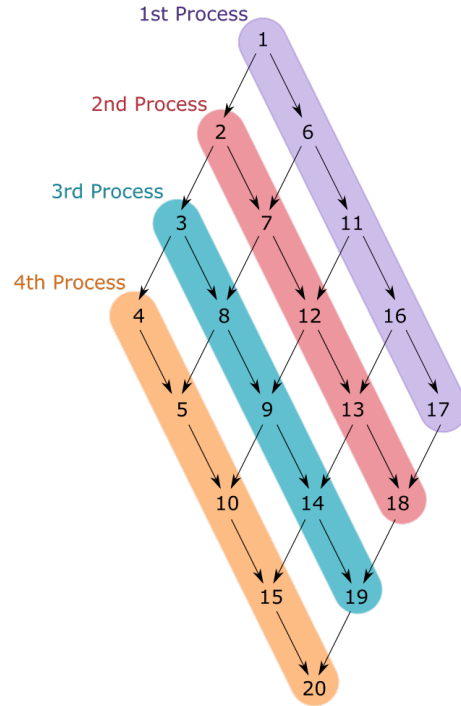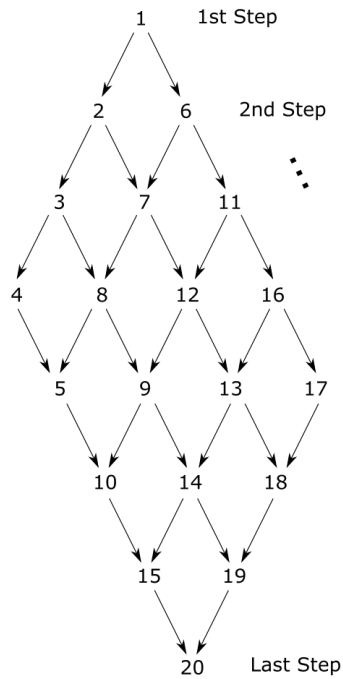
(b) Directed Acyclic Graph (DAG) for one direction showing the dependencies and constraints.

(c) Distribution of macrocells with each processor assigned a column of cells along the DAG.

Figure 6.1 Example domain and associated Directed Acyclic Graph (DAG) showing the dependencies of the sweep.

size is increased proportionally to the increase in number of CPUs, such that the calculation time should ideally stay constant. It is usually easier to achieve higher efficiency with weak scaling as the number of cores increases, as opposed to with strong scaling. [84]

Finally, another interesting metric to look at is the Processor Usage Efficiency (PUE). As previously mentioned, in the example in Fig. 6.1, some of the processors are idle for some of the time, leading to an uneven use of the computational resources. The PUE can be viewed as the number of agglomerated steps for which all processors are busy. For this given example, the PUE is $5/8 = 0.625$ for the one given direction of neutron travel.

Both the PCE and PUE can be increased by concatenating the different directions in one octant together, effectively increasing the work pipeline. In our current example, when the first processor has finished computing cell 17 for the given direction, it can start working on macrocell 1 for the next direction. This gives rise to the Koch-Baker-Alcouffe (KBA) scheme where the work for several directions (usually within the same octant) is pipelined together. If considering a number M' of directions per octant, in this case, the PUE would increase to $(2 \times M')/((2 \times M') + 3)$. A lot, if not most, of the processor idle time is eliminated with the PCE for an S_8 computation on a large 3D test case reaching 91% [85].

While we have presented here a 2D model with the processors arranged in a linear fashion, the discussion for a three-dimensional Cartesian grid is not that much more different save for the processors now arranged in a 2D virtual process grid.

6.2.2   Beyond KBA

The KBA scheme or variations on it remain the most implemented parallelisation scheme for the S_N method. While it was initially for structured two- or three-dimensional meshes, Pautz [85] extended this work to unstructured meshes in 2001. The Oak Ridge National Laboratory (ORNL) code also uses it concurrently with parallelisation over energy to provide better scalability. The code PENTRAN [86] uses a 3D virtual process grid with one dimension for each of space, angle and energy.

In 2013, Adams *et al.* [87] presented a variant of the KBA method where the sweep is parallelised over a volumetric decomposition of the spatial domain. They used a 3D process grid mapped to a volumetric spatial domain decomposition. Each process has to compute a number, A_g, of energy groups, A_m of directions and A_z of z-planes within the spatial block, before communication occurs between the processes.

More importantly, they published provably optimal scheduling algorithms for the sweep operation where all octants are computed at once, on both structured [87] and unstructured [88]

grids. They presented results showing that their algorithms allows for the completion of the sweeps in the minimum possible number of stages. They achieved weak scaling results of 60% efficiency [87] on the order of 10^5 cores, with an eight-processor calculation as the reference. They do note the use and indispensability of the STAPL [89] library which provides all the parallel containers and handles all the vectorisation and communication.

A lot of new development was also fairly recently done in the DOMINO code through the PhD work of Moustafa [90, 91, 92, 93] where a hybrid parallelisation model was implemented. They approached the problem from a bottom-up perspective and initially studied the efficiency of the SIMD vectorisation of the sweep by building on top of the EIGEN library. They showed that it was limited by the padding in the arrays that was necessary to ensure data alignment in the vector registers. However, vectorisation over the directions of neutron travel were much more successful, reaching about 63% of the peak performance of their tested CPU.

They then went on to investigate the parallelisation of the sweep using three different emerging task-based models: INTEL TBB [94], STARPU [95] and PARSEC [96]. Task-based models work with tasks, instead of lower-level threads – the task being defined as the inversion of the transport operator on a macrocell to allow for sufficient grind on data already pipelined into memory, thus going from memory bound to compute bound. These tasks are then mapped onto the hardware at runtime by an intrinsic task scheduler. Each of these investigated models was coupled with MPI for the distributed memory aspect to run on several nodes at the same time. As a result, they found that while they all performed well, they most efficient was PARSEC, reaching about 34% of the theoretical peak performance of their cluster.

Finally, they also implemented a parallelised synthetic acceleration and coupled it with their fully parallel transport solver. After validation on, amongst others, the Takeda [16] benchmarks, they were able to run a full Pressurised Water Reactor (PWR) core with 26 energy groups, comprising of about 1.02×10^{12} unknowns in a mere 45 minutes using 1536 cores.

6.3 Choice of Parallelisation Strategy

Before diving into the specifics of our current implementation, it is important to justify the choice of parallelisation strategy. Indeed, as outlined in Chap. 1, the primary objective was the investigation into and the development of a rapid S_N solver on hexagonal geometries. For this reason, we put aside (at least for this project) SIMD vectorisation. Indeed, without resorting to libraries, the complexity of array padding and vector intrinsics might not have been feasible within a reasonable timeframe.

Moreover, Moustafa [91] had shown that this was a far more suitable prospect for parallelisation over angles. On top of that, they had only vectorised the sweep for the Diamond Difference (DD) method which meant only a few floating point operations. For higher order spatial discretisation, the vectorisation of the matrix resolution (whether by Gaussian elimination or otherwise) would prove to be much more complex.

An OpenMP implementation was also carried out by Hébert [51] in DRAGON5 and investigated. However, as we were limited to about 40 cores[6] per node, investigating a parallelisation over angles and macrocells would not be easily feasible. For 3D cases, such a scenario would generate enough concurrent computations to necessitate, in general, at least 2 nodes.

Indeed, as this represented our first foray into parallelism, the main implementation goal was the parallelisation over angles and macrocells while investigating and adapting KBA to the hexagonal geometry. This does decrease the PUE but if sufficient processors are available, the angles can be distributed over different sets of processors at the same time such that the speedup is increased. However, communication times can drastically increase such that the observed speedup can be much less than the theoretical one.

Therefore, we settled on the MPI paradigm. This was chosen due to the advantage of distributed memory parallelism and the number of processors it can give access to, while remaining very much accessible and feasible.

6.4 Cartesian Implementation in WYVERN

At the time of writing, the MPI implementation resides in a separate library called WYVERN that relies on the whole DRAGON5 code library described in Sec. 2.3. For technical reasons, it was not possible to use the scripting language CLE-2000 with WYVERN, which is part of the reason for this decision. It is expected that WYVERN will be added to the DRAGON5 code repository at some point this year, along with a draft user manual.

The way the processes (also called *ranks* in MPI jargon) were organised was through the use of *communicators*. Communicators can be somewhat of a complex idea within MPI but simply put, they are a group of ranks that have an associated context, essentially a tag which facilitates communication between ranks within a communicator. The communicator which regroups all ranks at runtime is known as `MPI_COMM_WORLD`.

A rudimentary scheduler was set up using the `MPI_COMM_SPLIT` function to split the MPI communicator into subsets of communicators, as shown in Fig. 6.2. This was done using a

---

[6]The Narval cluster (which is one of the DRAC clusters) has 64 cores but default resource allocation groups have incredibly low priority, rendering it all but nearly unusable.

simple algorithm with integer division and the modulus function, as given in Alg. 3 where we used $[\cdot / \cdot]$ to represent integer division. $N_{\text{oct.}}$ represents the number of octants to be computed simultaneously while $N_{\text{dir.}}$ and $N_{\text{mcel.}}$ have similar definitions for angular directions and macrocells respectively.
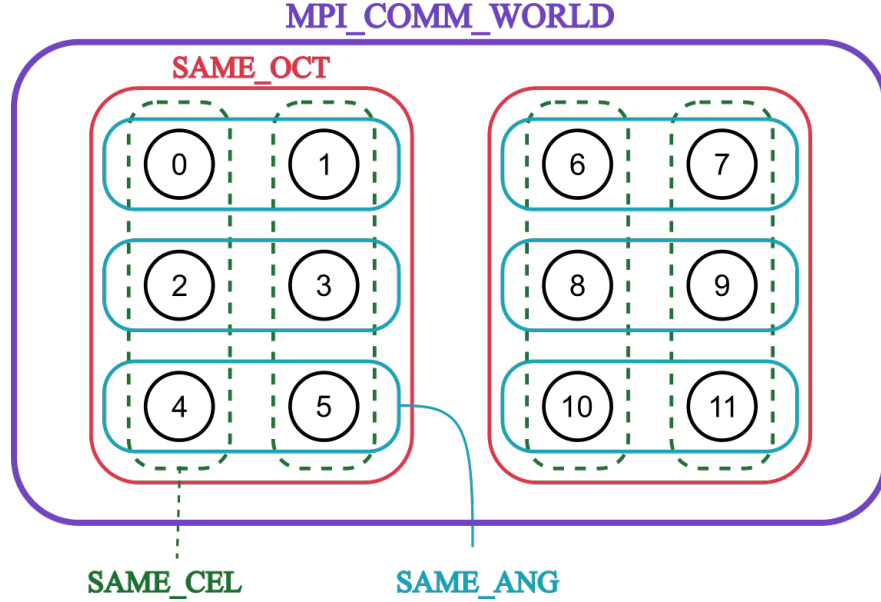
Figure 6.2 Schematic showing how the `MPI_COMM_WORLD` communicator comprising 12 ranks is split into different communicators working on either the same octant (red), the same angular direction (dashed green) or the same macrocell (blue). This is assuming a 2D test case with two macrocells along each axis, 2 octants and 3 angular directions per octant.
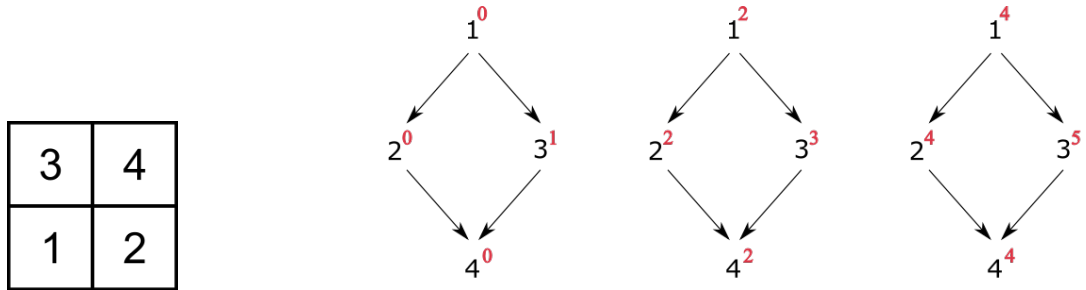
Figure 6.3 Small 2D test domain along with graphs showing sweeps for each of the three directions for the 1st octant. The process assigned to each cell is shown in red as a superscript.

Using the rank numbers, it is possible to obtain (lines 3-6 in Alg. 3) three IDs that define the position of the processes on a 3D grid. These are then used to group together processes that will be working on the same direction (`SAME_ANG`) or the same macrocell (`SAME_CEL`).

It is perhaps easier to illustrate with a simple example. Consider a small 2D domain with two

---

Algorithm 3: How different processes are distributed over a 3D virtual process grid.

---

   input : $N_{\mathrm{oct.}}$, $N_{\mathrm{dir.}}$, $N_{\mathrm{mcel.}}$
   output: IDGRID_I, IDGRID_J, IDGRID_K, SAME_OCT, SAME_CEL, SAME_ANG

   /* Obtain world rank and size.   */
1 call MPI_COMM_SIZE(MPI_COMM_WORLD, IS_WRL, IERROR)
2 call MPI_COMM_RANK(MPI_COMM_WORLD, IR_WRL, IERROR)

   /* Obtain grid IDs for each process.   */
3 IDGRID_I $= \left[\mathrm{IR\_WRL}/(N_{\mathrm{dir.}}N_{\mathrm{mcel.}})\right]$
4 IR_OCT $= \mathrm{IR\_WRL} - (N_{\mathrm{dir.}} * N_{\mathrm{mcel.}} * \mathrm{IDGRID\_I})$
5 IDGRID_J $= \mathrm{MOD}(\mathrm{IR\_OCT}, N_{\mathrm{mcel.}})$
6 IDGRID_K $= \left[\mathrm{IR\_OCT}/N_{\mathrm{mcel.}}\right]$

   /* Split into number of octants.   Line 7 below: regroup ranks with
      same tag IDGRID_I from initial communicator MPI_COMM_WORLD into new
      communicator SAME_OCT according to the order defined by IR_WRL.   */

7 call MPI_COMM_SPLIT(MPI_COMM_WORLD, IDGRID_I, IR_WRL, SAME_OCT, IERROR)
8 call MPI_COMM_RANK(SAME_OCT, IR_OCT, IERROR)

   /* Split into number of directions.   */
9 call MPI_COMM_SPLIT(SAME_OCT, IDGRID_J, IR_OCT, SAME_CEL, IERROR)
10 call MPI_COMM_RANK(SAME_CEL, IR_CEL, IERROR)

   /* Split into number of simultaneous macrocells.   */
11 call MPI_COMM_SPLIT(SAME_OCT, IDGRID_K, IR_OCT, SAME_ANG, IERROR)

---

macrocells along each axis run with S_4, *i.e.* three directions per octant, the calculation is run with twelve processes, and two octants are computed at the same time. The domain along with the graphs for one octant containing the cell indices with the rank IDs in superscript are shown in Fig. 6.3. The whole rank distribution is also given in Tab. 6.1.

Table 6.1 ID values for the different processes of the virtual process grid for a small 2D test with two macrocells along each axis, run with three directions per octant, with two octants run simultaneously.

IR_WRL	0	1	2	3	4	5	6	7	8	9	10	11
IDGRID_I	0	0	0	0	0	0	1	1	1	1	1	1
IR_OCT	0	1	2	3	4	5	0	1	2	3	4	5
IDGRID_J	0	1	0	1	0	1	0	1	0	1	0	1
IDGRID_K	0	0	1	1	2	2	0	0	1	1	2	2

A few things to note here :

- The implementation is still somewhat in its infancy stages, so the exact number of processes needed must be passed along at runtime to ensure everything works correctly. Too few or too many may result in unexpected behaviour.

- Moreover, in its current form, the implementation is rigidly *synchronous*, resulting in computational times that quickly level off with increasing number of cores used.

- While the parallelisation over octants was implemented, it is seldom, if ever, used. The calculations were run on the Digital Research Alliance of Canada (DRAC) clusters where our research group only has a default resource allocation. Requesting a few nodes for a computation of a few hours or a day already entails a waiting period of days at times. Parallelisation over octants would require more nodes and hence, more waiting. For this reason, we limit ourselves to a study over angles and macrocells.

Finally, Alg. 4 shows how the backbone of our initial (Alg. 1) inner iteration changes. This is represented in 2D but similar changes were done for the 3D case. We wish to highlight the following points:

- How we handle reflective or albedo boundary conditions are not represented in an effort to help readability but it is fairly straightforward to manage.

- The number of macrocells along each axis is assumed constant to simplify things. Also, the number of wavefronts in 2D is calculated using $n_{\text{wave.}} = N_{\text{mcel.}} + N_{\text{mcel.}} - 1$.

- The macrocell indices within the domain can be deduced implicitly using the wavefront index, $i_{\text{wave.}}$ and $N_{\text{mcel.}}$.

- It can be seen that the changes to the code, while strategically placed, remain minimal. Using the virtual process grid indices (`IDGRID_I`, `IDGRID_J`, and `IDGRID_K`) and the number of octants, macrocells and angles ($N_{\text{oct.}}$, $N_{\text{mcel.}}$, and $N_{\text{dir.}}$) over which to parallelise, it is possible to limit which process enters which loops.

Algorithm 4: Representation of one parallelised inner iteration for the 2D case, as implemented in WYVERN using MPI.

$\quad$ **input** : $Q_{l,i,j}^{m,[\alpha,\beta]}$, $\varphi_{n,BC}^{[\alpha]}$, $\varphi_{n,BC}^{[\beta]}$
$\quad$ **output:** $\phi_{l,i,j}^{m,[\alpha,\beta]}$, $\varphi_{n,BC}^{[\alpha]}$, $\varphi_{n,BC}^{[\beta]}$

1 **for** $i_{\text{oct.}} = (1 + \texttt{IDGRID\_I}) : N_{\text{oct.}} : 4$ **do**

$\quad$ /* Loop over wavefronts. $\qquad\qquad\qquad\qquad\qquad\qquad\qquad$ */
2 $\quad$ **for** $i_{\text{wave.}} = 1 : 1 : n_{\text{wave.}}$ **do**

$\quad\quad$ /* Loop over macrocells in each wavefront. $\qquad\qquad\qquad\quad$ */
3 $\quad\quad$ **for** $i_{\text{mcel.}} = (1 + \texttt{IDGRID\_J}) : N_{\text{mcel.}} : cells/wavefront$ **do**

$\quad\quad\quad$ /* Loop over directions in one octant. $\qquad\qquad\qquad\qquad$ */
4 $\quad\quad\quad$ **for** $i_{\text{dir.}} = (1 + \texttt{IDGRID\_K}) : N_{\text{dir.}} : N_{\text{dir.}}$ **do**

$\quad\quad\quad\quad$ /* Loop over elements in macrocell. $\qquad\qquad\qquad\qquad$ */
5 $\quad\quad\quad\quad$ **for** $i = 1 : 1 : l_x/N_{\text{mcel.}}$ **do**

6 $\quad\quad\quad\quad\quad$ **for** $j = 1 : 1 : l_y/N_{\text{mcel.}}$ **do**

7 $\quad\quad\quad\quad\quad\quad$ Solve system of equations $\mathbb{T}\Psi_{n,i,j} = \mathbb{Q}_{n,i,j}$.

8 $\quad\quad\quad\quad\quad\quad$ Compute the outgoing fluxes, $\varphi_{n,i+}^{[\beta]}$ and $\varphi_{n,j+}^{[\alpha]}$.

9 $\quad\quad\quad\quad\quad\quad$ Store BC in temp. variables, $\Upsilon_{n,i-}^{[\beta]} \leftarrow \varphi_{n,i+}^{[\beta]}$, $\Upsilon_{n,j-}^{[\alpha]} \leftarrow \varphi_{n,j+}^{[\alpha]}$

10 $\quad\quad\quad\quad\quad\quad$ Sum to obtain scalar flux $\phi_{l,i,j}^{m,[\alpha,\beta]} = \phi_{l,i,j}^{m,[\alpha,\beta]} + 2\omega_n \psi_{l,i,j}^{m,[\alpha,\beta]}$.

11 $\quad\quad$ call `MPI_ALL_REDUCE` on $\Upsilon_{n,i-}^{[\beta]}$ and $\Upsilon_{n,j-}^{[\alpha]}$.

12 $\quad\quad$ $\varphi_{n,BC}^{[\beta]} \leftarrow \Upsilon_{n,i-}^{[\beta]}$

13 $\quad\quad$ $\varphi_{n,BC}^{[\alpha]} \leftarrow \Upsilon_{n,j-}^{[\alpha]}$

14 call `MPI_ALL_REDUCE` on $\phi_{l,i,j}^{m,[\alpha,\beta]}$

6.4.1 3D-TAK2 benchmark

We had noticed with 2D test cases (not presented) that the increase in speed-up scales better with increasing number of processors for higher S_N orders and higher spatial discretisation

order, Λ. This was not surprising as the arithmetic intensity would increase with these two parameters such that calculation times would be less overshadowed by communication time.

We present in this section results for the 3D-TAK2 benchmark with S_6 quadrature and a mesh refinement of 3. We do not apply any synthetic acceleration at this point to see the speed-up thanks to only the parallelisation. The results, run on the DRAC cluster, BELUGA, are presented in tabular form (Tab. 6.2) and plotted in Fig. 6.4. We found that this presentation made it easier to see clearly the number of processors and how they were distributed over the virtual grid. High Order Diamond Difference (HODD) is used for $\Lambda = 0$, while the rest are run using Discontinuous Galerkin Finite Element Method (DGFEM). It should be emphasised that within each order, Λ, these results correspond to strong scaling tests.

Tab. 6.2 refer to two theoretical values: $S_{\text{theo.}}^{\text{mcel.+ang.}}$ and $t_{\text{theo.}}$, and two experimentally measured values: $t_{\text{meas.}}$ and $S_{\text{meas.}}^{\text{mcel.+ang.}}$. $S_{\text{theo.}}^{\text{mcel.+ang.}}$ is the expected speed-up from the parallelisation over macrocells and angles. While speed-up from parallelisation over angles, $S_{\text{theo.}}^{\text{ang.}}$, is simply the number of directions per octant[7], the speed-up from macrocells, $S_{\text{theo.}}^{\text{mcel.}}$, is given by

$$S_{\text{theo.}}^{\text{mcel.}} = \frac{\text{number of macrocells in domain}}{\text{number of wavefronts}} \ . \tag{6.2}$$

The overall theoretical speed-up is then

$$S_{\text{theo.}}^{\text{mcel.+ang.}} = S_{\text{theo.}}^{\text{mcel.}} \times S_{\text{theo.}}^{\text{ang.}} \ , \tag{6.3}$$

and the $t_{\text{theo.}}$ is the serial computation time multiplied by $S_{\text{theo.}}^{\text{mcel.+ang.}}$, and represents the expected result in the absence of communication time.

On the other hand, $t_{\text{meas.}}$ refers to the experimentally measured computational time for each calculation, and $S_{\text{meas.}}^{\text{mcel.+ang.}}$ is calculated using that and the sequential time.

It can be seen that while for low number of processes, the observed speedup is pretty close to the theoretical speedup, this quickly changes as the number of processes increases. This is partly due to the communication time, but also probably to the unoptimised code.

Finally, a note on running calculations on DRAC clusters: we observed during our experiments that the same calculations run at different points in time yielded slightly different results – within the realm of about 10-15%. While we would have liked to run all computations about five to 10 times, established an average and standard deviation, again, with our limited resource allocation, this would not have been realistic – especially given our priority

---

[7]Recall, at this point in time, one has to provide exactly the number of processes to fit the number of direction per octant if one wishes to parallelise the computation over angles.

had plummeted even further after all the testing and development.

Looking over the plots in Fig. 6.4, we can see quite clearly that indeed, as the order – and hence, arithmetic intensity – increases, the efficiency of implementation remains higher for longer. Overall, while we are only able to see a decrease of about six times for HODD-0, for DG-3, we observe nearly 42× with 182 cores. Again, recall that these are strong scaling tests within each order Λ. As such, having measured speed-ups that are more than 50% than the theoretical speed-ups on 162 cores is not bad for a first implementation.

The plots do also indicate more clearly something that we mentioned: that the parallelisation scheme as it has been implemented has a parallel efficiency that degrades rapidly, as exemplified by the non-linear nature of the plots (both theoretical and experimental).

6.5 Hexagonal Implementation in WYVERN

Hexagons have a three-fold connectivity between each other. While this is definitely more regular than an unstructured grid, it is more complex than an orthogonal grid. As such, this requires appropriate handling. Having already explained quite thoroughly our Cartesian implementation of the parallelisation, in section, we will focus on what makes the hexagonal implementation different.

6.5.1 Parallel hexagonal sweep algorithm and implementation details

A hexagonal planar grid has the marked difference of having three-way dependencies between each hexagonal element, unlike two-way dependencies in a flat Cartesian grid. This means that three incoming sides need to be known before being able to compute a hexagon, implying that for inner hexagons, three hexagons need to be computed before they can be processed. This has two direct consequences:

1. The Directed Acyclic Graph (DAG) is constricted throughout the sweep such that, compared to a Cartesian grid of similar number of macrocells, less cells may be computed at the same time. This is shown in Fig. 6.5, with an example domain and its associated DAG.

2. On the hexagonal plane, the number of macrocells is not variable like it is within a Cartesian domain. Indeed, the hexagons cannot be regrouped such that one process solves all the hexagons in that group before communication between processes happens again. We do not offer a mathematical proof but it seems that, without fail, some hexagons would need information from other processes. Without resorting to domain

Table 6.2 Summarised results obtained using Wyvern for the 3D-TAK2 benchmark presented in Chap. 3 with S_6 quadrature and mesh refinement of 3. Note that the 'series' data point for $\Lambda = 3$ is an estimated value.

Λ	$N_\text{mcel.}$	$N_\text{proc.}$	proc. grid	$S_\text{theo.}^\text{KBA+ang.}$	$t_\text{theo.}$ (s)	$t_\text{meas.}$ (s)	$S_\text{meas.}^\text{KBA+ang.}$
0	series	1	1	-	455	455	-
	ang. only	6	6×1	6	75.8	105	4.33
	2	18	6×3	12	37.9	74	6.15
1	series	1	1	-	3520	3520	-
	ang. only	6	6×1	6	587	799	4.41
	2	18	6×3	12	293	666	5.29
	3	36	6×6	23.2	152	504	6.98
2	series	1	1	-	55800	55800	-
	ang. only	6	6×1	6	9300	11900	4.69
	2	18	6×3	12	4650	8680	6.43
	3	36	6×6	23.2	2410	4730	11.8
	6	162	6×27	81	689	2110	26.5
3	series	1	1	-	591300	591300	-
	ang. only	6	6×1	6	98600	111000	5.33
	2	18	6×3	12	49300	76900	7.69
	3	36	6×6	23.2	25500	42800	13.8
	6	162	6×27	81	7300	14100	41.9
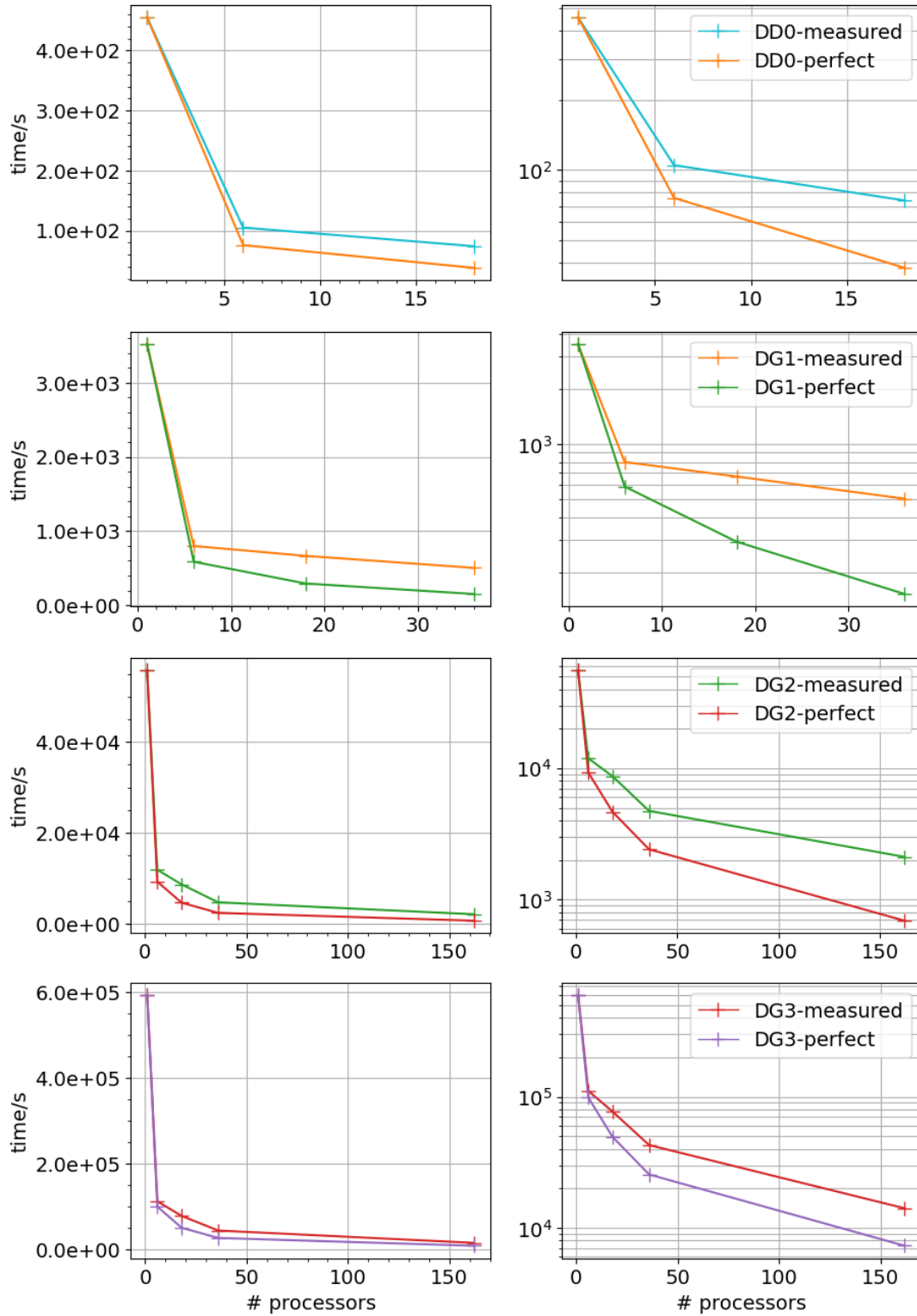
Figure 6.4 Time against number of processors obtained using WYVERN for the 3D-TAK2 benchmark presented in Chap. 3. The plots on the right are the same as those on the left, except with a logarithmic y scale. The first plot point is the series calculation, the next is parallelisation over angles only and the subsequent ones represent parallelisation over angles and an increasing number of macrocells along each of the three cardinal axes, in the order: 1, 2, 3, 6. The 'perfect' curves refer to expected computation times in the absence of communication times.

decomposition techniques, we have only been able to have the number of hexagons as the number of macrocells. Grouping some hexagons together does not appear possible.

This is slightly different for a 3D hexagonal case though. It is possible to have a variable number of macrocells along the z axis. While this ultimately allows for better efficiencies, we expect the results to be worse than with Cartesian geometries.

Also, while we had initially implemented a column-like sweep within the hexagonal domain (as illustrated previously in Fig. 4.4), Fig. 6.5 shows clearly that the sweep for a parallelisation is different. The graph subroutine we had previously written had to be adapted to take that into consideration.

Lastly, in the two-dimensional Cartesian sweep, two arrays corresponding to one column and one row of the domain of surface boundary fluxes are kept in memory to propagate the sweep – the sizes of which do not change. Indeed, as the sweep progresses, information in these arrays are continuously being overwritten as they are not needed beyond each current sweep. However, for the hexagonal sweep, consider Fig. 6.6. If the orange wavefront is to be computed, information is needed from the purple wavefront *as well as* the blue wavefront, which is two wavefronts prior.

We do not provide an algorithm here as beyond the latter point, it is extremely similar to what we have seen Alg. 2 and Alg. 4.

6.5.2 **3D-TAK4 benchmark**

Results for parallelised computations of the 3D-TAK4 benchmark using an S_6 DG-3 configuration are presented here to demonstrate the possible gains in calculation wall time.

Indeed, Tab. 6.3 and Fig. 6.7 show the unaccelerated results, while Tab. 6.4 and Fig. 6.8 summarise accelerated (using RT-SP_nSA with the recommended parameters of Chap. 5) results for mesh refinements of 1 and 2. All discussions around the theoretical speed-up values and DRAC from the previous section apply here too. And, again, the results essentially correspond to strong scaling tests when comparing for the same Λ values.

While expected, it is comforting to note that the use of the synthetic acceleration does not really affect the performance of the parallelisation implementation. Also, the submeshed benchmark offered slightly higher speed-ups that the unrefined one – again, an anticipated result. Less anticipated is the fact that for a similar number of processes, the speed-up for the hexagonal benchmark is higher than the Cartesian one – about nearly one and a half as much. It would suggest than an uneven distribution of macrocells along the geometry could be helpful.
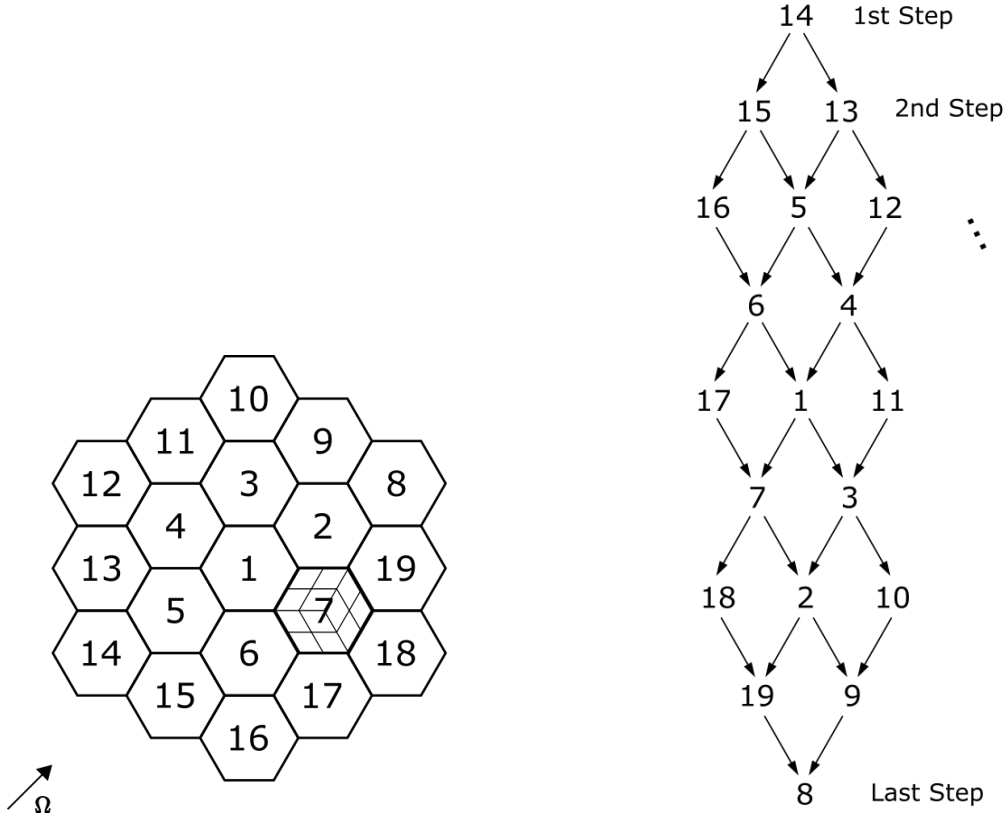
Figure 6.5 Left: example of a 2D hexagonal domain with 19 hexagons, each considered a macrocell – one macrocell being represented as 3×4 elements of the computational mesh. Right: DAG for one direction showing the dependencies and constraints.

Table 6.3 Summarised results obtained using WYVERN for the 3D-TAK4 benchmark presented in Chap. 4 with S_6 DG-3 and no mesh refinement. Note that the 'series' data point is an estimated value.

$N_{\text{mcel.}}$	$N_{\text{proc.}}$	proc. grid	$S_{\text{theo.}}^{\text{KBA+ang.}}$	$t_{\text{theo.}}$ (s)	$t_{\text{meas.}}$ (s)	$S_{\text{meas.}}^{\text{KBA+ang.}}$
series	1	1	-	340000	340000	-
ang. only	4	4×1	4	85100	127000	2.68
1	32	4×8	23.3	14600	16100	21.1
2	60	4×15	45.1	7550	8860	38.3
3	92	4×23	65.4	5200	6640	51.2
6	180	4×45	119	2850	4700	72.4
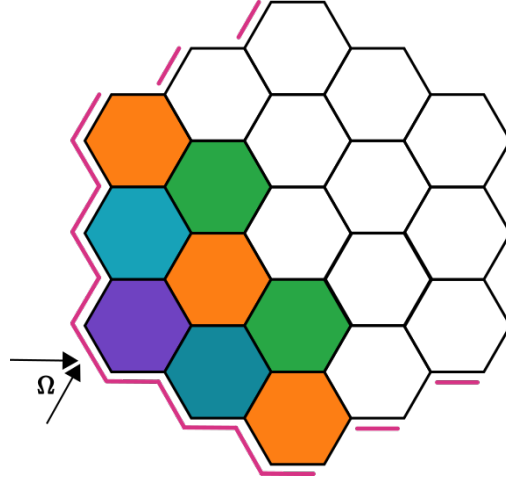
Figure 6.6 Wavefront sweep showing the need for information from two wavefronts prior for the resolution of the blue wavefront.

6.5.3 Mock Fast Breeder Reactor (FBR) core: 3D-FBR

Finally, in this section, a mock Fast Breeder Reactor (FBR) core is simulated. The domain is given in Fig. 6.9 with the material description in Tab. 6.6. This is a previously unpublished benchmark similar to the core investigated by Bay [15]. SAPHYB-formatted 33-group multiparameter cross-section libraries were obtained using 295-group DRAGON5 simplified colorset calculations. All the information and libraries can be found on the official DRAGON5 distribution website [51] – more specifically, in the `Donjon/data/fbr_core_proc` directory.

The idea here is to run a test case that is as close as possible to a full-core calculation usually run in the industry. These are usually so demanding that it would take days to run, if not more than a month. The test case with 1.02×10^{12} unknowns run by Moustafa [91] in 45 minutes was actually a PWR benchmark with 26 energy groups, and linear anisotropy.

3D-FBR is a 33-energy-group benchmark with 547 hexagons on the radial plane, meant to be run at a quartic order of anisotropy. Unfortunately, because of limits in the amount of memory that was available per node, for such a big benchmark, we could only run at linear anisotropy. Indeed, while this does not affect the number of unknowns to be solved for, it greatly impacts the amount of memory used for storing the spherical harmonic moments of the flux. We also had to reduce the recommended of total sublayers from 23 to 12.

The results are given in Tab. 6.5, where the number of unknowns, $N_{\text{unk.}}$, is calculated using

$$N_{\text{unk.}} = N_{\text{hex.}} \times 3 \times N_{\text{z}} \times N_{\text{g}} \times N_{\text{dir.}} \times \Lambda^3 \,, \tag{6.4}$$
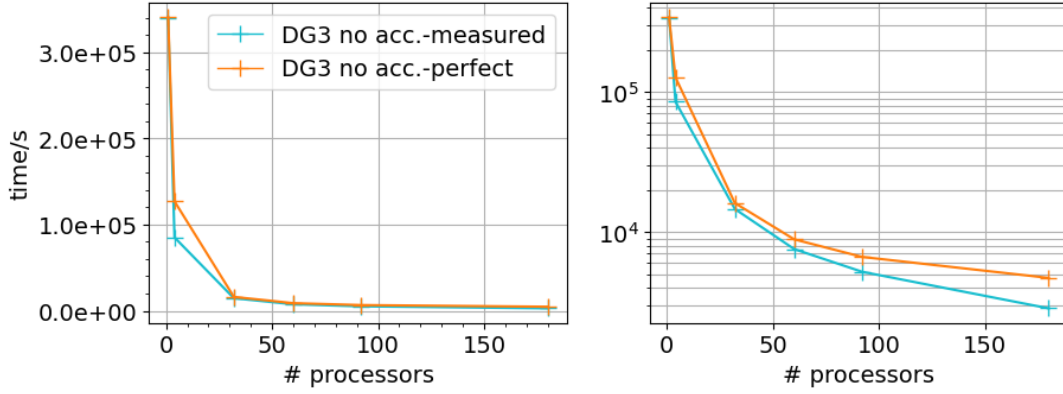
Figure 6.7 Time against number of processors obtained using WYVERN for the 3D-TAK4 benchmark presented in Chap. 4. Calculations run with S_6 DG-3 no lozenge mesh refinement and no acceleration. The plots on the right are the same as those on the left, except with a logarithmic y scale. The first plot point is the series calculation, the next is parallelisation over angles only and the subsequent ones represent parallelisation over angles and an increasing number of macrocells along the z axis, in the order: 1, 2, 3, 6. The 'perfect' curves refer to expected computation times in the absence of communication times.

Table 6.4 Summarised results obtained using WYVERN for the 3D-TAK4 benchmark presented in Chap. 4 with S_6 DG-3 and RT-SP_nSA with the recommended parameters from Chap. 5, for two different lozenge mesh refinements. Note that the 'series' data points are estimated values.

subm.	$N_{\text{mcel.}}$	$N_{\text{proc.}}$	proc. grid	$S_{\text{theo.}}^{\text{KBA+ang.}}$	$t_{\text{theo.}}$ (s)	$t_{\text{meas.}}$ (s)	$S_{\text{meas.}}^{\text{KBA+ang.}}$
1	series	1	1	-	78500	78500	-
	ang. only	4	4×1	4	19600	29200	2.69
	1	32	4×8	23.3	3370	3850	20.4
	2	60	4×15	45.1	1740	2150	36.5
	3	92	4×23	65.4	1200	1600	49.1
	6	180	4×45	119	658	1130	69.5
2	series	1	1	-	314000	314000	-
	ang. only	4	4×1	4	78500	117000	2.68
	1	32	4×8	23.3	13500	13829	22.7
	2	60	4×15	45.1	6970	7660	41.0
	3	92	4×23	65.4	4800	5580	56.3
	6	180	4×45	119	2630	3540	88.7
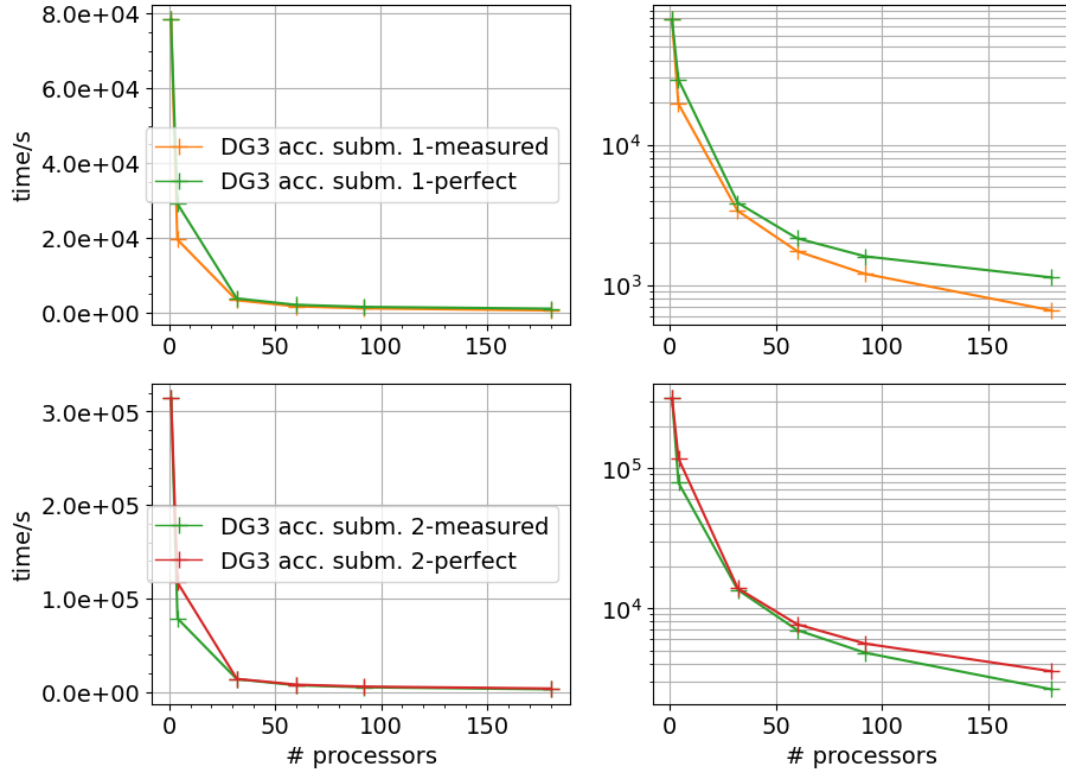
Figure 6.8 Time against number of processors obtained using WYVERN for the 3D-TAK4 benchmark presented in Chap. 4. Calculations run with S_6 DG-3 and RT-SP_nSA with recommended values from Chap. 5. The plots on the right are the same as those on the left, except with a logarithmic y scale. The first plot point is the series calculation, the next is parallelisation over angles only and the subsequent ones represent parallelisation over angles and an increasing number of macrocells along the z axis, in the order: 1, 2, 3, 6. The 'perfect' curves refer to expected computation times in the absence of communication times.

where $N_{hex.}$, N_z and N_g are the number of planar hexagons, z-layers, and energy groups respectively. This means that, for example, at an S_{10} DG-1 configuration without any lozenge mesh refinement, this problem stands at 6.23×10^8 unknowns.

Tab. 6.5 shows again that with increasing Λ, the measured speed-ups get increasingly closer to the theoretical speed-ups. In this case however, the effect is more pronounced than for the 3D-TAK2 benchmark. This is possibly due to the fact that the planar domain size is bigger than previously seen and also the fact that there are less z-layers compared to the 3D-TAK4 benchmark, meaning that the macrocells are effectively smaller in this benchmark. Hence, until there is enough arithmetic intensity (such as in DG-3), the measured speed-ups are unfortunately even less than 50% of the theoretical ones.

For example, sequentially, the S_{10} DG-1 calculation was 52400 s with RT-SP_nSA and estimated at 102000 s without. With WYVERN and RT-SP_nSA, using 154 cores, this ran in just over 75 minutes, representing a speed-up of 11.6$\times$ over the accelerated sequential run – which is significant, but very much under the expected speed-up (see Tab. 6.5).

That being said, running S_2 DG-3 in series took 748000 s without the Synthetic Acceleration (SA) and 374000 s with. This allowed us to estimate that for the accelerated sequential S_{10} DG-3 calculation, it would take nearly 65 days, and probably more than four months without SA. Using WYVERN and RT-SP_nSA, the same S_{10} DG-3 calculation took just over a day using 154 cores. This is a speed-up of over sixty times on the accelerated sequential run.

6.6   Concluding Remarks

A preliminary framework for the parallelisation of the sweep in Cartesian and hexagonal geometries was set up in the DRAGON5 code through a separate library named WYVERN. This was done using a KBA-style (adapted in the case of hexagonal geometries) approach but without the use of pipelining, such that the parallelisation was over both macrocells and angles. The parallelisation over octants was also set up but not tested due to a lack of resources. This was achieved by distributing processes over a 3D virtual process grid using mostly the `MPI_COMM_SPLIT` function of MPI API. Together with the synthetic acceleration, we were able to model a Fast Neutron Reactor (FNR) mock core with 4.99×10^9 unknowns (in one particular configuration) in just under 25 hours, using 154 cores – down from an estimated unaccelerated sequential runtime of around four months.

Table 6.5 Accelerated sequential and parallelised results obtained using DRAGON5 and WYVERN respectively for the 3D-FBR benchmark, with RT-SP_nSA using the recommended parameters from Chap. 5. Note that the data in *italic* are estimated values.

S_N	Λ	$N_{\text{unk.}}$	$N_{\text{mcel.}}$	$N_{\text{proc.}}$	proc. grid	$S_{\text{theo.}}^{\text{KBA+ang.}}$	$t_{\text{meas.}}$ (s)	$S_{\text{meas.}}^{\text{KBA+ang.}}$
6	1	2.49×10^8	series	1	1	-	19100	-
			1	56	4×14	41.3	4740	4.03
	2	8.42×10^8	series	1	1	-	*249600*	-
			1	56	4×14	41.3	16100	15.5
	3	1.99×10^9	series	1	1	-	*2244000*	-
			1	56	4×14	41.3	96900	23.2
			2	108	4×27	81.0	68700	32.7
10	1	6.23×10^8	series	1	1	-	52400	-
			1	154	11×14	114	4510	11.6
	2	2.11×10^9	series	1	1	-	*624000*	-
			1	154	11×14	114	15600	40.0
	3	4.99×10^9	series	1	1	-	*5610000*	-
			1	154	11×14	114	89500	62.6

Table 6.6 Description of the colour representation for each region in the 3D-FBR benchmark, given in Fig. 6.9.

Region	Description
	Void
	Internal fissile fuel assembly
	External fertile fuel assembly
	Internal fertile fuel assembly
	COSU control rod
	COSV control rod follower
	ARRE shutdown rod
	ARSV shutdown rod follower
	REFS reflector
	PLNA sodium plenum
	VSXP expansion bellow
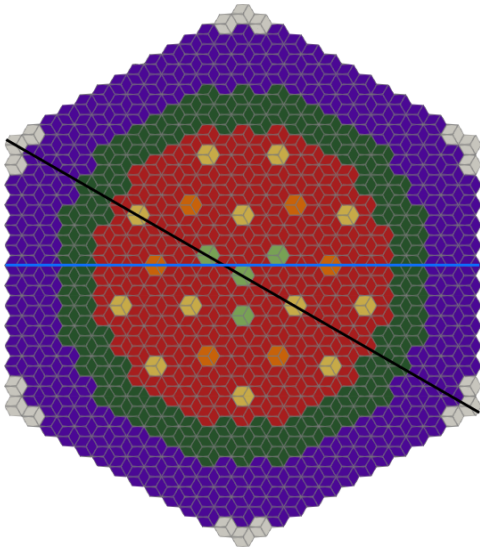
(a) Cut-through view at $z = 150$ cm.
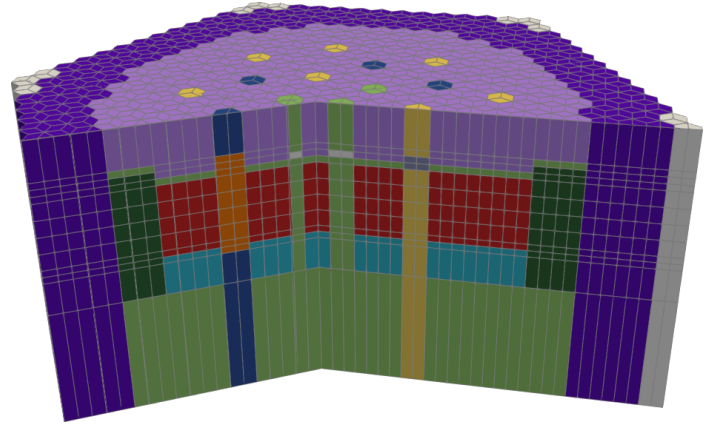
(b) Overall view with part removed.
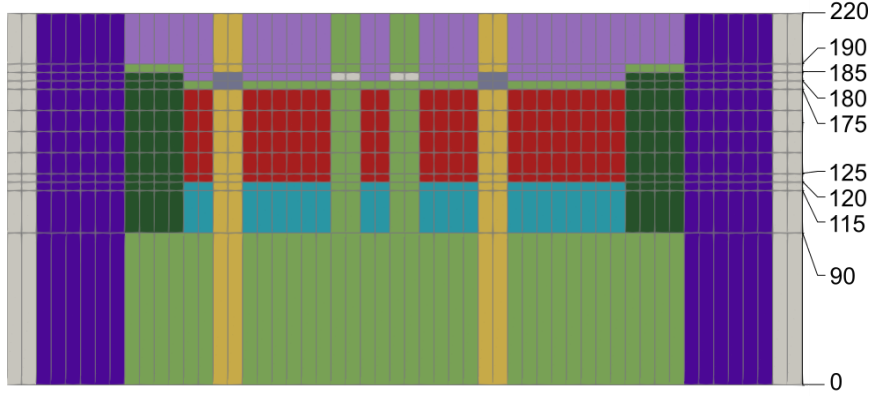
(c) Cut-through side view 1: through black line.
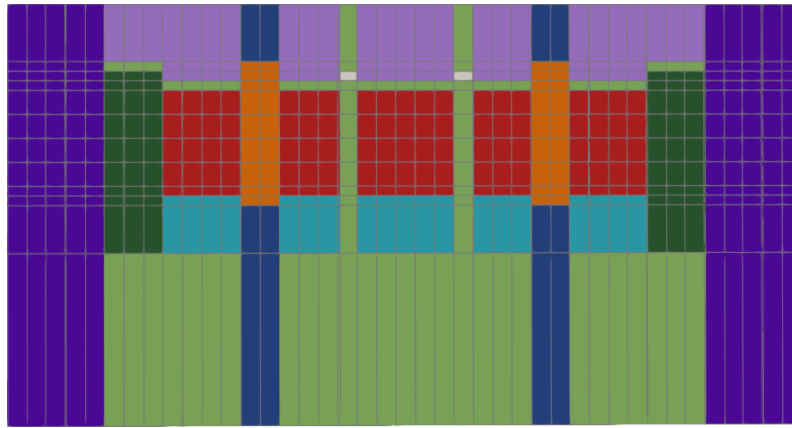
(d) Cut-through side view 2: through blue line.

Figure 6.9 Domain of the 3D-FBR benchmark at various angles. Dimensions are in cm; side of one hexagon is 10.104 cm. Vacuum boundary conditions are applied to the whole outer edge of the domain. The initial computational mesh before subsequent refinement is shown in all subfigures. Refer to Tab. 6.6 for a description of the materials. (A small mistake slid through in the input geometry file for the presented results: it should have been the the light green mix and not a void (light grey mix), around the top centre of the domain.)

CHAPTER 7   CONCLUSION

This chapter summarises the work done through this dissertation as well as the major findings. Some possible avenues for continuing the research are also suggested.

7.1   Conclusion and Findings

This work was initially inspired by the results of Bay [15] who found that SP_n solvers lacked when it came to modelling Fast Neutron Reactors (FNRs) correctly, and that S_N solvers, while very precise and accurate, were around $100 \times$ slower. From this result, we set out to develop a parallel S_N solver using discontinuous finite elements that would be able to bridge this gap.

This goal required a few interlocking parts that had to be completed before it would be achieved. We first set out to implement a Discontinuous Galerkin Finite Element Method (DGFEM) solver in DRAGON5 and compared it with the High Order Diamond Difference (HODD). While the DGFEM method is not new by any means, this is a new solver capability in the code upon which new methods can be added.

Also, there was and still is a certain belief that DGFEM is just better. This is not what we found. In the eigenvalue-problem benchmarks we investigated, we found that once fully converged, both methods agree to a few pcm, taking about the same number of iterations and computational time. Moreover, very often, HODD would yield a more accurate result at a lower mesh refinement than DGFEM would. While it is true, as demonstrated by Schunert [34], that DG is slightly better at dealing with highly anisotropic meshes or optically thick regions, the differences are not, in our opinion, enough to discard HODD as a method.

The next rung on the ladder was adapting the solvers to hexagonal geometries. We outlined the reasoning behind the choice to use a lozenge-based submeshing of the hexagon. The ease of implementation has been more formally shown. We personally found that this had been lacking in previous literature. The implementation entails some subtleties which we highlight in this document. Furthermore, we took the opportunity to compare the two spatial discretisation schemes again on this new geometry using the well-known Takeda benchmarks [16]. Both methods performed very well getting within a few pcms of the Monte-Carlo reference value. It should be noted that a lot of new reactor and Gen-IV designs are hexagonal. Being able to model them could allow us to carry out more advanced reactor studies.

At this point, we sought to use the well-established synthetic acceleration technique to ac-

celerate our computation. While this is usually based on a diffusion solver, having access to an SP_n solver allowed us to apply it as a synthetic acceleration and test parameters such as the SP_n order and the source anisotropy. While it was a novel study, we ultimately found out that using SP_1 with an isotropic source and applying the acceleration every two to four transport iterations yielded the best results.

Moreover, this particular solver had been spatially discretised using Raviart-Thomas (RT) finite elements. As this was the first application of this spatial discretisation to either HODD or DGFEM, we presented a one-dimensional Fourier Analysis (FA) to investigate the consistency of the discretisation schemes. While not unconditionally stable, we showed that it accelerated our computations from about 60% and up to 80% in a lot of cases. This was in large part thanks to a novel algorithm for dealing with reflective or albedo boundary conditions as well as the implementation of a new flux correction method. In the latter, the zeroth moment is applied as a scaled fraction to higher moments, allowing low-order (spatially) SP_n calculations to accelerate high order transport iterations.

Finally, using the Message Passing Interface (MPI) Application Programming Interface (API) and a Koch-Baker-Alcouffe (KBA)-style method, the transport sweep was parallelised in the S_N solvers. This was done by creating a 3D virtual process grid distributing the processes over octants, macrocells and angles. The KBA sweep was also investigated for hexagonal geometries. While very similar to the Cartesian sweep, some differences were highlighted and explained. This was implemented in a separate library called WYVERN that will hopefully become part of the larger DRAGON5 code.

While the solvers are far from being optimised, gains of up to 80× could be observed on 3D hexagonal benchmarks. Coupled with the synthetic acceleration, WYVERN was able to model a mock full Fast Breeder Reactor (FBR) core with about 4.99×10^9 unknowns in about 25 hours using 154 cores, down from an expected 65 days. In this first exploratory work, the focus of implementation has been more on speed rather than efficiency. While this leads to a somewhat uneven – and sometimes even rather poor – use of resources, the aim was to provide a first step proof-of-concept.

7.2   Future Work

It is safe to say that there are many other avenues for exploration within this work. We had hoped to investigate quite a few of them but could not, for lack of time.

The HODD and DGFEM solvers could be tested with the serependity family of polynomials and investigate the difference with the Lagrange family currently implemented. This could

represented a substantial decrease in degrees of freedom (d.o.f.) and hence computation time. Variable-order expansion for the polynomials in different dimensions or hierarchical bases could also be experimented with.

However, the area with definitely the most untapped potential is the parallelism. As we mentioned, the High Performance Computing (HPC) landscape has become incredibly diverse and we have only explored a small space. To begin with, our current implementation could probably benefit a lot from a hybrid MPI + OpenMP model. Indeed, in hindsight, taking advantage of shared memory parallelism on nodes could have helped with runtimes but mostly with the issue of available memory on nodes when trying to run huge real-world simulations.

Furthermore, the way the parallelisation is currently programmed is quite rigid and decidedly synchronous. This results in very limited efficiency that levels off quickly. Reviewing the parallelisation strategy from a bottom-up approach similar to the work of Moustafa [91] would yield a lot of performance gains. This would probably entail a vectorised distributed task-based approach. However, if the code is properly vectorised and the algorithm is significantly rethought (using for example, domain decomposition approaches), the sheer computing power of Graphic Processing Units (GPUs) could be harnessed as well.

REFERENCES

[1] World Nuclear Association, *Outline History of Nuclear Energy*, 2020. [Online]. Available: `https://world-nuclear.org/information-library/current-and-future-generation/outline-history-of-nuclear-energy.aspx` (visited on 07/19/2022).

[2] W. M. Stacey, *Nuclear Reactor Physics*, 2nd ed. Wiley-VCH, 2007.

[3] World Nuclear Association, *Plans For New Reactors Worldwide*, 2022. [Online]. Available: `https://www.world-nuclear.org/information-library/current-and-future-generation/plans-for-new-reactors-worldwide.aspx`.

[4] *The Natrium technology: Providing reliable, carbon-free energy to complement wind and solar*, 2021. [Online]. Available: `https://www.ans.org/news/article-2782/the-natrium-technology-providing-reliable-carbonfree-energy-to-complement-wind-and-solar/` (visited on 08/01/2022).

[5] Canadian Nuclear Safety Commission (CNSC), *Phase 1 Pre-Licensing Vendor Design Review Executive Summary: ARC Nuclear Canada Inc.* 2022. [Online]. Available: `https://nuclearsafety.gc.ca/eng/reactors/power-plants/pre-licensing-vendor-design-review/arc-nuclear-canada-executive-summary.cfm` (visited on 04/10/2022).

[6] Financial Post, *ARC Canada Closes $30 Million Series A Financing*, 2022. [Online]. Available: `https://financialpost.com/pmn/press-releases-pmn/business-wire-news-releases-pmn/arc-canada-closes-30-million-series-a-financing` (visited on 04/19/2022).

[7] World Nuclear News, *Minister foresees 2022 completion date for Indian FBR*, 2021. [Online]. Available: `https://world-nuclear-news.org/Articles/Minister-foresees-2022-completion-date-for-Indian` (visited on 01/05/2022).

[8] World Nuclear News, *Chinese fast reactor completes full-power test run*, 2014. [Online]. Available: `http://www.world-nuclear-news.org/NN-Chinese-fast-reactor-completes-full-power-test-run-1912144.html` (visited on 02/28/2017).

[9] A. l. Izhutov *et al.*, "Prolongation of the BOR-60 reactor operation", *Nuclear Engineering and Technology*, vol. 47, no. 3, pp. 253–259, 2015, ISSN: 17385733. DOI: `10.1016/j.net.2015.03.002`.

[10] World Nuclear Association, *Nuclear Power in Russia*, 2021. [Online]. Available: `https://world-nuclear.org/information-library/country-profiles/countries-o-s/russia-nuclear-power.aspx` (visited on 07/10/2022).

[11] *Welcome to Generation IV International forum*, 2020. [Online]. Available: `https://www.gen-4.org/gif/`.

[12] World Nuclear Association, *Generation IV Nuclear Reactors*, 2020. [Online]. Available: `https://world-nuclear.org/information-library/nuclear-fuel-cycle/nuclear-power-reactors/generation-iv-nuclear-reactors.aspx` (visited on 07/10/2022).

[13] T. C. Hales, "The Honeycomb Conjecture", *Discrete and Computational Geometry*, vol. 25, no. 1, pp. 1–22, 2001, ISSN: 01795376. DOI: `10.1007/s004540010071`. eprint: `9906042` (math).

[14] "Bioinspired engineering of honeycomb structure - Using nature to inspire human innovation", *Progress in Materials Science*, vol. 74, pp. 332–400, 2015, ISSN: 00796425.

[15] C. Bay, "Étude des Performances de Solveurs Déterministes sur un Coeur Rapide à Caloporteur Sodium", M. Sc. A. report, Polytechnique Montréal, 2013.

[16] T. Takeda and H. Ikeda, "3-D neutron transport benchmarks", *Journal of Nuclear Science and Technology*, vol. 28, no. 7, pp. 656–669, 1991, ISSN: 00223131. DOI: `10.1080/18811248.1991.9731408`.

[17] CEA, "Les réacteurs à neutrons rapides de 4ème génération à caloporteur sodium - Le démonstrateur technologique ASTRID", *Direction de l'énergie nucléaire*, Dec. 2012.

[18] G. Marleau, A. Hébert, and R. Robert, "New Computational Methods Used in the Lattice Code Dragon", in *Proc. Int. Topl. Mtg. on Advances in Reactor Physics*, Charleston, USA: American Nuclear Society, 1992.

[19] J. M. Ruggieri *et al.*, "ERANOS 2.1: International code system for GEN IV fast reactor analysis", *Proceedings of the 2006 International Congress on Advances in Nuclear Power Plants, ICAPP'06*, no. June 2015, pp. 2–10, 2006.

[20] J.-Y. Moller and J.-J. Lautard, "MINARET, A Deterministic Neutron Transport Solver for Nuclear Core Calculations", in *International Conference on Mathematics and Computational Methods Applied to Nuclear Science and Engineering (M&C)*, Rio de Janeiro, RJ, Brazil: American Nuclear Society, May 2011, ISBN: 978-85-63688-00-2.

[21] A.-M. Baudron and J.-J. Lautard, "MINOS: a simplified pn solver for core calculation", *Nuclear science and engineering*, vol. 155, no. 2, pp. 250–263, 2007, ISSN: 00295639. DOI: `10.13182/NSE07-A2660`.

[22] A. Hébert, *Applied Reactor Physics*, 3rd ed., J. Yelon, Ed. Montréal: Presses Internationales Polytechnique, 2020, ISBN: 9782553017353.

[23] E. E. Lewis and W. F. Miller, *Computational Methods of Neutron Transport.* John Wiley & Sons, 1984, ISBN: 0-471-09245-2.

[24] J. J. Duderstadt and W. R. Martin, *Transport Theory.* New York, Chichester, Brisbane, London: John Wiley & Sons, 1979.

[25] C. Mark, "The Spherical Harmonics Methods I (General Development of the Theory)", Chalk River National Laboratory, Chalk River, Ontario, Tech. Rep. 491, Feb. 1957.

[26] C. Mark, "The Spherical Harmonics Methods, II (Application to Problems with Plane & Spherical Symmetry)", Chalk River National Laboratory, Chalk River, Ontario, Tech. Rep. 490 (Reprint), 1958.

[27] E. M. Gelbard, "Application of Spherical Harmonics Method to Reactor Problems", Westinghouse Electric Corp. Bettis Atomic Power Lab., Pittsburgh, Pennsylvania, USA, Tech. Rep., 1960.

[28] G. C. Pomraning, "Asymptotic and variational derivations of the simplified PN equations", *Annals of Nuclear Energy*, vol. 20, no. 9, pp. 623–637, 1993, ISSN: 03064549. DOI: 10.1016/0306-4549(93)90030-S.

[29] S. Chandrasekhar, *Radiative Transfer.* Dover Publications Inc., 1960, p. 393, ISBN: 0-486-60590-6.

[30] B. G. Carlson, "Solution of the Transport Equation by Sn Approximations", Los Alamos National Laboratory, Tech. Rep. 1599, Oct. 1953.

[31] B. G. Carlson and C. E. Lee, "Mechanical Quadrature and the Transport Equation", Los Alamos National Laboratory, Tech. Rep. 2573, Aug. 1961.

[32] S. Loubiere *et al.*, "APOLLO2, Twelve Years Later", in *International Conference on Mathematical and Computation (M&C)*, American Nuclear Society, 1999.

[33] G. Marleau, A. Hébert, and R. Roy, "A User Guide for DRAGON Version5", Polytechnique Montreal, Tech. Rep., 2022.

[34] S. Schunert, "Development of a Quantitative Decision Metric for Selecting the Most Suitable Discretization Method for SN Transport Problems", Ph.D. dissertation, North Carolina State University, 2013, p. 93.

[35] A. Hébert, "High order diamond differencing schemes", *Annals of Nuclear Energy*, vol. 33, no. 17-18, pp. 1479–1488, 2006, ISSN: 03064549. DOI: 10.1016/j.anucene.2006.10.003.

[36] N. Martin and A. Hébert, "A three-dimensional SN high-order diamond differencing discretization with a consistent acceleration scheme", *Annals of Nuclear Energy*, vol. 36, no. 11-12, pp. 1787–1796, 2009, ISSN: 03064549. DOI: `10.1016/j.anucene.2009.08.014`.

[37] R. S. Jeffers, "Spatial Goal-Based Error Estimation and Adaptive Mesh Refinement (AMR) for Diamond Difference Discrete Ordinate (DD-SN) Methods", Ph.D. dissertation, Imperial College London, 2017.

[38] N. K. Madsen, "Convergence of singular difference approximations for the discrete ordinate equations in x - y geometry", *Mathematics of Computation*, vol. 26, no. 117, pp. 45–50, 1972.

[39] E. W. Larsen, "Spatial Convergence Properties of the Diamond Difference Method in x,y Geometry", *Nuclear Science and Engineering*, vol. 80, no. 4, pp. 710–713, 1982, ISSN: 0029-5639. DOI: `10.13182/NSE82-a18980`.

[40] J. I. Duo and Y. Y. Azmy, "Error Comparison of Diamond Difference, Nodal, and Characteristic Methods for Solving Multidimensional Transport Problems with the Discrete Ordinates Approximation", *Nuclear Science and Engineering*, vol. 156, no. 2, pp. 139–153, 2007, ISSN: 00295639. DOI: `10.13182/NSE05-91`.

[41] F. Brezzi, L. D. Marini, and E. Süli, "Discontinuous Galerkin Methods for First-Order Hyperbolic Problems", *Mathematical Models and Methods in Applied Sciences*, vol. 14, no. 12, pp. 1893–1903, 2004, ISSN: 0218-2025. DOI: `10.1142/S0218202504003866`.

[42] Y. Nakasone, S. Yoshimoto, and T. A. Stolarski, "Basics of Finite-Element Method", in *Engineering Analysis with ANSYS Software*, 1st ed., Elsevier Butterworth-Heinemann, 2006, ch. 1, pp. 1–36.

[43] W. H. Reed and T. R. Hill, "Triangular mesh methods for the neutron transport equation", Los Alamos Scientific Laboratory, Tech. Rep. LA-UR-73-479, 1973.

[44] Y. Wang and J. C. Ragusa, "A high-order discontinuous Galerkin method for the SN transport equations on 2D unstructured triangular meshes", *Annals of Nuclear Energy*, vol. 36, no. 7, pp. 931–939, 2009, ISSN: 03064549. DOI: `10.1016/j.anucene.2009.03.002`.

[45] Y. Wang and J. C. Ragusa, "On the Convergence of DGFEM Applied to the Discrete Ordinates Transport Equation for Structured and Unstructured Triangular Meshes", *Nuclear Science and Engineering*, vol. 163, no. 1, pp. 56–72, 2009, ISSN: 00295639.

[46] S. Schunert, Y. Y. Azmy, D. Fournier, and R. Le Tellier, "Comparison of the Accuracy of Various Spatial Discretization Schemes of the Discrete Ordinates Equations in 2D Cartesian Geometry", in *International Conference on Mathematics and Computational Methods Applied to Nuclear Science & Engineering Science (M&C)*, ANS, Ed., Rio de Janeiro, RJ, Brazil, May 2011, ISBN: 9788563688002.

[47] R. Le Tellier, "Interfaces en physique des réacteurs nucléaires - Contribution à la modélisation et au développement de méthodes numeriques associées en neutronique et physique du corium", Thèse d'habilitation à diriger des recherches, Université Grenoble Alpes, 2019. DOI: `10.13140/RG.2.2.27006.84801`.

[48] J. P. Hennart and E. del Valle, "A generalized nodal finite element formalism for discrete ordinates equations in slab geometry part I: Theory in the continuous moment case", *Transport Theory and Statistical Physics*, vol. 24, no. 4-5, pp. 449–478, 1995, ISSN: 15322424. DOI: `10.1080/00411459508206013`.

[49] O. C. Zienkiewicz, R. L. Taylor, and J. Z. Zhu, *The Finite Element Method: Its Basis and Fundamentals*, 6th ed. Elsevier, 2005. DOI: `10.1016/b978-1-85617-633-0.00020-4`.

[50] A. A. Calloo and A. Hébert, "Development and Evaluation of a Discontinuous Galerkin Method in the DRAGON5 Code", in *38th Annual Conference of the Canadian Nuclear Society and 42nd Annual CNS/CNA Student Conference*, Saskatoon, SK, Canada: Canadian Nuclear Society, Jun. 2018.

[51] A. Hébert, *Dragon Version5 Download and Information Page*, 2022. [Online]. Available: `http://merlin.polymtl.ca/version5.htm` (visited on 07/01/2022).

[52] A. Hébert, "High-Order Linear Discontinuous and Diamond Differencing Schemes Along Cyclic Characteristics", *Nuclear Science and Engineering*, vol. 184, 2016. DOI: `http://dx.doi.org/10.13182/NSE16-82`.

[53] A. Canbakan and A. Hébert, "Accuracy of a 2-level scheme based on a subgroup method for pressurized water reactor fuel assembly models", *Annals of Nuclear Energy*, vol. 81, pp. 164–173, 2015, ISSN: 18732100. DOI: `10.1016/j.anucene.2015.03.034`.

[54] U. Ayachit, *The ParaView Guide: A Parallel Visualization Application*. Clifton Park, NY, USA: Kitware, Inc., 2015, ISBN: 1930934300.

[55] M. Yamasaki, T. Takeda, Y. Tahara, and M. Nakano, "Development of Transport Code for Hexagonal Geometry", *Journal of Nuclear Science and Technology*, vol. 29, no. 12, pp. 1143–1151, 1992, ISSN: 0022-3131. DOI: `10.1080/18811248.1992.9731650`.

[56] D. Labeurthre, A. Calloo, and R. Le Tellier, "Extending Gout's Wachspress Finite Elements on Regular Hexagons to Higher Orders", in *International Conference on Mathematical and Computational Methods Applied to Nuclear Science and Engineering (M&C)*, Raleigh, North Carolina: American Nuclear Society, 2021.

[57] Y. Wang, "Adaptive mesh refinement solution techniques for the multigroup SN transport equation using a high-order discontinuous finite element methods", Ph.D. dissertation, Texas A&M University, 2009.

[58] D. Schneider, "Éléments finis mixtes duaux pour la résolution numérique de l'équation de la diffusion neutronique en géométrie hexagonale", Ph.D. dissertation, University of Paris VI, 2000.

[59] H. A. Schwarz, "Ueber einige Abbildungsaufgaben.", 1869.

[60] E. B. Christoffel, "Sul problema delle temperature stazionarie e la rappresentazione di una data superficie", *Annali di Matematica Pura ed Applicata (1867-1897)*, vol. 1, no. 1, pp. 89–103, 1867.

[61] R. Le Tellier, C. Suteau, D. Fournier, and J. M. Ruggieri, "High-order discrete ordinate transport in hexagonal geometry: A new capability in ERANOS", *Il Nuovo Cimento C*, vol. 33, no. 1, pp. 121–128, 2010. DOI: 10.1393/ncc/i2010-10565-5.

[62] E. del Valle and E. H. Mund, "RTk/SN Solutions of the 2D Multigroup Transport Equations in Hexagonal Geometry", in *Physics of Reactors (PHYSOR)*, Seoul, Oct. 2002, pp. 1–10.

[63] J. Hennart, E. Mund, and E. Del Valle, "A composite nodal finite element for hexagons", *Nuclear Science and Engineering*, vol. 127, no. 2, 1997, ISSN: 00295639. DOI: 10.13182/NSE97-A28593.

[64] A. Ern and J.-L. Guermond, *Finite Elements I: Approximation and Interpolation.* Springer, 2021, ISBN: 9783030563417. DOI: 10.1007/978-3-030-56341-7.

[65] A. Hébert, "A Raviart-Thomas-Schneider implementation of the simplified Pn method in 3-D hexagonal geometry", in *International Conference on the Physics of Reactors (PHYSOR)*, American Nuclear Society, 2010, pp. 163–181, ISBN: 9781617820014.

[66] M. Bando, T. Yamamoto, Y. Saito, and T. Takeda, "Three-Dimensional Transport Calculation Method for Eigenvalue Problems Using Diffusion Synthetic Acceleration", *Journal of Nuclear Science and Technology*, vol. 22, no. 10, pp. 841–850, 1985, ISSN: 00223131. DOI: 10.1080/18811248.1985.9735733.

[67] M. L. Adams and E. W. Larsen, "Fast Iterative Methods for Discrete-Ordinates Particle Transport Calculations", *Progress in Nuclear Energy*, vol. 40, no. I, pp. 3–159, 2002.

[68] W. Ford, "The Advancement of Stable, Efficient and Parallel Acceleration Methods for the Neutron Transport Equation", Ph.D. dissertation, Université Paris-Saclay, 2019.

[69] F. Févotte, "Piecewise Diffusion Synthetic Acceleration scheme for neutron transport simulations in optically thick systems", *Annals of Nuclear Energy*, vol. 118, pp. 71–80, 2018, ISSN: 18732100. DOI: 10.1016/j.anucene.2018.03.044.

[70] S. Schunert *et al.*, "A flexible nonlinear diffusion acceleration method for the SN transport equations discretized with discontinuous finite elements", *Journal of Computational Physics*, vol. 338, pp. 107–136, 2017, ISSN: 10902716. DOI: 10.1016/j.jcp.2017.01.070.

[71] J. S. Warsa, T. A. Wareing, and J. E. Morel, "Fully Consistent Diffusion Synthetic Acceleration of Linear Discontinuous SN Transport Discretizations on Unstructured Tetrahedral Meshes", *Nuclear Science and Engineering*, vol. 141, pp. 236–251, 2002, ISSN: 00295639.

[72] W. H. Reed, "The Effectiveness of Acceleration Techniques for Iterative Methods in Transport Theory", *Nuclear Science and Engineering*, vol. 45, no. 3, pp. 245–254, 1971, ISSN: 0029-5639. DOI: 10.13182/NSE71-a19077.

[73] R. E. Alcouffe, "Diffusion Synthetic Acceleration Methods for the Diamond-Differenced Discrete-Ordinates Equations.", *Nuclear Science and Engineering*, vol. 64, no. 2, pp. 344–355, 1977, ISSN: 00295639. DOI: 10.13182/NSE77-1.

[74] H. Khalil, "Effectiveness of a Consistently Formulated Diffusion-Synthetic Acceleration Differencing Approach", *Nuclear Science and Engineering*, vol. 98, no. 3, pp. 226–243, 1988, ISSN: 00295639. DOI: 10.13182/NSE88-A22324.

[75] M. L. Adams and W. R. Martin, "Diffusion Synthetic Acceleration of Discontinuous Finite Element Transport Iterations", *Nuclear Science and Engineering*, vol. 111, pp. 145–167, 1992, ISSN: 00295639.

[76] A. Hébert, "Mixed-Dual Implementations of the Simplified Pn Method", *Annals of Nuclear Energy*, vol. 37, no. 4, pp. 498–511, 2010, ISSN: 03064549. DOI: 10.1016/j.anucene.2010.01.006.

[77] A. Hébert, "The Search for Superconvergence in Spherical Harmonics Approximations", *Nuclear Science and Engineering*, vol. 154, no. 2, pp. 134–173, 2006, ISSN: 00295639. DOI: 10.13182/NSE06-A2623.

[78] M. L. Adams, "New nonlinear methods for linear transport calculations", in *Joint international conference on mathematical methods and supercomputing in nuclear applications (M&C & SNA)*, Karlsruhe, Apr. 1993, pp. 683–694. [Online]. Available: `http://inis.iaea.org/Search/search.aspx?orig_q=RN:25062078`.

[79] M. J. Flynn, "Some computer organizations and their effectiveness", *IEEE Transactions on Computers*, vol. C-21, no. 9, pp. 948–960, 1972, ISSN: 00189340. DOI: `10.1109/TC.1972.5009071`.

[80] M. Pharr and W. R. Mark, "ispc: A SPMD compiler for high-performance CPU programming", in *Innovative Parallel Computing (InPar)*, San Jose: IEEE, May 2012, ISBN: 9781467326322. DOI: `10.1109/InPar.2012.6339601`.

[81] K. R. Koch, R. S. Baker, and R. E. Alcouffe, "A Parallel Algorithm for 3D SN Transport Sweeps", Los Alamos National Laboratory, Tech. Rep., 1992.

[82] K. R. Koch, R. S. Baker, and R. E. Alcouffe, "Solution of the first-order form of the 3-D discrete ordinates equation on a massively parallel processor", *Transactions of the American Nuclear Society*, vol. 65, no. 108, pp. 198–199, 1992.

[83] R. S. Baker and K. R. Koch, "An Sn algorithm for the massively parallel CM-200 computer", *Nuclear Science and Engineering*, vol. 128, no. 3, pp. 312–320, 1998, ISSN: 00295639. DOI: `10.13182/NSE98-1`.

[84] Digital Research Alliance of Canada, *Scalability*, 2019. [Online]. Available: `https://docs.alliancecan.ca/wiki/Scalability/en` (visited on 07/08/2022).

[85] S. D. Pautz, "An Algorithm for Parallel SN Sweeps on Unstructured Meshes", Los Alamos National Laboratory, Tech. Rep., 2001.

[86] G. G. Davidson, T. M. Evans, J. J. Jarrell, S. P. Hamilton, T. M. Pandya, and R. N. Slaybaugh, "Massively parallel, three-dimensional transport solutions for the k-eigenvalue problem", *Nuclear Science and Engineering*, vol. 177, no. 2, pp. 111–125, 2014, ISSN: 00295639. DOI: `10.13182/NSE12-101`.

[87] M. P. Adams *et al.*, "Provably Optimal Parallel Transport Sweeps on Regular Grids", in *International Conference on Mathematics and Computational Methods Applied to Nuclear Science & Engineering (M&C)*, Sun Valley, Idaho, USA: American Nuclear Society, 2013.

[88] M. P. Adams *et al.*, "Provably optimal parallel transport sweeps on semi-structured grids", *Journal of Computational Physics*, vol. 407, p. 109 234, 2020, ISSN: 10902716. DOI: `10.1016/j.jcp.2020.109234`.

[89] G. Tanase *et al.*, "The STAPL Parallel Container Framework", in *ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, 2011, ISBN: 9781450301190.

[90] S. Moustafa, I. Dutka-Malen, L. Plagne, A. Ponçot, and P. Ramet, "Shared Memory Parallelism for 3D Cartesian Discrete Ordinates Solver", in *Joint International Conference on Supercomputing in Nuclear Applications and Monte Carlo (SNA + MC)*, Paris, France, Oct. 2013.

[91] S. Moustafa, "Massively Parallel Cartesian Discrete Ordinates Method for Neutron Transport Simulation", Ph.D. dissertation, Université de Bordeaux, 2015.

[92] S. Moustafa, M. Faverge, L. Plagne, and P. Ramet, "3D Cartesian Transport Sweep for Massively Parallel Architectures with PaRSEC", *Proceedings - 2015 IEEE 29th International Parallel and Distributed Processing Symposium, IPDPS 2015*, pp. 581–590, 2015. DOI: `10.1109/IPDPS.2015.75`.

[93] S. Moustafa, I. Dutka-Malen, L. Plagne, A. Ponçot, and P. Ramet, "Shared memory parallelism for 3D Cartesian discrete ordinates solver", *Annals of Nuclear Energy*, vol. 82, pp. 179–187, 2015, ISSN: 18732100. DOI: `10.1016/j.anucene.2014.08.034`.

[94] James Reingers, *Intel Threading Building Blocks*, 1st ed. Sebastopol: O'Reilly & Associates, Inc., 2007. DOI: `10.1535/itj.1201.03`.

[95] C. Augonnet, S. Thibault, R. Namyst, and P.-A. Wacrenier, "StarPU: a unified platform for task scheduling on heterogeneous multicore architectures", *Concurrency and Computation: Practice and Experience*, vol. 23, pp. 187–198, 2011, ISSN: 15320634. DOI: `10.1002/cpe`.

[96] G. Bosilca, A. Bouteiller, A. Danalis, T. Herault, P. Lemarinier, and J. Dongarra, "DAGuE: A generic distributed DAG engine for High Performance Computing", *Parallel Computing*, vol. 38, no. 1-2, pp. 37–51, 2012, ISSN: 01678191. DOI: `10.1016/j.parco.2011.10.003`.

APPENDIX A   MATLAB SCRIPTS FOR GENERATING DGFEM EQUATIONS

(Note: all scripts make about 2000 lines especially the parts for formatting into DRAGON5-friendly Fortran code. For this reason, we only reproduce part here. The rest will be hosted on the DRAGON5 Archives webpage [51].)

```matlab
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% SET THE OPTIONS IN THE FIRST SECTION (HERE) %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clc ;
clear all ; close all ; format loose %compact
format short %long

dim   = 2;
order = 1;
len_ret = 50 ;

% Specify if using Lagrange (lag = true) or Legendre (lag = false)
% polynomials.
lag   = false ;

% If using Lagrange polynomials, convert to Legendre basis (true) or not
% (false) .
conv  = false ;
% if lag; conv = true ;   end

hodd = false ;

syms MU ETA XI SIG

%% Set cases depending on dimension and build Change-Of-Base matrices if
%  needed

if dim == 1 ;
    cases = {'G'; 'L'} ;
elseif dim == 2 ;
    cases = {'GG'; 'LG'; 'GL'; 'LL'};
elseif dim == 3 ;
    cases = {'GGG'; 'LGG'; 'GLG'; 'LLG'; 'GGL'; 'LGL'; 'GLL'; 'LLL'};
```

```matlab
else
    error('main: cannot process this dimension. Check input.')
end

if lag
    % Get Change-Of-Basis matrices for DG and DD to Standard
    S_DG = getCOBmatrix(dim,2,order) ;
    S_DD = getCOBmatrix(dim,1,order) ;

    % Get Change-Of-Basis matrices from DG to DD, and vice-versa
    DD_DG = sym(inv(S_DD)*S_DG) ;
    DG_DD = sym(inv(S_DG)*S_DD) ;
end

%% HODD

if hodd
    if dim==1
        [mat_terms, res_q, jump_q] = get1DmatDD(order,cases) ;
    elseif dim==2
        [mat_terms, res_q, jump_q] = get2DmatDD(dim,order) ;
    elseif dim==3
        [mat_terms, res_q, jump_q] = get3DmatDD(dim,order) ;
    end
end


%% Get DG local matrix, source vector, and jump terms vector (for each
    case
%   e.g. GG, LG, GL, LL)

if ~hodd
if dim==1
    [mat_terms, res_q, jump_q] = get1Dmatrices(lag,order,cases) ;
elseif dim==2
    [mat_terms, res_q, jump_q] = get2Dmatrices(lag,dim,order) ;
elseif dim==3
    [mat_terms, res_q, jump_q] = get3Dmatrices(lag,dim,order) ;
end
end


%% USEFUL FOR "DG LAGRANGE EXPRESSED IN LEGENDRE" ONLY (? - TBC).
%   Transform DG matrices and vectors to DD matrices and vectors
```

```matlab
78
79  if dim == 1;
80     for i_cases=1:length(cases)
81        if lag && conv
82           new_mat(:,:,i_cases) = simplify(DD_DG*(mat_terms(:,:,i_cases)*...
83              DG_DD)) ;
84           new_q(:,1,i_cases) = simplify(DD_DG*res_q(:,1,i_cases)) ;
85           new_j(:,1,i_cases) = collect(simplify...
86              (DD_DG*jump_q(:,1,i_cases)),MU) ;
87        else
88           new_mat(:,:,i_cases) = mat_terms(:,:,i_cases) ;
89           new_q(:,1,i_cases) = res_q(:,1,i_cases) ;
90           new_j(:,1,i_cases) = jump_q(:,1,i_cases) ;
91        end
92     end
93  else
94     if lag && conv
95        new_mat = simplify(DD_DG*(mat_terms*DG_DD)) ;
96        new_q = simplify(DD_DG*res_q) ;
97        for i_cases=1:length(cases)
98           new_j(:,i_cases) = collect(simplify(DD_DG*jump_q(:,i_cases)),...
99              [MU ETA XI]) ;
100       end
101    else
102       new_mat = mat_terms;
103       new_q = res_q;
104       new_j = jump_q;
105    end
106 end
```

```matlab
function [mat_terms, result_q, jump_q] = ...
    get2Dmatrices(lag,dim,order)

%%%%%%%% IMAGE SHOWING POSITIONS OF VARIABLES, REF. DISCONTINUITY
    %%%%%%%
%                         ooh                 oog
%                     +----------------------+
%                ooa | iic                 iid | oof
%                    |                         |
%                    |                         |
%                    |                         |
%                    |                         |
%                    |                         |
%                    |                         |
%                    |                         |
%                    |                         |
%                    |                         |
%                    |                         |
%                oob | iia                 iib | ooe
%                    +----------------------+
%                         ooc                 ood
%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


syms MU ETA SIG QQQ
syms ABSMU ABSETA
if lag
    syms ooa oob ooc ood ooe oof oog ooh ooi ooj ook ool oom oon ooo oop
    syms iia iib iic iid iie iif iig iih iii iij iik iil iim iin iio iip
    varDG = [iia iib iic iid iie iif iig iih iii iij iik iil iim iin iio
    iip] ;
else
    syms     PH1 PH2 PH3 PH4 PH5 PH6 PH7 PH8 PH9 P10 P11 P12 P13 P14 P15
    P16
    varDG = [PH1 PH2 PH3 PH4 PH5 PH6 PH7 PH8 PH9 P10 P11 P12 P13 P14 P15
    P16] ;
end
alpha = [1 2 3 4]; beta = [1 2 3 4];
cases = {'GG'; 'LG'; 'GL'; 'LL'};


%%
```

```matlab
39 count_m = 1;
40 % mat_vjump = sym(zeros((order+1)^2,(order+1)^2,length(cases)));
41 for j=1:(order+1)
42    for i=1:(order+1)
43        X = [alpha(i),beta(j)];
44        disp(X);
45
46        for k=1:length(cases)
47            curCase=char(cases(k));
48            switch curCase
49            case 'GG';
50            jump = getJumpVal(order, alpha(i), beta(j), 1, curCase);
51            case 'LG';
52            jump = getJumpVal(order, alpha(i), beta(j), 1, curCase);
53            case 'GL';
54            jump = getJumpVal(order, alpha(i), beta(j), 1, curCase);
55            case 'LL';
56            jump = getJumpVal(order, alpha(i), beta(j), 1, curCase);
57            end
58 %                disp(jump)
59
60            if lag
61                [term_J_MU, term_J_ETA, term_S_MU, term_S_ETA, term_SIGMA]...
62                    = get2DTerms(dim,order,alpha(i),beta(j)) ;
63            else
64                [term_J_MU, term_J_ETA, term_S_MU, term_S_ETA, term_SIGMA]...
65                    = get2DLeg(dim,order,alpha(i),beta(j),curCase) ;
66                jump(1:2) = 1 ;
67            end
68
69            result(i,j,k) = collect(...
70                MU   * term_J_MU *jump(1) + ...
71                ETA  * term_J_ETA*jump(2) + ...
72                MU   * term_S_MU + ...
73                ETA * term_S_ETA + ...
74                SIG*term_SIGMA, varDG) ;
75 %                ooa oob ooc ood ooe oof oog ooh]) ;
76 %                disp(result(i,j,k))
77            result_q(k,(i-1)+(j-1)*(order+1)+1) = collect(QQQ*term_SIGMA, ...
78                varDG) ;
79
80        end
81
82        %%%%%%%%%%%%%%%%%%%%%% GREP COEFFS.
```

```matlab
      %%%%%%%%%%%%%%%%%%%%%%%%%%
83        for i_cases = 1:4
84            for i_var=1:(order+1)^2
85                [cfs,trms] = coeffs(result(i,j,i_cases), varDG(i_var)) ;
86                if length(trms) == 2;
87                    % to make sure not to assign if required variable
88                    % varDG(i_var) is not present
89                    var(i_var,i_cases) = cfs(1) ;
90                else
91                    var(i_var,i_cases) = 0 ;
92                end
93            end
94            [cfs,trms] = coeffs(result(i,j,i_cases), varDG) ;
95            %
      ----------------------------------------------------------------%
96            temp = cfs(trms == 1) ;
97            if isempty(temp) == 1; temp = 0; end
98            jump_q(i_cases,(i-1)+(j-1)*(order+1)+1) = 0 - temp ;
99        end
100
101       %%%%%%%%%%%%%%%%%%%%%%%% COMPARE COEFFS.
      %%%%%%%%%%%%%%%%%%%%%%%%%%%
102       for i_cases = 2:4
103           for i_var=1:(order+1)^2
104               lg_eq = isequal(var(i_var,1),var(i_var,i_cases));
105 %                 var(i_var,1)
106 %                 var(i_var,i_cases)
107               if(lg_eq == 0) && (i_cases ~=4)
108                   checkCoeff(var(i_var,1),var(i_var,i_cases),dim) ;
109               elseif(lg_eq == 0) && (i_cases ==4)
110                   [var(i_var,1)] = findABS(var(i_var,1),var(i_var,i_cases),
      dim) ;
111               end
112           end
113       end
114
115       mat_terms(count_m,:) = var(:,1) ;
116       count_m = count_m+1;
117
118    end
119 end
120
121 result_q = transpose(result_q(1,:)) ;
122 % just want to take one case since should be same for all cases.
```

```
123  jump_q    = transpose(        jump_q) ;
124
125  end
```

```matlab
function [term_J_MU, term_J_ETA, term_S_MU, term_S_ETA, term_SIGMA] ...
    = get2DLeg(dim,order,alpha,beta,curCase,varargin)

    %
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    syms U V
    syms L1(U)
    syms PHI
    syms P0xm P0ym % P0xp P0yp
    syms PH1
    syms MU ETA SIG
    if order == 0
        L1(U) = 1 ;
        E1 = collect(L1(U)*L1(V), [U V]) ;
        %
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        PHI(U,V) = collect(PH1*E1, [U V]) ; % flux Inside domain
        % PHO(U,V) = collect(PHA*E1, [U V]) ; % flux Outside domain
        %
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        PHImmv = P0xm ;
        PHIppv = P0xm ;
        PHImmu = P0ym ;
        PHIppu = P0ym ;
    elseif order == 1
        syms L2(U)
        syms PH2 PH3 PH4
        syms P1xm P1ym % P1xp P1yp
        L1(U) = 1 ;
        L2(U) = 2*sqrt(3)*U ;
        E1 = collect(L1(U)*L1(V), [U V]) ;
        E2 = collect(L2(U)*L1(V), [U V]) ;
        E3 = collect(L1(U)*L2(V), [U V]) ;
        E4 = collect(L2(U)*L2(V), [U V]) ;
        %
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        PHI(U,V) = collect(PH1*E1 + PH2*E2 + PH3*E3 + PH4*E4, [U V]) ;
        %expand, collect, combine
        %
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        PHImmv = P0xm + P1xm*L2(V) ;
        PHIppv = P0xm + P1xm*L2(V) ;
        PHImmu = P0ym + P1ym*L2(U) ;
        PHIppu = P0ym + P1ym*L2(U) ;
```

```matlab
40    elseif order == 2
41        syms L2(U) L3(U)
42        syms PH2 PH3 PH4 PH5 PH6 PH7 PH8 PH9
43        syms P1xm P1ym P2xm P2ym
44        L1(U) = 1 ;
45        L2(U) = 2*sqrt(3)*U ;
46        L3(U) = sqrt(5)*(6*U*U - 1/2) ;
47        E1 = collect(L1(U)*L1(V), [U V]) ;
48        E2 = collect(L2(U)*L1(V), [U V]) ;
49        E3 = collect(L3(U)*L1(V), [U V]) ;
50        E4 = collect(L1(U)*L2(V), [U V]) ;
51        E5 = collect(L2(U)*L2(V), [U V]) ;
52        E6 = collect(L3(U)*L2(V), [U V]) ;
53        E7 = collect(L1(U)*L3(V), [U V]) ;
54        E8 = collect(L2(U)*L3(V), [U V]) ;
55        E9 = collect(L3(U)*L3(V), [U V]) ;
56        %
      %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
57        PHI(U,V) = collect(PH1*E1 + PH2*E2 + PH3*E3 + PH4*E4 + PH5*E5 ...
58            + PH6*E6 + PH7*E7 + PH8*E8 + PH9*E9 , [U V]) ;
59        %
      %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
60        PHImmv = P0xm + P1xm*L2(V) + P2xm*L3(V) ;
61        PHIppv = P0xm + P1xm*L2(V) + P2xm*L3(V) ;
62        PHImmu = P0ym + P1ym*L2(U) + P2ym*L3(U) ;
63        PHIppu = P0ym + P1ym*L2(U) + P2ym*L3(U) ;
64    elseif order == 3
65        syms L2(U) L3(U) L4(U)
66        syms PH2 PH3 PH4 PH5 PH6 PH7 PH8 PH9 P10 P11 P12 P13 P14 P15 P16
67        syms P1xm P1ym P2xm P2ym P3xm P3ym
68        L1(U) = 1 ;
69        L2(U) = 2*sqrt(3)*U ;
70        L3(U) = sqrt(5)*(6*U*U - 1/2) ;
71        L4(U) = sqrt(7)*(20*U^3 - 3*U) ;
72        E1  = collect(L1(U)*L1(V), [U V]) ;
73        E2  = collect(L2(U)*L1(V), [U V]) ;
74        E3  = collect(L3(U)*L1(V), [U V]) ;
75        E4  = collect(L4(U)*L1(V), [U V]) ;
76        E5  = collect(L1(U)*L2(V), [U V]) ;
77        E6  = collect(L2(U)*L2(V), [U V]) ;
78        E7  = collect(L3(U)*L2(V), [U V]) ;
79        E8  = collect(L4(U)*L2(V), [U V]) ;
80        E9  = collect(L1(U)*L3(V), [U V]) ;
81        E10 = collect(L2(U)*L3(V), [U V]) ;
```

```matlab
        E11 = collect(L3(U)*L3(V), [U V]) ;
        E12 = collect(L4(U)*L3(V), [U V]) ;
        E13 = collect(L1(U)*L4(V), [U V]) ;
        E14 = collect(L2(U)*L4(V), [U V]) ;
        E15 = collect(L3(U)*L4(V), [U V]) ;
        E16 = collect(L4(U)*L4(V), [U V]) ;
        %
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        PHI(U,V) = collect(PH1*E1 + PH2*E2 + PH3*E3 + PH4*E4 + PH5*E5 ...
            + PH6*E6 + PH7*E7 + PH8*E8 + PH9*E9 + P10*E10 + P11*E11 ...
            + P12*E12 + P13*E13 + P14*E14 + P15*E15 + P16*E16, [U V]) ;
        %
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        PHImmv = P0xm + P1xm*L2(V) + P2xm*L3(V) + P3xm*L4(V) ;
        PHIppv = P0xm + P1xm*L2(V) + P2xm*L3(V) + P3xm*L4(V) ;
        PHImmu = P0ym + P1ym*L2(U) + P2ym*L3(U) + P3ym*L4(U) ;
        PHIppu = P0ym + P1ym*L2(U) + P2ym*L3(U) + P3ym*L4(U) ;
    else
        error('get2DLeg: ORDER not available! Choose another order.')
    end
    %
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % In PHImmv for example, the 'v' indicates that this is a function in
    % terms of v, i.e. PHImmv = f(v) .
    PHImpv = collect(PHI(-1/2,V) , [U V]) ;
    PHIpmv = collect(PHI( 1/2,V) , [U V]) ;
    PHImpu = collect(PHI(U,-1/2) , [U V]) ;
    PHIpmu = collect(PHI(U, 1/2) , [U V]) ;
    %
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    DIFF_PHI_U = diff(PHI,U) ;
    DIFF_PHI_V = diff(PHI,V) ;
    PHI_AV = int((int(PHI,U,-1/2,1/2)),V,-1/2,1/2)/...
        int((int(1,U,-1/2,1/2)),V,-1/2,1/2) ;
    %
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


    if alpha==1
        if curCase(1)=='G'
            jump_MU  = (PHImpv - PHImmv)*L1(-1/2) ;
        elseif curCase(1)=='L'
            jump_MU  = (PHIppv - PHIpmv)*L1( 1/2) ;
        end
```

```matlab
        if beta==1
            if curCase(2)=='G'
                jump_ETA = (PHImpu - PHImmu)*L1(-1/2) ;
            elseif curCase(2)=='L'
                jump_ETA = (PHIppu - PHIpmu)*L1( 1/2) ;
            end
            term_J_MU  = int(L1(V)*(jump_MU ),V,-1/2,1/2) ;
            term_J_ETA = int(L1(U)*(jump_ETA),U,-1/2,1/2) ;
            term_S_MU  = int(L1(V)*(int(L1(U)*DIFF_PHI_U,U,-1/2,1/2)),V,...
                -1/2,1/2) ;
            term_S_ETA = int(L1(U)*(int(L1(V)*DIFF_PHI_V,V,-1/2,1/2)),U,...
                -1/2,1/2) ;
            term_SIGMA = int((int(L1(U)*L1(V)*PHI,U,-1/2,1/2)),V,...
                -1/2,1/2) ;
        elseif beta==2
            if curCase(2)=='G'
                jump_ETA = (PHImpu - PHImmu)*L2(-1/2) ;
            elseif curCase(2)=='L'
                jump_ETA = (PHIppu - PHIpmu)*L2( 1/2) ;
            end
            term_J_MU  = int(L2(V)*(jump_MU ),V,-1/2,1/2) ;
            term_J_ETA = int(L1(U)*(jump_ETA),U,-1/2,1/2) ;
            term_S_MU  = int(L2(V)*(int(L1(U)*DIFF_PHI_U,U,-1/2,1/2)),V,...
                -1/2,1/2) ;
            term_S_ETA = int(L1(U)*(int(L2(V)*DIFF_PHI_V,V,-1/2,1/2)),U,...
                -1/2,1/2) ;
            term_SIGMA = int((int(L1(U)*L2(V)*PHI,U,-1/2,1/2)),V,...
                -1/2,1/2) ;
        elseif beta==3
            if curCase(2)=='G'
                jump_ETA = (PHImpu - PHImmu)*L3(-1/2) ;
            elseif curCase(2)=='L'
                jump_ETA = (PHIppu - PHIpmu)*L3( 1/2) ;
            end
            term_J_MU  = int(L3(V)*(jump_MU ),V,-1/2,1/2) ;
            term_J_ETA = int(L1(U)*(jump_ETA),U,-1/2,1/2) ;
            term_S_MU  = int(L3(V)*(int(L1(U)*DIFF_PHI_U,U,-1/2,1/2)),V,...
                -1/2,1/2) ;
            term_S_ETA = int(L1(U)*(int(L3(V)*DIFF_PHI_V,V,-1/2,1/2)),U,...
                -1/2,1/2) ;
            term_SIGMA = int((int(L1(U)*L3(V)*PHI,U,-1/2,1/2)),V,...
                -1/2,1/2) ;
        elseif beta==4
            if curCase(2)=='G'
```

```
165         jump_ETA = (PHImpu - PHImmu)*L4(-1/2) ;
166      elseif curCase(2)=='L'
167         jump_ETA = (PHIppu - PHIpmu)*L4( 1/2) ;
168      end
169      term_J_MU  = int(L4(V)*(jump_MU ),V,-1/2,1/2) ;
170      term_J_ETA = int(L1(U)*(jump_ETA),U,-1/2,1/2) ;
171      term_S_MU  = int(L4(V)*(int(L1(U)*DIFF_PHI_U,U,-1/2,1/2)),V,...
172         -1/2,1/2) ;
173      term_S_ETA = int(L1(U)*(int(L4(V)*DIFF_PHI_V,V,-1/2,1/2)),U,...
174         -1/2,1/2) ;
175      term_SIGMA = int((int(L1(U)*L4(V)*PHI,U,-1/2,1/2)),V,...
176         -1/2,1/2) ;
177    end
178  elseif alpha==2
179    if curCase(1)=='G'
180       jump_MU  = (PHImpv - PHImmv)*L2(-1/2) ;
181    elseif curCase(1)=='L'
182       jump_MU  = (PHIppv - PHIpmv)*L2( 1/2) ;
183    end
184    if beta==1
185       if curCase(2)=='G'
186          jump_ETA = (PHImpu - PHImmu)*L1(-1/2) ;
187       elseif curCase(2)=='L'
188          jump_ETA = (PHIppu - PHIpmu)*L1( 1/2) ;
189       end
190       term_J_MU  = int(L1(V)*(jump_MU ),V,-1/2,1/2) ;
191       term_J_ETA = int(L2(U)*(jump_ETA),U,-1/2,1/2) ;
192       term_S_MU  = int(L1(V)*(int(L2(U)*DIFF_PHI_U,U,-1/2,1/2)),V,...
193          -1/2,1/2) ;
194       term_S_ETA = int(L2(U)*(int(L1(V)*DIFF_PHI_V,V,-1/2,1/2)),U,...
195          -1/2,1/2) ;
196       term_SIGMA = int((int(L2(U)*L1(V)*PHI,U,-1/2,1/2)),V,...
197          -1/2,1/2) ;
198    elseif beta==2
199       if curCase(2)=='G'
200          jump_ETA = (PHImpu - PHImmu)*L2(-1/2) ;
201       elseif curCase(2)=='L'
202          jump_ETA = (PHIppu - PHIpmu)*L2( 1/2) ;
203       end
204       term_J_MU  = int(L2(V)*(jump_MU ),V,-1/2,1/2) ;
205       term_J_ETA = int(L2(U)*(jump_ETA),U,-1/2,1/2) ;
206       term_S_MU  = int(L2(V)*(int(L2(U)*DIFF_PHI_U,U,-1/2,1/2)),V,...
207          -1/2,1/2) ;
208       term_S_ETA = int(L2(U)*(int(L2(V)*DIFF_PHI_V,V,-1/2,1/2)),U,...
```

```matlab
209                 -1/2,1/2) ;
210            term_SIGMA = int((int(L2(U)*L2(V)*PHI,U,-1/2,1/2)),V,...
211                -1/2,1/2) ;
212        elseif beta==3
213            if curCase(2)=='G'
214                jump_ETA = (PHImpu - PHImmu)*L3(-1/2) ;
215            elseif curCase(2)=='L'
216                jump_ETA = (PHIppu - PHIpmu)*L3( 1/2) ;
217            end
218            term_J_MU  = int(L3(V)*(jump_MU ),V,-1/2,1/2) ;
219            term_J_ETA = int(L2(U)*(jump_ETA),U,-1/2,1/2) ;
220            term_S_MU  = int(L3(V)*(int(L2(U)*DIFF_PHI_U,U,-1/2,1/2)),V,...
221                -1/2,1/2) ;
222            term_S_ETA = int(L2(U)*(int(L3(V)*DIFF_PHI_V,V,-1/2,1/2)),U,...
223                -1/2,1/2) ;
224            term_SIGMA = int((int(L2(U)*L3(V)*PHI,U,-1/2,1/2)),V,...
225                -1/2,1/2) ;
226        elseif beta==4
227            if curCase(2)=='G'
228                jump_ETA = (PHImpu - PHImmu)*L4(-1/2) ;
229            elseif curCase(2)=='L'
230                jump_ETA = (PHIppu - PHIpmu)*L4( 1/2) ;
231            end
232            term_J_MU  = int(L4(V)*(jump_MU ),V,-1/2,1/2) ;
233            term_J_ETA = int(L2(U)*(jump_ETA),U,-1/2,1/2) ;
234            term_S_MU  = int(L4(V)*(int(L2(U)*DIFF_PHI_U,U,-1/2,1/2)),V,...
235                -1/2,1/2) ;
236            term_S_ETA = int(L2(U)*(int(L4(V)*DIFF_PHI_V,V,-1/2,1/2)),U,...
237                -1/2,1/2) ;
238            term_SIGMA = int((int(L2(U)*L4(V)*PHI,U,-1/2,1/2)),V,...
239                -1/2,1/2) ;
240        end
241    elseif alpha==3
242        if curCase(1)=='G'
243            jump_MU  = (PHImpv - PHImmv)*L3(-1/2) ;
244        elseif curCase(1)=='L'
245            jump_MU  = (PHIppv - PHIpmv)*L3( 1/2) ;
246        end
247        if beta==1
248            if curCase(2)=='G'
249                jump_ETA = (PHImpu - PHImmu)*L1(-1/2) ;
250            elseif curCase(2)=='L'
251                jump_ETA = (PHIppu - PHIpmu)*L1( 1/2) ;
252            end
```

```matlab
253          term_J_MU  = int(L1(V)*(jump_MU ),V,-1/2,1/2) ;
254          term_J_ETA = int(L3(U)*(jump_ETA),U,-1/2,1/2) ;
255          term_S_MU  = int(L1(V)*(int(L3(U)*DIFF_PHI_U,U,-1/2,1/2)),V,...
256              -1/2,1/2) ;
257          term_S_ETA = int(L3(U)*(int(L1(V)*DIFF_PHI_V,V,-1/2,1/2)),U,...
258              -1/2,1/2) ;
259          term_SIGMA = int((int(L3(U)*L1(V)*PHI,U,-1/2,1/2)),V,...
260              -1/2,1/2) ;
261      elseif beta==2
262          if curCase(2)=='G'
263              jump_ETA = (PHImpu - PHImmu)*L2(-1/2) ;
264          elseif curCase(2)=='L'
265              jump_ETA = (PHIppu - PHIpmu)*L2( 1/2) ;
266          end
267          term_J_MU  = int(L2(V)*(jump_MU ),V,-1/2,1/2) ;
268          term_J_ETA = int(L3(U)*(jump_ETA),U,-1/2,1/2) ;
269          term_S_MU  = int(L2(V)*(int(L3(U)*DIFF_PHI_U,U,-1/2,1/2)),V,...
270              -1/2,1/2) ;
271          term_S_ETA = int(L3(U)*(int(L2(V)*DIFF_PHI_V,V,-1/2,1/2)),U,...
272              -1/2,1/2) ;
273          term_SIGMA = int((int(L3(U)*L2(V)*PHI,U,-1/2,1/2)),V,...
274              -1/2,1/2) ;
275      elseif beta==3
276          if curCase(2)=='G'
277              jump_ETA = (PHImpu - PHImmu)*L3(-1/2) ;
278          elseif curCase(2)=='L'
279              jump_ETA = (PHIppu - PHIpmu)*L3( 1/2) ;
280          end
281          term_J_MU  = int(L3(V)*(jump_MU ),V,-1/2,1/2) ;
282          term_J_ETA = int(L3(U)*(jump_ETA),U,-1/2,1/2) ;
283          term_S_MU  = int(L3(V)*(int(L3(U)*DIFF_PHI_U,U,-1/2,1/2)),V,...
284              -1/2,1/2) ;
285          term_S_ETA = int(L3(U)*(int(L3(V)*DIFF_PHI_V,V,-1/2,1/2)),U,...
286              -1/2,1/2) ;
287          term_SIGMA = int((int(L3(U)*L3(V)*PHI,U,-1/2,1/2)),V,...
288              -1/2,1/2) ;
289      elseif beta==4
290          if curCase(2)=='G'
291              jump_ETA = (PHImpu - PHImmu)*L4(-1/2) ;
292          elseif curCase(2)=='L'
293              jump_ETA = (PHIppu - PHIpmu)*L4( 1/2) ;
294          end
295          term_J_MU  = int(L4(V)*(jump_MU ),V,-1/2,1/2) ;
296          term_J_ETA = int(L3(U)*(jump_ETA),U,-1/2,1/2) ;
```

```matlab
297         term_S_MU  = int(L4(V)*(int(L3(U)*DIFF_PHI_U,U,-1/2,1/2)),V,...
298             -1/2,1/2) ;
299         term_S_ETA = int(L3(U)*(int(L4(V)*DIFF_PHI_V,V,-1/2,1/2)),U,...
300             -1/2,1/2) ;
301         term_SIGMA = int((int(L3(U)*L4(V)*PHI,U,-1/2,1/2)),V,...
302             -1/2,1/2) ;
303     end
304 elseif alpha==4
305     if curCase(1)=='G'
306         jump_MU  = (PHImpv - PHImmv)*L4(-1/2) ;
307     elseif curCase(1)=='L'
308         jump_MU  = (PHIppv - PHIpmv)*L4( 1/2) ;
309     end
310     if beta==1
311         if curCase(2)=='G'
312             jump_ETA = (PHImpu - PHImmu)*L1(-1/2) ;
313         elseif curCase(2)=='L'
314             jump_ETA = (PHIppu - PHIpmu)*L1( 1/2) ;
315         end
316         term_J_MU  = int(L1(V)*(jump_MU ),V,-1/2,1/2) ;
317         term_J_ETA = int(L4(U)*(jump_ETA),U,-1/2,1/2) ;
318         term_S_MU  = int(L1(V)*(int(L4(U)*DIFF_PHI_U,U,-1/2,1/2)),V,...
319             -1/2,1/2) ;
320         term_S_ETA = int(L4(U)*(int(L1(V)*DIFF_PHI_V,V,-1/2,1/2)),U,...
321             -1/2,1/2) ;
322         term_SIGMA = int((int(L4(U)*L1(V)*PHI,U,-1/2,1/2)),V,...
323             -1/2,1/2) ;
324     elseif beta==2
325         if curCase(2)=='G'
326             jump_ETA = (PHImpu - PHImmu)*L2(-1/2) ;
327         elseif curCase(2)=='L'
328             jump_ETA = (PHIppu - PHIpmu)*L2( 1/2) ;
329         end
330         term_J_MU  = int(L2(V)*(jump_MU ),V,-1/2,1/2) ;
331         term_J_ETA = int(L4(U)*(jump_ETA),U,-1/2,1/2) ;
332         term_S_MU  = int(L2(V)*(int(L4(U)*DIFF_PHI_U,U,-1/2,1/2)),V,...
333             -1/2,1/2) ;
334         term_S_ETA = int(L4(U)*(int(L2(V)*DIFF_PHI_V,V,-1/2,1/2)),U,...
335             -1/2,1/2) ;
336         term_SIGMA = int((int(L4(U)*L2(V)*PHI,U,-1/2,1/2)),V,...
337             -1/2,1/2) ;
338     elseif beta==3
339         if curCase(2)=='G'
340             jump_ETA = (PHImpu - PHImmu)*L3(-1/2) ;
```

```
341        elseif curCase(2)=='L'
342            jump_ETA = (PHIppu - PHIpmu)*L3( 1/2) ;
343        end
344        term_J_MU  = int(L3(V)*(jump_MU ),V,-1/2,1/2) ;
345        term_J_ETA = int(L4(U)*(jump_ETA),U,-1/2,1/2) ;
346        term_S_MU  = int(L3(V)*(int(L4(U)*DIFF_PHI_U,U,-1/2,1/2)),V,...
347            -1/2,1/2) ;
348        term_S_ETA = int(L4(U)*(int(L3(V)*DIFF_PHI_V,V,-1/2,1/2)),U,...
349            -1/2,1/2) ;
350        term_SIGMA = int((int(L4(U)*L3(V)*PHI,U,-1/2,1/2)),V,...
351            -1/2,1/2) ;
352    elseif beta==4
353        if curCase(2)=='G'
354            jump_ETA = (PHImpu - PHImmu)*L4(-1/2) ;
355        elseif curCase(2)=='L'
356            jump_ETA = (PHIppu - PHIpmu)*L4( 1/2) ;
357        end
358        term_J_MU  = int(L4(V)*(jump_MU ),V,-1/2,1/2) ;
359        term_J_ETA = int(L4(U)*(jump_ETA),U,-1/2,1/2) ;
360        term_S_MU  = int(L4(V)*(int(L4(U)*DIFF_PHI_U,U,-1/2,1/2)),V,...
361            -1/2,1/2) ;
362        term_S_ETA = int(L4(U)*(int(L4(V)*DIFF_PHI_V,V,-1/2,1/2)),U,...
363            -1/2,1/2) ;
364        term_SIGMA = int((int(L4(U)*L4(V)*PHI,U,-1/2,1/2)),V,...
365            -1/2,1/2) ;
366    end
367 end
```

```matlab
function jump = getJumpVal(order, alpha, beta, gamma, curCase)

    if (order==1) || (order==2) || (order==3) || (order==0)

        jump(1, 1) = false; % for mu direction
        jump(2, 1) = false; % for eta direction
        jump(3, 1) = false; % for xi direction

        if alpha==1 && curCase(1)=='G'
            jump(1) = true;
        elseif alpha==(order+1) && curCase(1)=='L'
            jump(1) = true;
        end

        if length(curCase) > 1
            if beta==1 && curCase(2)=='G'
                jump(2) = true;
            elseif beta==(order+1) && curCase(2)=='L'
                jump(2) = true;
            end
        end

        if length(curCase) > 2
            if gamma==1 && curCase(3)=='G'
                jump(3) = true;
            elseif gamma==(order+1) && curCase(3)=='L'
                jump(3) = true;
            end
        end

    else
        error('getJumpVal: not valid value of order.')
    end

end
```

APPENDIX B HODD AS DPGFEM

(This appendix is adapted from Schunert's proof [34]. While most of their presentation is three-dimensional, we stick with 2D for consistency with the rest of this document as well as for the greater simplicity of the proof without loss of generality.)

We start with the Discontinuous Finite Element Method (DFEM) formulation derived in subsection 3.1.4. Note that while the test and trial solution spaces in that presentation are from the same space, it changes nothing to the derivation to assume they are not. The formulation is reproduced below for easy accessibility,

$$\int_{\xi}\left(\left(\mathbf{\Omega}_n \cdot \nabla \psi_{n,h}\right)w_h + \Sigma\psi_{n,h}w_h\right)ds - \int_{\partial\xi_-}\left(\mathbf{n}\cdot\mathbf{\Omega}\right)[\![\psi_{n,h}]\!]w_h^+ dl = \int_{\xi}Q_n w_h ds \ . \tag{B.1}$$

where $[\![\psi_{n,h}]\!] = \left(\psi_{n,h}^+ - \psi_{n,h}^-\right)$ was used as the *jump operator.*

Now, consider the two-dimensional problem whereby the flux within a cell element can instead be approximated as

$$\begin{aligned}
\psi_{n,i,j}(u,v) = \sum_{\alpha=0}^{\Lambda}\sum_{\beta=0}^{\Lambda}\widetilde{P}_\alpha(u)\widetilde{P}_\beta(v)\ G_{n,i,j}^{[\alpha,\beta]} + \sum_{\beta=0}^{\Lambda}\frac{1}{\sqrt{2(\Lambda+1)+1}}\widetilde{P}_{\Lambda+1}(u)\widetilde{P}_\beta(v)\ G_{n,i,j}^{[\Lambda+1,\beta]} \\
+ \sum_{\alpha=0}^{\Lambda}\frac{1}{\sqrt{2(\Lambda+1)+1}}\widetilde{P}_\alpha(u)\widetilde{P}_{\Lambda+1}(v)\ G_{n,i,j}^{[\alpha,\Lambda+1]}\ ,
\end{aligned} \tag{B.2}$$

where G are just generic expansion coefficients, and the test (weight) space is given by

$$\mathcal{W} = \left\{\widetilde{P}_\alpha(u)\widetilde{P}_\beta(v), 0 \le \alpha, \beta \le \Lambda\right\} \ . \tag{B.3}$$

We consider throughout this demonstration only the direction of neutron travel defined by $\mu > 0, \eta > 0$. This is sufficient as the rest follow by analogy. This means that the unknowns are the cell Legendre moments and the face Legendre moments on the right and top sides of the square reference element.

These can be obtained with their respective usual definitions, *i.e.*,

$$\psi_{n,i,j}^{[\alpha,\beta]} = \int_{-1/2}^{1/2} \int_{-1/2}^{1/2} du \; dv \widetilde{P}_\alpha(u) \; \widetilde{P}_\beta(v) \; \psi_{n,i,j}(u,v) \; ,$$

$$\psi_{n,i+,j}^{[*,\beta]} = \int_{-1/2}^{1/2} dv \; \widetilde{P}_\beta(v) \; \psi_{n,i,j}(+1/2,v) \; , \tag{B.4}$$

$$\psi_{n,i,j+}^{[\alpha,*]} = \int_{-1/2}^{1/2} du \; \widetilde{P}_\alpha(u) \; \psi_{n,i,j}(u,+1/2) \; .$$

It can be shown that Eqn. B.2 can be recast in the following form

$$\begin{aligned}
\psi_{n,i,j}(u,v) = &\sum_{\alpha=0}^{\Lambda} \sum_{\beta=0}^{\Lambda} \widetilde{P}_\alpha(u) \widetilde{P}_\beta(v) \; \psi_{n,i,j}^{[\alpha,\beta]} \\
&+ \sum_{\beta=0}^{\Lambda} \frac{1}{\sqrt{2(\Lambda+1)+1}} \widetilde{P}_{\Lambda+1}(u) \widetilde{P}_\beta(v) \left[\psi_{n,i+,j}^{[*,\beta]} - \sum_{\alpha=0}^{\Lambda} \sqrt{2\alpha+1} \; \psi_{n,i,j}^{[\alpha,\beta]} \right] \\
&+ \sum_{\alpha=0}^{\Lambda} \frac{1}{\sqrt{2(\Lambda+1)+1}} \widetilde{P}_\alpha(u) \widetilde{P}_{\Lambda+1}(v) \left[\psi_{n,i,j+}^{[*,\beta]} - \sum_{\beta=0}^{\Lambda} \sqrt{2\beta+1} \; \psi_{n,i,j}^{[\alpha,\beta]} \right] \; .
\end{aligned} \tag{B.5}$$

It might be easier to see this if we consider a specific example. For the 2D case with $\Lambda = 1$ for the incoming left edge: from the definition of $\psi_{n,i+,j}^{[*,\beta]}$ and $\psi_{n,i,j}^{[\alpha,\beta]}$ (Eqn. B.4), we obtain

$$\psi_{n,i+,j}^{[*,\beta]} = G_{n,i,j}^{[0,\beta]} + \sqrt{3} G_{n,i,j}^{[1,\beta]} + \sqrt{5} G_{n,i,j}^{[2,\beta]} \; ,$$

$$\psi_{n,i,j}^{[\alpha,\beta]} = G_{n,i,j}^{[\alpha,\beta]} \; ,$$

leaving the first square bracket term as $G_{n,i,j}^{[\Lambda+1,\beta]}$.

Looking at Eqn. B.5 above, we can see that we have more unknowns than equations, specifically, the $(\Lambda+1)^{\text{th}}$-order terms. Schunert [34] then proposes to impose that the approximated angular flux be continuous on the incoming surface *in an integral sense*. This, as we will see, ensures that a solution can be found.

The flux on the interior trace of the incoming face can be obtained by simply substituting the appropriate coordinates in Eqn. B.5. Considering again only the left incoming edge, this

is given by

$$
\begin{aligned}
\psi_{n,i,j}(-1/2, v) = &\sum_{\alpha=0}^{\Lambda}\sum_{\beta=0}^{\Lambda}\sqrt{2\alpha+1}\,(-1)^{\alpha}\,\widetilde{P}_{\beta}(v)\,\psi_{n,i,j}^{[\alpha,\beta]} \\
&+\sum_{\beta=0}^{\Lambda}(-1)^{\Lambda+1}\,\widetilde{P}_{\beta}(v)\,\left[\psi_{n,i+,j}^{[*,\beta]}-\sum_{\alpha=0}^{\Lambda}\sqrt{2\alpha+1}\,\psi_{n,i,j}^{[\alpha,\beta]}\right] \\
&+\sum_{\alpha=0}^{\Lambda}\frac{\sqrt{2\alpha+1}}{\sqrt{2(\Lambda+1)+1}}\,(-1)^{\alpha}\,\widetilde{P}_{\Lambda+1}(v)\,\left[\psi_{n,i,j+}^{[*,\beta]}-\sum_{\beta=0}^{\Lambda}\sqrt{2\beta+1}\,\psi_{n,i,j}^{[\alpha,\beta]}\right]\,,
\end{aligned}
\tag{B.6}
$$

while, on the exterior trace, the flux is expanded in terms of Legendre polynomials ($\widetilde{P}_{\beta}(v)$) and surface Legendre moments ($\psi_{n,i-,j}^{[*,\beta]}$), up to order Λ,

$$
\psi_{n,i,j}^{-}(-1/2, v) = \sum_{\beta=0}^{\Lambda}\widetilde{P}_{\beta}(v)\,\psi_{n,i-,j}^{[*,\beta]}\,.
\tag{B.7}
$$

Now, requiring the angular flux to be continuous at the interface *in an integral sense* essentially means that the difference of the interior and exterior traces should be zero when weighed against the test space on that restricted domain of the surface, *i.e.*,

$$
\int_{-1/2}^{1/2}dv\widetilde{P}_{\beta}(v)[\![\psi_{n,i,j}(-1/2, v)]\!] = \int_{-1/2}^{1/2}dv\widetilde{P}_{\beta}(v)\left(\psi_{n,i,j}(-1/2, v)-\psi_{n,i,j}^{-}(-1/2, v)\right) = 0\,,
\tag{B.8}
$$

with $\beta = [0, \Lambda]$. This condition is the same as not requiring pointwise continuity across the whole of the boundary.

Looking at the difference between the traces, we find

$$
\begin{aligned}
\psi_{n,i,j}(-1/2, v)-\psi_{n,i,j}^{-}(-1/2, v) = &\sum_{\beta=0}^{\Lambda}\widetilde{P}_{\beta}(v)\left[(-1)^{\Lambda+1}\,\psi_{n,i+,j}^{[*,\beta]}-\psi_{n,i-,j}^{[*,\beta]}\right] \\
&+\sum_{\alpha=0}^{\Lambda}\sum_{\beta=0}^{\Lambda}\sqrt{2\alpha+1}\,\widetilde{P}_{\beta}(v)\,\psi_{n,i,j}^{[\alpha,\beta]}\left[(-1)^{\alpha}-(-1)^{\Lambda+1}\right] \\
&+ \text{other terms}\,,
\end{aligned}
\tag{B.9}
$$

where only terms that would be non-zero when weighed against the test space have been retained. Indeed, terms in $\widetilde{P}_{\Lambda+1}(v)$ would be orthogonal to $\widetilde{P}_{\beta}(v)$, $\beta = [0, \Lambda]$ within Eqn. B.8 and are implicitly gathered in "other terms". Therefore, substituting the retained terms back

in Eqn. B.8 gives

$$\left[(-1)^{\Lambda+1}\,\psi_{n,i+,j}^{[*,\beta]} - \psi_{n,i-,j}^{[*,\beta]}\right] + \sum_{\alpha=0}^{\Lambda} \sqrt{2\alpha+1}\,\psi_{n,i,j}^{[\alpha,\beta]}\left[(-1)^{\alpha} - (-1)^{\Lambda+1}\right] = 0 \;,\beta = [0,\Lambda] \;. \quad \text{(B.10)}$$

Multiplying by $(-1)^{\Lambda+1}$ throughout and rearranging yields

$$\left[\psi_{n,i+,j}^{[*,\beta]} - (-1)^{\Lambda+1}\,\psi_{n,i-,j}^{[*,\beta]}\right] = \sum_{\alpha=0}^{\Lambda} \sqrt{2\alpha+1}\,\psi_{n,i,j}^{[\alpha,\beta]}\left[1 + (-1)^{\alpha+\Lambda}\right] \;,\beta = [0,\Lambda] \;, \quad \text{(B.11)}$$

which are actually the exact same auxiliary equations given by Hébert [35] for the x-axis. Considering the two incoming edges, this leads to $2(\Lambda+1)$ auxiliary equations or constraints.

Having imposed the condition Eqn. B.8, the strong formulation of the DFEM equation (Eqn. B.1) reduces to

$$\int_{\xi}\left((\boldsymbol{\Omega}_n \cdot \nabla\psi_{n,h})\,w_h + \Sigma\psi_{n,h}w_h\right)ds = \int_{\xi}Q_n w_h ds \;. \quad \text{(B.12)}$$

Substituting in the flux representation (given by Eqn. B.5) leads to $(\Lambda+1)^2$ equations with $(\Lambda+1)^2 + 2(\Lambda+1)$ unknowns. Making use of the aforementioned constraints, this reduces to $(\Lambda+1)^2$ unknowns, hence ensuring that the problem is well-defined. The resulting set of equations are then identical to those derived by Hébert [35].

APPENDIX C   ANALYTICAL EQUATIONS FOR THE DSA

We begin with Eqn 5.1, with the iteration indices made explicit,

$$\boldsymbol{\Omega} \cdot \nabla \psi^{(\kappa+1/2)}(\boldsymbol{r}, \boldsymbol{\Omega}) + \Sigma_t(\boldsymbol{r})\psi^{(\kappa+1/2)}(\boldsymbol{r}, \boldsymbol{\Omega}) = \frac{\Sigma_s(\boldsymbol{r})}{4\pi}\phi^{(\kappa)}(\boldsymbol{r}) + Q_{\text{ext}}(\boldsymbol{r}, \boldsymbol{\Omega}) , \qquad \text{(C.1)}$$

and ϕ is defined as before,

$$\phi(\boldsymbol{r}) = \phi_0(\boldsymbol{r}) = \int_{4\pi} d^2\Omega \ \psi(\boldsymbol{r}, \boldsymbol{\Omega}) , \qquad \text{(C.2)}$$

where the 0 subscript indicates that it is the zeroth moment of the angular flux – when considering an expansion of the angular flux in real spherical harmonics written in terms of polynomials of direction cosines [22].

Using Larsen's [39] four-step approach, one takes the zeroth and first moment of Eqn. C.1 to get,

$$\nabla \cdot \Phi_1^{(\kappa+1/2)}(\boldsymbol{r}) + \Sigma_t(\boldsymbol{r})\phi_0^{(\kappa+1/2)}(\boldsymbol{r}) = \Sigma_s(\boldsymbol{r})\phi_0^{(\kappa)}(\boldsymbol{r}) + q_0(\boldsymbol{r}) , \qquad \text{(C.3a)}$$

$$\frac{1}{3}\nabla\phi_0^{(\kappa+1/2)}(\boldsymbol{r}) + \frac{2}{3}\nabla \cdot \Phi_2^{(\kappa+1/2)}(\boldsymbol{r}) + \Sigma_t(\boldsymbol{r})\Phi_1^{(\kappa+1/2)}(\boldsymbol{r}) = \boldsymbol{q}_1(\boldsymbol{r}) , \qquad \text{(C.3b)}$$

where

$$\begin{aligned} \Phi_1(\boldsymbol{r}) &= \int_{4\pi} d^2\Omega \ \boldsymbol{\Omega}\psi(\boldsymbol{r}, \boldsymbol{\Omega}) , \\ \Phi_2(\boldsymbol{r}) &= \int_{4\pi} d^2\Omega \ \frac{1}{2}(3\boldsymbol{\Omega}\boldsymbol{\Omega} - \boldsymbol{I})\psi(\boldsymbol{r}, \boldsymbol{\Omega}) , \\ q_0(\boldsymbol{r}) &= \int_{4\pi} d^2\Omega \ Q_{\text{ext}}(\boldsymbol{r}, \boldsymbol{\Omega}) , \\ \boldsymbol{q}_1(\boldsymbol{r}) &= \int_{4\pi} d^2\Omega \ \boldsymbol{\Omega}Q_{\text{ext}}(\boldsymbol{r}, \boldsymbol{\Omega}) . \end{aligned} \qquad \text{(C.4)}$$

This is the first step, and produces a scalar equation and a vector equation. Step 2 is the definition of the acceleration equations, obtained by the re-indexing of some of the iteration indices,

$$\nabla \cdot \Phi_1^{(\kappa+1)}(\boldsymbol{r}) + \Sigma_t(\boldsymbol{r})\phi_0^{(\kappa+1)}(\boldsymbol{r}) = \Sigma_s(\boldsymbol{r})\phi_0^{(\kappa+1)}(\boldsymbol{r}) + q_0(\boldsymbol{r}) , \qquad \text{(C.5a)}$$

$$\frac{1}{3}\nabla\phi_0^{(\kappa+1)}(\boldsymbol{r}) + \frac{2}{3}\nabla \cdot \Phi_2^{(\kappa+1/2)}(\boldsymbol{r}) + \Sigma_t(\boldsymbol{r})\Phi_1^{(\kappa+1)}(\boldsymbol{r}) = \boldsymbol{q}_1(\boldsymbol{r}) . \qquad \text{(C.5b)}$$

These equations define the end-of-iteration fluxes, with the former being a statement of

conservation of neutrons such that the balance at the end of iteration will be correct. If a linearly anisotropic flux is assumed, $\Phi_2^{(\kappa+1/2)}(\boldsymbol{r})$ is null. Hence, leaving the $(\kappa + 1/2)$ index implicitly implies that as it will be eliminated in the next step.

Indeed, step 3 is subtracting Eqns. C.3 from Eqns. C.5 to obtain Eqns. C.6 for the corrections:

$$\nabla \cdot \boldsymbol{F}_1^{(\kappa+1)}(\boldsymbol{r}) + (\Sigma_t(\boldsymbol{r}) - \Sigma_s(\boldsymbol{r})) F_0^{(\kappa+1)}(\boldsymbol{r}) = \Sigma_s(\boldsymbol{r}) \left(\phi_0^{(\kappa+1/2)}(\boldsymbol{r}) - \phi_0^{(\kappa)}(\boldsymbol{r}) \right) , \qquad \text{(C.6a)}$$

$$\frac{1}{3}\nabla F_0^{(\kappa+1)}(\boldsymbol{r}) + \Sigma_t(\boldsymbol{r})\boldsymbol{F}_1^{(\kappa+1)}(\boldsymbol{r}) = 0 , \qquad \text{(C.6b)}$$

where

$$F_0^{(\kappa+1)}(\boldsymbol{r}) = \phi_0^{(\kappa+1)}(\boldsymbol{r}) - \phi_0^{(\kappa+1/2)}(\boldsymbol{r}) ,$$
$$\boldsymbol{F}_1^{(\kappa+1)}(\boldsymbol{r}) = \Phi_1^{(\kappa+1)}(\boldsymbol{r}) - \Phi_1^{(\kappa+1/2)}(\boldsymbol{r}) .$$

The final step is eliminating the current correction, $\boldsymbol{F}_1$ by substituting Eqn. C.6b into Eqn. C.6a, to obtain a diffusion equation for the flux correction,

$$-\nabla \cdot \frac{1}{3\Sigma_t(\boldsymbol{r})}\nabla F_0^{(\kappa+1)}(\boldsymbol{r}) + (\Sigma_t(\boldsymbol{r}) - \Sigma_s(\boldsymbol{r})) F_0^{(\kappa+1)}(\boldsymbol{r}) = \Sigma_s(\boldsymbol{r}) \left(\phi_0^{(\kappa+1/2)}(\boldsymbol{r}) - \phi_0^{(\kappa)}(\boldsymbol{r}) \right) . \quad \text{(C.8)}$$

The usual DSA scheme is then given by Eqns. C.1 and C.8 (both reproduced below for ease) as well as Eqn. C.9:

$$\boldsymbol{\Omega} \cdot \nabla \psi^{(\kappa+1/2)}(\boldsymbol{r}, \boldsymbol{\Omega}) + \Sigma_t(\boldsymbol{r})\psi^{(\kappa+1/2)}(\boldsymbol{r}, \boldsymbol{\Omega}) = \frac{\Sigma_s(\boldsymbol{r})}{4\pi}\phi^{(\kappa)}(\boldsymbol{r}) + Q_{\text{ext}}(\boldsymbol{r}, \boldsymbol{\Omega}) ;$$

$$-\nabla \cdot \frac{1}{3\Sigma_t(\boldsymbol{r})}\nabla F_0^{(\kappa+1)}(\boldsymbol{r}) + (\Sigma_t(\boldsymbol{r}) - \Sigma_s(\boldsymbol{r})) F_0^{(\kappa+1)}(\boldsymbol{r}) = \Sigma_s(\boldsymbol{r}) \left(\phi_0^{(\kappa+1/2)}(\boldsymbol{r}) - \phi_0^{(\kappa)}(\boldsymbol{r}) \right) ;$$

$$\phi_0^{(\kappa+1)}(\boldsymbol{r}) = \phi_0^{(\kappa+1/2)}(\boldsymbol{r}) + F_0^{(\kappa+1)}(\boldsymbol{r}) . \qquad \text{(C.9)}$$