Libowski: A Numerical Framework for Solving Depletion and Mass Transport in Molten Salt Reactors

A Dissertation Presented for the

Doctor of Philosophy

Degree

The University of Tennessee, Knoxville

Robert Zachary Taylor

May 2021

I'll see you on the dark side of the moon

Acknowledgments

I would like to thank Ivan Maldonado and Ben Collins for their mentorship and support. This report was funded by the US Department of Energy's Molten Salt Reactor Campaign headed by Lou Qualls.

Abstract

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

In a world ravaged by air pollution and climate change, producing clean and safe energy becomes ever more vital. Nuclear energy can fill the void of coal and natural gas, bringing both the clean and safe energy that our world requires. In 2018 the world produced over 2500 TWh of nuclear energy. While this is only a small percentage of the worlds total energy production, it has been steadily growing since 1970. A lot of the commercial success, longevity, safety benefits and efficiency for these reactors have been optimized from the development of modeling and simulation tools. The majority of these commercial nuclear reactors operate based on designs that are decades old and are phasing out of commission. The next generation of nuclear reactors brings with it added economic promise, increased efficiency, fuel management and additional safety benefits. To aid in the design, operation and licensing of these advanced reactors, the existing modeling tools must be modified and new simulation tools need to be built.

Liquid fuel molten salt reactors (MSRs) are a class of next generation advanced nuclear reactors with great promise and previous operational history. By design, MSRs operate in a much different way than traditional nuclear reactors. While traditional nuclear reactors work with fixed fuel elements, MSRs dissolved their fuel in a molten salt that continuously flows throughout the reactor loop. This allows for various chemical and isotopic species to transport and react in the loop. During operation of a nuclear reactor, the material composition is constantly changing. These changes come from the neutron flux, nuclear decay and chemical reactions. Understanding how the composition changes is key to many

other physical process which occur in nuclear reactors. Modeling these compositional changes also gives insight into fuel safety and performance.

Modeling the compositional changes in nuclear reactors are referred to as nuclear fuel depletion calculations. These calculations model the atomic density of various isotopes in a nuclear reactor over long and short time steps. Modeling shorter time steps is required in accident scenarios and changes in neutron precursor concentrations for short transients. Long depletion steps are important for understanding fuel burnup, economy and optimizing fuel performance. In traditional nuclear reactors these calculations involve solving a large system of stiff first order ordinary differential equations. Using modern matrix exponential methods, the problem of accurately solving these equations has been solved. The problem is that the existing depletion codes where written to model the current fleet of reactors. These codes do not represent the underlying physical phenomena that are occurring in advanced reactors with flowing fuel.

The goal of this dissertation is to start over from scratch, and to redefine the way in which nuclear depletion calculations are done in advanced reactors. Using this ideology the depletion equations are developed from the fundamental chemical engineering phenomena of mass transport. Starting from the chemical species transport equation, the author has developed a software to model fuel depletion in MSRs. This c++ software is entirely written by the author and is built using the linear algebra library Eigen. The software is named after one of the greatest movies of all time and is called *libowski*.

1.1 Outline and Research Goals

The goal of this work is to redefine what fuel depletion calculations are in a flowing fueled nuclear reactor. Not only to set up the mathematical expressions to model the physical phenomena but to also develop robust and accurate solutions to these equations. This work is laid out in 6 chapters. Chapter 2 introduces the nuclear burnup equations and works to develop the equations required to model depletion in traditional nuclear reactors and MSRs. Chapter 3 discusses the mathematical framework used to solve the MSR depletion equations. Chapter 4 applies these methods and their practical application in libowski.

Chapter 5 shows the existing results so far in this work. Finally chapter 6 introducing the plan of action moving forward with this dissertation.

Chapter 2

Nuclear Fuel Depletion

Over the life time of a nuclear reactor the composition of the fuel changes from being exposed to a neutron rich environment. The changes in fuel composition have an important impact on the reactor power distribution, stability and control. Reactor efficiency as well as fuel utilization and economics are also greatly influenced from changes in fuel composition. As a nuclear reactor operates through time, the composition of the fuel elements changes due to fission, radioactive decay and neutron capture reactions. This change in composition is important because nuclear reaction rates are dependent on the material they occur. Fission rates are most important to calculate in order to maintain core critically. Not only does fissile material deplete producing fission products but fertile material can be converted into new fissile material.

A complete depletion calculation would involve the simultaneous solution of the neutron transport equation along with the isotopic evolution of thousands of fission products and fissile material depletion. This is not possible in practice and approximations to these calculations must be done. The neutron transport and fuel depletion calculations are done separately and their coupling is often done with predictor corrector methods. During neutron transport calculations fuel composition is assumed to be constant. For depletion, reaction rates are assumed to be constant. This process is repeated until a stopping criteria is reached.

In traditional light water reactors, fuel elements are housed in fuel rod bundles, which are not allowed to move during operation. Because molten salt reactors are liquid fueled, they can be refueled and have fission products processed online. In addition to online processing,

fissial material circulates through the reactor loop. These unique design features of molten salt reactors produce new challenges for fuel depletion calculations. The following sections discuss the nuclear reactions involved in the production of radionuclides, neutron transport and the burnup equations.

2.1 Nuclear Reactions

2.1.1 Radioactive Decay

When the nucleus of an atomic element is unstable it can undergo spontaneous transformation into a different element by the emission of energy and or particles [2]. The most relevant modes for decay in burnup calculations are α decay, proton emission, neutron emission, β^- and β^+. A summary of these decay modes are shown in Table 2.1 [1]. The rate at which a nuclide i changes as a function of time is proportional to its decay constant λ_i and the decay of other nuclides which contribute as a source to nuclide i [2],

$$\frac{dn_i}{dt} = \sum_{j=1}^{N} \lambda_{j \to i} n_j(t) - \lambda_i n_i(t). \tag{2.1}$$

The radioactive decay constant λ is a natural constant and is related to the half life of the nuclide. This relation is,

$$\lambda = \frac{\ln(2)}{T_{1/2}}. \tag{2.2}$$

As the half life of the isotope decreases, its decay constant increases.

Some nuclides have more than one mode of decay. Some of these decay modes compete with one another such as β^+ and electron capture. When nuclide i has more than one decay mode, its decay constant is a summation of all k decay modes,

$$\lambda_i = \lambda_{i,1} + \lambda_{i,2} + \lambda_{i,3} ... = \sum_k \lambda_{i,k}. \tag{2.3}$$

Table 2.1: Summary of Decay Modes where Z is atomic number and A is mass number [1]

Mode of decay	Daughter nuclide	By-product nuclide
α decay	(Z - 2, A - 1)	^{4}He
Proton emission	(Z - 1, A - 1)	^{1}H
Neutron emission	(Z, A - 1)	-
β^-	(Z + 1, A)	-
β^+	(Z - 1, A)	-

While total decay is sufficient for use in the loss of nuclide i from Equation 2.1, a branching ratio must be used to denoted the fraction of nuclide j which decays into i. The branching ratio is defined as,

$$b_{j\to i} = \frac{\lambda_{j\to i}}{\lambda_j}. \tag{2.4}$$

For nuclide j which has multiple decay modes k, the branching ratio denotes the fraction of decays for nuclide j which produce nuclide i. Substituting this relation into Equation 2.1,

$$\frac{dn_i}{dt} = \sum_{j=1}^{N} b_{j\to i}\lambda_j n_j(t) - \lambda_i n_i(t). \tag{2.5}$$

2.1.2 Neutron Induced Reactions

Neutron induced reactions follow the schematic shown in Figure 2.1. The projectile neutron encounters a target, resulting in the nuclear reaction. In this reaction, the neutron is denoted by a and the target by b, c and d are the resulting products. These reactions are often written in the following notation [2],

$$b(a,c)d$$

The frequency of a reaction is governed by the probability at which it can occur. This probability is related to the cross section. The microscopic cross section is a factor that characterizes the probability of a neutron-nuclear reaction with an atomic nucleus [2]. Specifically, the formal definition of the microscopic cross section is,

$$\sigma = \frac{\text{Number of reactions/nucleus/sec}}{\text{Number of incident neutrons/cm}^2\text{/sec}} = \frac{(R/N_i)}{\phi}, \tag{2.6}$$

$$R = \left[\frac{\#\text{ of reactions}}{\text{cm}^2 \cdot \text{sec}}\right], \quad \sigma = \left[\text{cm}^2\right], \quad \phi = \left[\frac{\#\text{ of neutrons}}{\text{cm}^2 \cdot \text{sec}}\right], \quad n_i = \left[\frac{\#\text{ of atoms}}{\text{cm}^3}\right].$$

There are a number of neutron induced reactions that occur in a nuclear reactor. They can be divided into two main categories: scattering and absorption. Scattering reactions can then be further divided into elastic and inelastic reactions. Elastic scattering reactions, denoted by (n, n) are those reactions in which an incoming neutron scatters off of a nucleus, changing the neutrons energy and direction. In some cases the neutron can be absorbed into the nucleus for a short time, before being ejected. This may leave the nucleus in an excited state before returning to its initial state. These type of reactions are noted by (n, n') and are known as inelastic scattering.

Absorption reactions are those in which a neutron is absorbed by the atomic nucleus. The resulting products from these reactions include, energy, neutrons, protons and α particles. A summary of these reactions which are relevant in burnup calculations are shown in Table 2.2 [1]. The change in a nuclides concentration as a function of time for a general absorption reaction is,

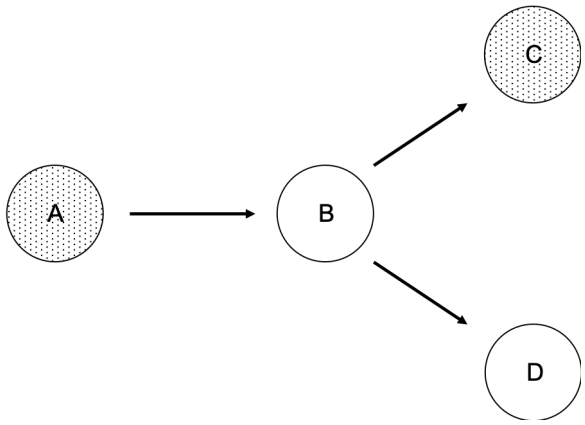

Figure 2.1: Schematic of general nuclear reaction

$$\frac{dn_i}{dt} = \sum_{j=1}^{N}\sum_{k=1}^{K}\gamma_{j\to i,k}\sigma_{k,j}\phi n_j(t) - \phi\sum_{k=1}^{K}\sigma_{k,i}n_i(t), \tag{2.7}$$

where $\gamma_{j\to i,k}$ is the fraction of nuclide i from nuclide j through reaction k. The first term in Equation 2.7 is the generation from the reaction of nuclide. First each nuclide is summed over, followed by a summation of each reaction for nuclide j. If reaction k results in generating nuclide i, then $\sigma_{k,j}$ and $\gamma_{j\to i,k}$ are both nonzero. The second term in Equation 2.7 is the loss of nuclide i from reaction k. For each nuclide i a summation is taken over all possible neutron induced reactions.

The microscopic cross section characterizes the probability that a neutron will have a reaction with a single nucleus. Often it is required to understand how neutrons will interact with macroscopic chunk of the material [2]. The macroscopic cross section provides the ability to compute how a large portion of material will react under a flux of particles or energy. This macroscopic cross section is the computation of the atomic number density multiplied by the microscopic cross section. For a material consisting of multiple atomic elements, the macroscopic cross section of reaction x is computed by,

$$\Sigma_x = \sum_{i=1}^{N} n_i\sigma_{i,x}, \tag{2.8}$$

The macroscopic cross section can be thought of as the probability per length path that a neutron will have a reaction with the nucleus [2].

When fissile material undergoes a neutron absorption (to relate it to the previous paragraphs), neutron induced fission can occur releasing neutrons, energy and fission products. Typically, fission is a two-body reaction, meaning that the target nucleus splits into two isotopes of smaller atomic number. The independent fission product yield is the expected fraction a nuclide is produced for a single fission event. Because the cross section is dependent on the incident neutron energy, fission product yields dependent on the neutron energy. The yields of fission products for ^{235}U for fast and thermal neutrons is shown in Figure 2.2.

Table 2.2: Summary of Decay Modes where Z is atomic number and A is mass number [1]

Mode of reaction	**Daughter nuclide**	**By-product nuclide**
(n,2n)	(Z, A - 1)	-
(n,3n)	(Z, A - 2)	-
(n,4n)	(Z, A - 3)	-
(n,γ)	(Z, A + 1)	-
(n,p)	(Z - 1, A)	^1H
(n,d)	(Z - 1, A - 1)	^{2}H
(n,t)	(Z - 1, A - 2)	^{3}H
(n,^{3}He)	(Z - 2, A - 2)	^{3}He
(n,α)	(Z - 2, A - 3)	^{4}He
Fission	fission products	-

2.2 Neutron Transport Equation

The neutron transport equation describes the angular neutron density as a function of time, space and energy. Let the spatial position be denoted by r and the energy of a neutron E. A neutron will travel in direction Ω and will scatter at angle Ω' and energy E'. The neutron transport equation is,

$$\frac{1}{v}\frac{\partial\varphi}{\partial t} + \Omega\cdot\nabla\varphi + \Sigma_t\varphi = \int_{4\pi} d\Omega' \int_0^\infty dE'\Sigma_s\varphi' + \frac{\chi}{4\pi}\int_0^\infty dE'\nu\Sigma_f\varphi, \tag{2.9}$$

where

$\varphi = \varphi(r, E, \Omega) \equiv$ Angular neutron flux at energy E and angle Ω

$\varphi' = \varphi(r, E', \Omega') \equiv$ Angular neutron flux at energy E' and angle Ω'

$v = v(E) \equiv$ Neutron velocity

$\Sigma_t = \Sigma_t(r, E) \equiv$ Total neutron cross section

$\Sigma_s = \Sigma_s(E' \to E, \Omega' \to \Omega) \equiv$ Neutron scattering cross section from energy E and angle Ω to energy E' and angle Ω'

$\chi = \chi(E) \equiv$ Energy distribution of fission neutrons

$\nu = \nu(E') \equiv$ Average number of neutrons released during fission at energy E'

$\Sigma_f = \Sigma_f(E') \equiv$ Cross section for neutron fission at energy E'

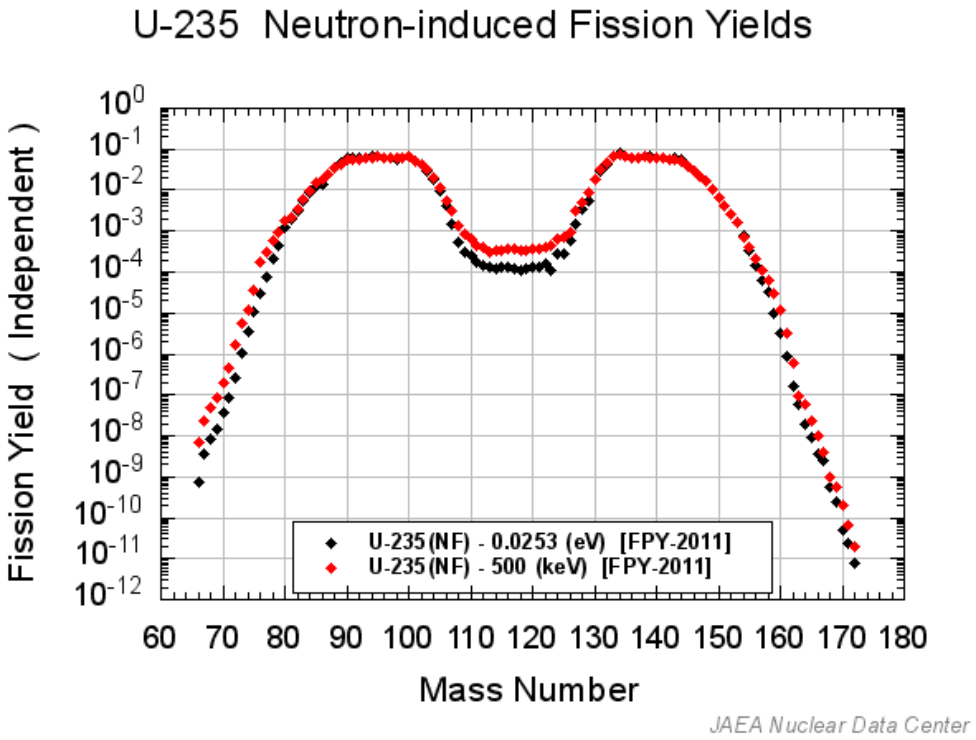

Figure 2.2: Fission product yield for thermal and fast neutrons

Starting from the left hand side of Equation 2.9 we have the change in neutron density with time, neutron streaming and neutron removal from reaction. On the right hand side is the source from a neutron scattering in from and different energy and angle, followed by the source from nuclear fission. The source for the scattering term must be integrated over all angles and energy and the same for the fission source. The angular neutron flux cannot be directly used in the neutron induced reaction equations presented before. These equations require the scalar flux, which is the integral over all angles for the angular neutron flux,

$$\phi(r, E) = \int \varphi(r, E, \Omega) d\Omega. \tag{2.10}$$

Angular dependence for the scattering source can be handled through the use of Legendre expansions or spherical harmonics. Spacial angular dependence is often treated with Gauss-Legendre quadratures such as the Sn or Pn equations, such methods are out of the scope of this work [3]. The energy dependence in Equation 2.9 is almost always handled using the multigroup approximation. This approximation works by dividing up the energy dependence

in groups by applying an energy interval for each group and integrating over that interval. The group cross sections are defined in a way to conserve the reaction rates. The multigroup constants are defined by [3] [2],

$$\Sigma_s^{gg'} = \frac{\int_g^{g-1}\int_{g'}^{g'-1}\Sigma_s(r, E' \to E, t)\varphi(r, E', t)dE'dE}{\int_g^{g-1}\varphi(r, E, t)dE}, \tag{2.11}$$

$$\Sigma_t^g = \frac{\int_g^{g-1}\Sigma_t(r, E, t)\varphi(r, E, t)dE}{\int_g^{g-1}\varphi(r, E, t)dE}, \tag{2.12}$$

$$\nu\Sigma_f^g = \frac{\int_g^{g-1}\nu\Sigma_f(r, E, t)\varphi(r, E, t)dE}{\int_g^{g-1}\varphi(r, E, t)dE}, \tag{2.13}$$

$$\chi^g = \int_g^{g-1}\chi(E)dE, \tag{2.14}$$

$$\varphi^g = \int_g^{g-1}\varphi(r, E, t)dE. \tag{2.15}$$

The time variance in Equation 2.9 is most often neglected for reactor analysis. Instead the problem is transformed into an eigenvalue problem. Although there are multiple different eigenvalue formulations, the λ (k-effective, criticality) eigenvalue is the most common [3]. If Equation 2.9 is written in matrix vector form, then the criticality equation is,

$$\boldsymbol{A\varphi} = \frac{1}{\lambda}\boldsymbol{B\varphi}. \tag{2.16}$$

where λ is the largest of the eigenvalues and can be found using the power iteration. When λ is greater than 1 then the reactor is supercritical and the neutron population will increase over time. If λ is less than 1 the system is subcritical and the neutron population will decrease over time. If λ is equal to 1 then the system is critical and the neutron population will remain constant [2].

2.3 Burnup Equations

Nuclear burnup calculations solve a set of first order ODE's which aim to simulate the isotopic concentrations of radionuclides in a nuclear reactor. These equations are important in reactor calculations because the neutronic properties and neutron distributions in the reactor is dependent on composition of the environments material. After a neutron transport solve, the cross sections and scalar neutron flux are collapsed into a single energy group. Because of the manner in which the groups are collapsed, the reaction rates from the multigroup equations equal the reaction rates for the single group. Over a single burn up calculation step all cross sections and the scalar neutron flux is assumed to be constant. When solving the depletion equations, the reactor is often subdivided into finite volume depletion zones. Each depletion zone with have different material, and differing neutron flux, this results in the reaction rates varying between each region. Using volume average operators and the multigroup approximation, the neutron flux and microscopic cross section is collapsed into single values for each depletion zone [4],

$$\overline{\sigma}_{k,j} = \frac{\int_V \int_0^\infty \sigma_{k,j}(r,E)\phi(r,E)dEdV}{\int_V \int_0^\infty \phi(r,E)dEdV}, \tag{2.17}$$

$$\overline{\phi} = \frac{1}{V}\int_V \int_0^\infty \phi(r,E)dEdV. \tag{2.18}$$

$$\overline{n}(t) = \frac{1}{V}\int_V n(r,t)dV \tag{2.19}$$

This results in the set of ODE's of the form,

$$\frac{d\overline{n}_i}{dt} = \sum_{j=1}^{N}\left(b_{j\to i}\lambda_j + \sum_{k=1}^{K}\gamma_{j\to i,k}\overline{\sigma}_{k,j}\overline{\phi}\right)\overline{n}_j(t) - \left(\lambda_i + \overline{\phi}\sum_{k=1}^{K}\overline{\sigma}_{k,i}\right)\overline{n}_i(t). \tag{2.20}$$

There are multiple methods for solving the isotopic concentrations in nuclear reactors [5] [6] [7]. Normal numerical integration techniques such as Runge Kutta face complications because of the various orders of magnitude for the decay constants of short and long lived nuclides. This results in a stiff system of ODE's, which are difficulty and costly to

solve. Recently, matrix exponential methods have become popular for depletion calculations because of their high level of accuracy [1] [4].

2.3.1 Extension to Molten Salt Reactors

Traditional nuclear reactors contain their fuel in a solid form that is unable to move during operation. These fuel elements are housed in pellets which are loaded in fuel rods. These rods are collected into fuel bundles which are then loaded into the reactor core for operation. Loading the reactor core with fuel in this form does not allow the material to transport through the reactor loop. Molten salt reactors work with a liquid form of the fuel that is dissolved in a molten salt. This liquid form of the fuel allows it to transport through the reactor loop during operation. Thus the depletion equations used in traditional reactors no longer hold true for molten salt reactors. Equation 2.20 must be modified to include the change in nuclide concentration with fluid movement: convection and diffusion.

Depletion calculations in MSRs can be better represented by the multicomponent chemical species transport equation. The species transport equation is derived by a conservation of mass basis which accounts for change in time, convective and diffusion transport and rate of generation from reaction [8]. Equation 2.21 is the species transport equation for species i.

$$\underbrace{\frac{\partial \rho_i}{\partial t}}_{\substack{\text{Change in}\\ \text{density}\\ \text{with time}}} + \underbrace{\nabla \cdot \rho_i(r,t)\boldsymbol{v}}_{\substack{\text{Transport}\\ \text{with fluid}\\ \text{velocity}}} + \underbrace{\nabla \cdot j_i(r,t)}_{\substack{\text{Transport}\\ \text{with mass}\\ \text{diffusion}}} = \underbrace{R_i(r,t)}_{\substack{\text{Generation}\\ \text{from}\\ \text{reaction}}} \tag{2.21}$$

It is important to note that the velocity in the second term must be the averaged velocity for the concentration of interest. In Equation 2.21 for example, the concentration units are for mass concentration or density. This means that the velocity used in the second term must be the mass averaged velocity [9].

Equation 2.20 is simply the rate of generation for nuclide species i and can be plugged in for r_i in Equation 2.21. Before it can be plugged in, the units in Equation 2.20 need to be transformed from atoms per volume to mass per volume using Avogadros constant the molar

mass, $n_i = \rho_i N_A / M_i$. Using this relation and plugging in Equation 2.20 into 2.21 results in the MSR depletion equation,

$$\begin{aligned}\frac{\partial \rho_i}{\partial t} + \nabla \cdot \rho_i(r,t)\boldsymbol{v} + \nabla \cdot j_i(r) = \sum_{j=1}^{N} \frac{M_i}{M_j}\left(b_{j\to i}\lambda_j + \sum_{k=1}^{K} \gamma_{j\to i,k}\sigma_{k,j}(r)\phi(r,t)\right)\rho_j(r,t) \\ - \left(\lambda_i + \phi(r,t)\sum_{k=1}^{K}\sigma_{k,i}(r)\right)\rho_i(r,t).\end{aligned} \tag{2.22}$$

The MSR depletion equation represents the change in an isotopes density via fluid transport, mass diffusion and nuclear reactions. Equation 2.22 is a nonlinear PDE, first order in time and second order in space. It should be noted that Equation 2.20 was already discretized in space. For completeness $\sigma_{k,j}$ and ϕ are also functions of space and time in Equation 2.22 however, the scalar flux is for a single energy group. Solving Equation 2.22 is the primary topic and will be discussed through out the remainder of the dissertation.

2.3.2 Mass Transport Models

Fission products that are generated in MSRs are often categorized based on their phase and chemical behavior in the molten salt. The common categories are [10]:

1. Salt seekers
2. Noble metals
3. Noble gases

Salt seekers are fission products which form halide compounds which are soluble in the molten salt. Noble metals, are elements which under go redox reactions and are reduced to their neutral state. These fission products form metallic compounds which tend to collect on liquid-solid or liquid-gas interfaces. When collecting on liquid-solid surfaces within the flow loop, these fission products can plate a metallic film on the surface which can possibly work to reduce corrosion. Noble metals can also decay which doses the surface with radiation and decay heat. Noble gases such as Xe and Kr are gases which form in the liquid salt

from nuclear reactions. These elements can be dissolved in the liquid salt or extracted via gas stripping. In addition, these products may nucleate and/or become trapped in reactor equipment such as a graphite moderator.

A first order rate model is utilized in modeling mass transfer from liquid to solid and liquid to gas surfaces. At the interface, the flux normal to the surface is defined as [8]:

$$j = k_{\text{loc}} \Delta\rho, \tag{2.23}$$

where j is the mass flux [kg/s/m^2] and $k_{i,\text{loc}}$ is the local mass transfer coefficient [m/s] in terms of density. Depending of the direction of mass transfer the sign of j_i can be positive or negative based on the concentration gradient of $\Delta\rho_i$. On of the key features of using this previous relation is understanding and calculating the mass transfer coefficient k in its appropriate form. This coefficient will depend upon the fluid profile and geometry within the local region of mass transfer. There are many analytic and empirical relations that can be used to calculate k for different situations. An exhausted summary of these calculation methods can be found in Perry's Chemical Engineers' Handbook [11].

Equations of the form Eq. 2.23 are sufficient for approximating the mass transfer of a liquid to a solid surface and was utilized in modeling noble metal wall deposition during the molten salt reactor experiment (MSRE) [12]. For this case, noble metals are transported from the liquid phase to the sold surfaces within the reactor loop. This model is call wall deposition and is represented as:

$$j = k(\rho_{\text{bulk}} - \rho_{\text{surface}}). \tag{2.24}$$

Liquid gas transport is a bit more involved than the wall deposition model. For a liquid-gas system, the species fluxes across the interface as represented by the equation below:

$$j = k_G(\rho^g_{\text{bulk}} - \rho^g_i) = k_L(\rho^l_i - \rho^l_{\text{bulk}}), \tag{2.25}$$

where k_G is the gas-phase mass transfer coefficient, k_L is the liquid-phase mass transfer coefficient, ρ^g_{bulk} is the bulk concentration in the gas phase, ρ^l_{bulk} is the bulk concentration in the liquid phase, ρ^g_i is the concentration at the interface on the gas side, and ρ^l_i is the

concentration at the interface on the liquid side. Because the concentration at the interface of a liquid-gas system is difficult to establish, it is convenient to rewrite Eq. (2.25) as:

$$j = K_G(\rho^g_{\text{bulk}} - \rho^g_*) = K_L(\rho^l_* - \rho^l_{\text{bulk}}), \tag{2.26}$$

where k_G and k_L have been replaced by K_G and K_L, the overall mass transfer coefficients, ρ^g_* is the gas composition in equilibrium with the liquid, and ρ^l_* is the liquid composition in equilibrium with the gas. For liquid-gas systems, these equilibrium concentrations can be calculated using Henry's law.

When the gas has a low solubility in the liquid, the process is dominated by the liquid side resistance [8]. Both Xe and Kr were found to be only slightly soluble in salt during the MSRE [12]. Because of this, the mass transfer flux becomes:

$$j = k_l(\rho^l_* - \rho^l_{\text{bulk}}). \tag{2.27}$$

2.3.3 Boundary Conditions

Mathematically, there are three types of boundary conditions for Equation 2.22: Dirichlet, Neumann and Robin. Each of these boundary conditions are implemented in a way that permit inlet and outlet flow. Dirichlet boundary conditions specify a single constant value at the boundary. A Neumann boundary condition specifies the gradient normal to the boundary surface to be constant. The Robin boundary condition includes both the dependent variable and gradient are included. Table 2.3 summarizes each of the boundary conditions.

Table 2.3: Boundary Conditions

Boundary condition	**Form**
Dirichlet	$\rho_i(x_b) = C$
Neumann	$\frac{\partial \rho_i}{\partial x}(x_b) = C$
Robin	$a\rho_i(x_b) + b\frac{\partial \rho_i}{\partial x}(x_b) = C$

Chapter 3

Matrix Exponential Methods

The primary method for solving the MSR depletion equation is accomplished by transforming it into a system of first order ODEs. This Chapter presents a method called matrix exponential methods by the author. It is called by this name because it requires the computation of the exponential of a matrix in order to solve the system of ODEs.

Many methods exist for solving systems of first order differential equations. In vector matrix form, the problem involves solving a system of ordinary differential equations of the form,

$$\frac{d\boldsymbol{y}}{dt} = \boldsymbol{f}(t, \boldsymbol{y}), \quad \boldsymbol{y}(t_0) = \boldsymbol{y_0} \tag{3.1}$$

where $\boldsymbol{y}$, $\boldsymbol{f}(t, \boldsymbol{y})$ and $\boldsymbol{y}_0$ are column vectors and $\boldsymbol{f}(t, \boldsymbol{y})$ can contain linear and nonlinear terms.

$$\boldsymbol{y}(t) = \begin{bmatrix} y_1(t) \\ y_2(t) \\ \vdots \\ y_m(t) \end{bmatrix}, \quad \boldsymbol{f}(t, \boldsymbol{y}) = \begin{bmatrix} f_1(t, y_1, y_2, \ldots y_m) \\ f_2(t, y_1, y_2, \ldots y_m) \\ \vdots \\ f_m(t, y_1, y_2, \ldots y_m) \end{bmatrix}, \quad \boldsymbol{y}_0 = \begin{bmatrix} y_{1,0} \\ y_{2,0} \\ \vdots \\ y_{m,0} \end{bmatrix}$$

There are many popular methods to solve these equations including Euler, Runge–Kutta and multistep. These methods involve dividing the time domain into discrete lengths and

stepping through the domain by approximating each time step from the previous steps solution and a slope between the points. When the function vector $\boldsymbol{f}(t, \boldsymbol{y})$ contains terms that are stiff, these methods become computationally expensive and inaccurate for larger time steps [13] [14].

A more advantageous method for solving Equation 3.1 is the use of exponential time differencing [13] [15] [16]. To explain this method, Equation 3.1 is rewritten in a form that decomposes $\boldsymbol{f}(t, \boldsymbol{y})$ into two operators, one representing a constant linear operator and one for the nonlinear operator. Equation 3.2 shows this separation with $\boldsymbol{L}$ being the linear operator and $\boldsymbol{N}$ being the nonlinear.

$$\frac{d\boldsymbol{y}}{dt} = \boldsymbol{L}\boldsymbol{y} + \boldsymbol{N}(t, \boldsymbol{y}) \tag{3.2}$$

Solving Equation 3.2 using a class of methods involving matrix exponentials can be broken down into two main categories, integration factor and exponential time differencing. Both of the previously stated methods are discussed further in the next sections.

3.1 Integrating Factor Method

The integrating factor method begins by defining the following expression [17],

$$\boldsymbol{v} = e^{-\boldsymbol{L}t}\boldsymbol{y}, \tag{3.3}$$

where $e^{-\boldsymbol{L}t}$ is the integrating factor. Next, Equation 3.3 is differentiated with respect with time to give,

$$\frac{d\boldsymbol{v}}{dt} = -e^{-\boldsymbol{L}t}\boldsymbol{L}\boldsymbol{y} + e^{-\boldsymbol{L}t}\frac{d\boldsymbol{y}}{dt}. \tag{3.4}$$

Multiplying Equation 3.2 by the integrating factor and bringing the linear operator to the right hand side gives,

$$e^{-\boldsymbol{L}t}\frac{d\boldsymbol{y}}{dt} - e^{-\boldsymbol{L}t}\boldsymbol{L}\boldsymbol{y} = e^{-\boldsymbol{L}t}\boldsymbol{N}(t,\boldsymbol{y}) = \frac{d\boldsymbol{v}}{dt}. \tag{3.5}$$

This brings the final form of the equation to solve,

$$\frac{d\boldsymbol{v}}{dt} = e^{-\boldsymbol{L}t}\boldsymbol{N}(t, e^{\boldsymbol{L}t}\boldsymbol{v}), \tag{3.6}$$

or,

$$\frac{d\boldsymbol{v}}{dt} = \boldsymbol{f}(t,\boldsymbol{v}). \tag{3.7}$$

Solving Equation 3.7 can be done using any usual Runge–Kutta, or multistep method. For example, the fourth order Runge-Kutta method gives the following formula [17].

$$\begin{aligned}
k_1 &= h\boldsymbol{f}(t_n, \boldsymbol{v}_n)\\
k_2 &= h\boldsymbol{f}(t_n + h/2, \boldsymbol{v}_n + k_1/2)\\
k_3 &= h\boldsymbol{f}(t_n + h/2, \boldsymbol{v}_n + k_2/2)\\
k_4 &= h\boldsymbol{f}(t_n + h, \boldsymbol{v}_n + k_3)\\
\boldsymbol{v}_{n+1} &\approx \boldsymbol{v}_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)\\
\boldsymbol{y}_{n+1} &= e^{\boldsymbol{L}(t_n+h)}\boldsymbol{v}_{n+1}
\end{aligned} \tag{3.8}$$

Integrating factor methods have the property of being exact when $\boldsymbol{N}(t,\boldsymbol{y}) = 0$ [13].

3.2 Exponential Time Differencing

Exponential time differencing methods are very similar to the integrating factor method except for how it handles the nonlinear portion of Equation 3.2. Deriving the exponential time differencing formula begins by multiplying Equation 3.2 by the integrating factor,

$$e^{-\boldsymbol{L}t}\left(\frac{d\boldsymbol{y}}{dt} - \boldsymbol{L}y\right) = e^{-\boldsymbol{L}t}\boldsymbol{N}(t, \boldsymbol{y}), \tag{3.9}$$

using the product rule this can be written as,

$$\frac{d}{dt}\left(e^{-\boldsymbol{L}t}\boldsymbol{y}\right) = e^{-\boldsymbol{L}t}\boldsymbol{N}(t, \boldsymbol{y}). \tag{3.10}$$

Next the function is integrated from a point t_n to $t_n + \Delta t$ where $\Delta t = t_{n+1} - t_n$,

$$e^{-(t_n+\Delta t)\boldsymbol{L}}\boldsymbol{y}(t_n + \Delta t) - e^{-t_n\boldsymbol{L}}\boldsymbol{y}(t_n) = \int_{t_n}^{t_n+\Delta t} e^{-\boldsymbol{L}t}\boldsymbol{N}(t, \boldsymbol{y})dt. \tag{3.11}$$

Let $t = t_n + \tau \quad dt = d\tau \quad \tau \in [0, \Delta t]$, applying these change of variables gives [18],

$$e^{-(t_n+\Delta t)\boldsymbol{L}}\boldsymbol{y}(t_n + \Delta t) - e^{-t_n\boldsymbol{L}}\boldsymbol{y}(t_n) = \int_{0}^{\Delta t} e^{-\boldsymbol{L}(t_n+\tau)}\boldsymbol{N}(t_n + \tau, \boldsymbol{y}(t_n + \tau))d\tau. \tag{3.12}$$

Because t_n, τ and Δt are scalars, the exponential can be written as,

$$\begin{aligned} e^{-\boldsymbol{L}(t_n+\tau)} &= e^{-\boldsymbol{L}t_n}e^{-\boldsymbol{L}\tau} \\ e^{-(t_n+\Delta t)\boldsymbol{L}} &= e^{-\boldsymbol{L}t_n}e^{-\boldsymbol{L}\Delta t}. \end{aligned} \tag{3.13}$$

Applying to Equation 3.12, $e^{\boldsymbol{L}t_n}$ cancels out on both sides. This leads to,

$$e^{-\Delta t\boldsymbol{L}}\boldsymbol{y}(t_n + \Delta t) - \boldsymbol{y}(t_n) = \int_{0}^{\Delta t} e^{-\boldsymbol{L}\tau}\boldsymbol{N}(t_n + \tau, \boldsymbol{y}(t_n + \tau))d\tau, \tag{3.14}$$

moving $\boldsymbol{y}(t_n)$ to the right hand side and multiplying both sides by $e^{\Delta t\boldsymbol{L}}$ gives the final result,

$$\boldsymbol{y}(t_n + \Delta t) = e^{\Delta t \boldsymbol{L}} \boldsymbol{y}(t_n) + e^{\Delta t \boldsymbol{L}} \int_0^{\Delta t} e^{-\boldsymbol{L}\tau} \boldsymbol{N}(t_n + \tau, \boldsymbol{y}(t_n + \tau)) d\tau, \tag{3.15}$$

where $\boldsymbol{L}$ is called the transition matrix. This formalization is exact and exponential time differencing methods work to approximate the integral of the nonlinear portion. Exponential time differencing methods have the property of being exact when $\boldsymbol{N}(\boldsymbol{y}, t) = \text{constant}$ [13].

Evaluating the integral in Equation 3.15 can be done using traditional multistep methods or with Runge-Kutta methods [15]. For example, a fourth order Runge-Kutta approximation for the integral gives the following formula for Equation 3.15 [15],

$$\begin{aligned}
\boldsymbol{a}_n &= e^{\boldsymbol{L}\Delta t/2} \boldsymbol{y}(t_n) + \boldsymbol{L}^{-1}\big(e^{\boldsymbol{L}\Delta t/2} - \boldsymbol{I}\big) \boldsymbol{N}(t_n, \boldsymbol{y}(t_n)) \\
\boldsymbol{b}_n &= e^{\boldsymbol{L}\Delta t/2} \boldsymbol{y}(t_n) + \boldsymbol{L}^{-1}\big(e^{\boldsymbol{L}\Delta t/2} - \boldsymbol{I}\big) \boldsymbol{N}(t_n + \Delta t/2, \boldsymbol{a}_n) \\
\boldsymbol{c}_n &= e^{\boldsymbol{L}\Delta t/2} \boldsymbol{a}_n + \boldsymbol{L}^{-1}\big(e^{\boldsymbol{L}\Delta t/2} - \boldsymbol{I}\big)\big(2\boldsymbol{N}(t_n + \Delta t/2, \boldsymbol{b}_n) - \boldsymbol{N}(t_n, \boldsymbol{y}(t_n))\big) \\
\boldsymbol{y}(t_{n+1}) &= e^{\boldsymbol{L}\Delta t} \boldsymbol{y}(t_n) + \Delta t^{-2} \boldsymbol{L}^{-3}\big([-4\boldsymbol{I} - \Delta t\boldsymbol{L} + e^{\boldsymbol{L}\Delta t}(4\boldsymbol{I} - 3\Delta t\boldsymbol{L} + (\Delta t\boldsymbol{L})^2)]\boldsymbol{N}(t_n, \boldsymbol{y}(t_n))\big) \\
&\quad + 2[2\boldsymbol{I} + \Delta t\boldsymbol{L} + e^{\boldsymbol{L}\Delta t}(-2\boldsymbol{I} + \Delta t\boldsymbol{L})](\boldsymbol{N}(t_n + \Delta t/2, \boldsymbol{a}_n) + \boldsymbol{N}(t_n + \Delta t/2, \boldsymbol{b}_n)) \\
&\quad + [-4\boldsymbol{I} - 3\Delta t\boldsymbol{L} - (\Delta \boldsymbol{L})^2 + e^{\boldsymbol{L}\Delta t}(4\boldsymbol{I} - \Delta t\boldsymbol{L})]\boldsymbol{N}(t_n + \Delta t, \boldsymbol{c}_n)\big)
\end{aligned} \tag{3.16}$$

3.3 Stability of the Transition Matrix in Linear Systems

Particular situations arise in systems for which Equation 3.2 becomes linear when $\boldsymbol{N}(t, \boldsymbol{y} = 0$. In this scenario, the system becomes:

$$\frac{d\boldsymbol{t}}{dt} = \boldsymbol{L}\boldsymbol{y}. \tag{3.17}$$

The existence of the solution to the initial value problem initial value problem $\boldsymbol{y}(t_0 = \boldsymbol{y}_0$ is guaranteed if the coefficients of $\boldsymbol{L}$ are continuous of a common interval I that contains the point t_0 [19]. The stability of this problem can be accessed by examining the eigenvalues of the linear operator $\boldsymbol{L}$.

3.4 Solutions to the Matrix Exponential

When obtaining solutions based on exponential time differencing, an exponential of a matrix needs to be computed. There are multiple computational methods for solving for the matrix exponential, many of them are developed specifically to evaluate $e^{\boldsymbol{A}t}$ or $e^{\boldsymbol{A}t}\boldsymbol{v}$. One evaluates the matrix exponential directly and the other calculates the action of the exponential on a vector. Because of the way in which the linear systems operate in EDT methods, only the action of the matrix on the vector is required. However, in some of the methods that will be presented only direct evaluation of the matrix exponential is possible.

Computation of the exponential of a matrix is by far the most difficult part of EDT methods. Not only is the basis for many of the methods mathematically in depth, but they are difficult to deploy on a level that makes the computation of the matrix exponential both fast and accurate. Numerous methods for computing the matrix exponential are discussed here however, they are not limited to the ones presented in this work.

3.4.1 Series Approximations Near the Origin

Solvers of this nature often exploit the following relation when computing the matrix exponential:

$$e^{\boldsymbol{A}t} = \left(e^{\boldsymbol{A}t/m}\right)^m, \tag{3.18}$$

where m is a scalar that *scales* matrix $\boldsymbol{A}t$. This method is known as scaling and squaring. The reason this is done is to reduce the norm of matrix $\boldsymbol{A}t$, especially in situations for $\boldsymbol{A}t$ when $t \to \infty$ as previously mentioned. For series methods near the origin, the accuracy of the method diminishes as the matrix norm increases.

Taylor Series Expansion

Formally, matrix exponential is defined using an infinite Taylor series [20] [21] [6].

$$e^{\boldsymbol{A}} = \sum_{k=0}^{\infty} \frac{1}{k!}\boldsymbol{A}^k \tag{3.19}$$

Therefore, a straight forward way to calculate the matrix exponential is using its formal definition. This method however, is not commonly used in application for either the matrix or the scalar case. The number of terms required to achieve convergence can be large and produce computational inefficiency. This method is also suffers from numerical round off errors from cancellation for large values of k [21]. There is one method in libowski based off of this method. While not commonly used in practice an algorithm for computing the action of the matrix exponential of matrix $\boldsymbol{A} \in \mathbb{C}^{n\times n}$ on matrix $\boldsymbol{B} \in \mathbb{C}^{n\times n_0}$ where $n_0 << n$ was developed by Al-Mohy and Higham based on a truncated Taylor series [22]:

$$e^{\boldsymbol{A}}\boldsymbol{B} = (e^{2^{-s}\boldsymbol{A}})^{2^s}\boldsymbol{B} \approx T_m(2^{-s}\boldsymbol{A})^{2^s}\boldsymbol{B}, \tag{3.20}$$

where T_m is a truncated Taylor series of order m. Unlike the other matrix exponential algorithms that will be presented, the Taylor series does not require linear solves. This method was developed using a backward error analysis based on $||\boldsymbol{A}^{1/k}||^{1/k}$, keeping in mind the computational cost. To reduce the norm of $\boldsymbol{A}$, the Taylor method uses two key preprocessing steps. These steps include shifting and optional balancing.

Padé Approximation

The Padé approximation represents a function by expanding it as a ratio of two power series. A (p, q) Padé approximation for $e^{\boldsymbol{A}t}$ is defined by [21]:

$$e^{\boldsymbol{A}t} \approx R_{p,q}(\boldsymbol{A}t) = \frac{N_{p,q}(\boldsymbol{A}t)}{D_{p,q}(\boldsymbol{A}t)} \tag{3.21}$$

where

$$N_{p,q}(\boldsymbol{A}t) = \sum_{j=0}^{p} \frac{(p+q-j)!p!}{(p+q)!j!(p-j)!}(\boldsymbol{A}t)^j, \tag{3.22}$$

$$D_{p,q}(\boldsymbol{A}t) = \sum_{j=0}^{q} \frac{(p+q-j)!q!}{(p+q)!j!(q-j)!}(-\boldsymbol{A}t)^j. \tag{3.23}$$

Padé methods are similar to Taylor series as they approximate a function using a series solution, however, Padé series usually out preform Taylor series. Series solutions methods,

such as Padé are also more accurate near the origin, meaning that the matrix norm $||\boldsymbol{A}t||$ must be sufficiently small for the approximation to be accurate [6]. Yet another problem arises when $\boldsymbol{A}$ has a wide spread of eigenvalues, causing an ill-conditioned linear system [20] [21].

There are many ways to develop an algorithm for computing the matrix exponential using the Padé approximation [20] [23] [24]. The key in deriving an algorithm is in both understanding the error associated with the size of the matrix norm and limiting the computation time. Moler and Van Loan derived an elegant and simple proof for determining values for p, q and m given a matrix norm [21]. As noted by Higham [23], this derivation contained weaknesses. Moler and Van Loan assumed that the matrix norm needed to be less than one half ($||\boldsymbol{A}t|| < 1/2$) but Higham proved that this was not the case. Higham further showed that the required minimal matrix norm is different for each order of the Padé implementation. Above this norm, a higher order Padé approximation would be required, or matrix scaling would need to be take place. One other weakness was the derivation of their error bound. It was designed to be easily commutable, which resulted in their error bound not being sharp [23]. When the error bound is not sharp, then it is possible to overscale the matrix, resulting in a loss of accuracy. Higham and Al-Mohy. described two algorithms to resolve the overscaling problem found in Ref. ([23]) and Ref. ([24]). Reference ([24]) is an updated algorithm to the one described in Ref. ([23]), which works to fix the overscaling problem.

The Padé approximation in combination with scaling and squaring is probably the most widely used method for computing the exponential of a matrix. In fact, the *expm* function in MATLAB is based on this approach [24]. Two methods based on the Padé approach are implemented in libowski, *Padé - Method 1* and *Padé - Method 2*. Both algorithms work to form a Padé approximation of type $p = q$ and will further be denoted by m, combined with scaling and squaring:

$$e^{\boldsymbol{A}} = (e^{2^{-s}\boldsymbol{A}})^{2^s} \approx r_m(2^{-s}\boldsymbol{A})^{2^s}, \tag{3.24}$$

where s is the scaling and squaring parameter and $\boldsymbol{A}$ is $\boldsymbol{A}t$. The scaling parameter s is chosen to that the exponential is computed with a backward error bounded by the unit roundoff. Method - 1 is based on the algorithm developed by Higham in Ref. ([23]), in which he backwards error is based on $||\boldsymbol{A}||$. Method - 2 is based on Ref. ([24]) which worked to reduce the problem of over scaling in the Method - 1 by tightening the backwards error based on $||\boldsymbol{A}^{1/k}||^{1/k}$ where:

$$||\boldsymbol{A}^{1/k}||^{1/k} \leq ||\boldsymbol{A}||, \quad k = 1 : \infty. \tag{3.25}$$

3.4.2 Solutions Based on the Cauchy Integral Formula

This method is based on transforming the matrix exponential into the complex plane using the Cauchy integral formula. The matrix exponential becomes,

$$e^{\boldsymbol{A}} = \frac{1}{2\pi i} \int_{\Gamma} e^{z} (z\boldsymbol{I} - \boldsymbol{A})^{-1} dz, \tag{3.26}$$

where $\boldsymbol{A}$ is analytic inside the closed contour Γ that winds once around the eigenvalues of $\boldsymbol{A}$ [1] [25] [26]. In practice, it is often required to evaluate the action of the matrix exponential on vector $\boldsymbol{v}$, this leads to evaluating,

$$e^{\boldsymbol{A}}\boldsymbol{v} = \frac{1}{2\pi i} \int_{\Gamma} e^{z} (z\boldsymbol{I} - \boldsymbol{A})^{-1} \boldsymbol{v} dz. \tag{3.27}$$

When evaluating Equations 3.26 or 3.27 it is vital to select a contour that encloses the spectrum of $\boldsymbol{A}t$. If this is not done, then the approximation is inaccurate. There are a number of ways to solve Equations 3.26 or 3.27 which involve knowing the nature of the spectrum of the transition matrix $\boldsymbol{A}t$.

Rational Approximation

A more generalized method for evaluating the contour integral can be accomplished by choosing an analytic function $\phi(\theta)$ that maps the real line onto the contour. Because the function $e^{\phi(\theta)}$ decreases exponentially as $|\theta| \to \infty$ the approximation can be truncated to a finite number of quadrature points. When the spectrum of the transition matrix falls on

the left hand side of the complex plane close to the real axis then the contour Γ denotes a Hankel like contour that winds from $-\infty - 0i$ on the lower half-plane and $\infty + 0i$ on the upper half-plane. [26]. This allows for the definition of a general contour function that will enclose the eigenvalues on the left hand side of the complex plane around the negative real axis.

The idea is to solve the contour integral of the form,

$$I = \frac{1}{2\pi i} \int_{-\infty}^{\infty} e^{\phi(\theta)} f(\phi(\theta))\phi'(\theta) d\theta. \tag{3.28}$$

The trapezoidal approximation with N points to Equation 3.28 becomes,

$$I_N = \frac{-i}{N} \sum_{k=1}^{N} e^{z_k} f(z_k) w_k, \tag{3.29}$$

where $z_k = \phi(\theta_k)$ and $w_k = \phi'(\theta_k)$ [26]. It can be shown that Equation 3.29 can be written in the form,

$$I_N = \frac{1}{2\pi i} \int_C r(z) f(z) dz, \quad r(z) = \sum_{k=1}^{N} \frac{c_k}{z - z_k}, \quad c_k = iN^{-1} e^{z_k} w_k, \tag{3.30}$$

where C is a contour that winds around each point z_k [26]. The points z_k and c_k are interpreted as the poles and residues of the rational function. Equation for $r(z)$ is a good approximation to e^z near the negative real axis. The error of the quadrature estimate is,

$$I - I_N = \frac{1}{2\pi i} \int_{\Gamma'} \big(e^z - r(z)\big) f(z) dz. \tag{3.31}$$

For a contour function $\phi(\theta)$ the goal is to find the rational function $r(z)$ that minimizes the error.

Trefethen noted three contour functions to Γ, two of the three are presented here [26]. The simplest contour function is a parabola defined by,

$$\phi = N[0.1309 - 0.1194\theta^2 + 0.2500i\theta] \tag{3.32}$$

which has a convergence rate of $\mathcal{O}(2.85^{-N})$. Accuracy of about 14 or more digits can be achieved with N = 32. The approximation of e^z on the complex plane is shown in Figure 3.1. The second contour function is that of a hyperbola defined by,

$$\phi = 2.246N[1 - \sin(1.1721 - 0.3443i\theta)] \tag{3.33}$$

which has a convergence rate of $\mathcal{O}(3.20^{-N})$, with an accuracy of about 16 or more digits with N = 32. The same approximation for e^z on the complex plane is shown in Figure 3.2. Both Figures 3.1 and 3.2 show high levels of accuracy not only on the negative real access but also for a wide region of the left hand side of the complex plane.

Applying these approximations to real valued scalars or matrices requires half the amount of computational cost, this is because the poles of a rational function with real valued coefficients form conjugate pairs [25]. For a real scalar, the rational approximation is,

$$e^x \approx r(x) = 2Re\left(\sum_{k=1}^{N/2} \frac{c_k}{x - z_k}\right). \tag{3.34}$$

When applying the rational function to a real valued matrix, the approximation becomes,

$$e^{\boldsymbol{A}} \approx r(\boldsymbol{A}) = 2Re\left(\sum_{k=1}^{N/2} c_k(\boldsymbol{A} - z_k\boldsymbol{I})^{-1}\right), \tag{3.35}$$

requiring $N/2$ matrix inversions. If instead the action on the matrix exponential on a vector $\boldsymbol{v}$ is calculated, the approximation becomes,

$$e^{\boldsymbol{A}}\boldsymbol{v} \approx r(\boldsymbol{A})\boldsymbol{v} = 2Re\left(\sum_{k=1}^{N/2} c_k(\boldsymbol{A} - z_k\boldsymbol{I})^{-1}\boldsymbol{v}\right), \tag{3.36}$$

requiring $N/2$ solves of the linear system $\boldsymbol{x} = (\boldsymbol{A} - z_k\boldsymbol{I})^{-1}c_k\boldsymbol{v}$. It is important to note that each of these linear systems or matrix inversions are independent of one another and can be done in parallel.

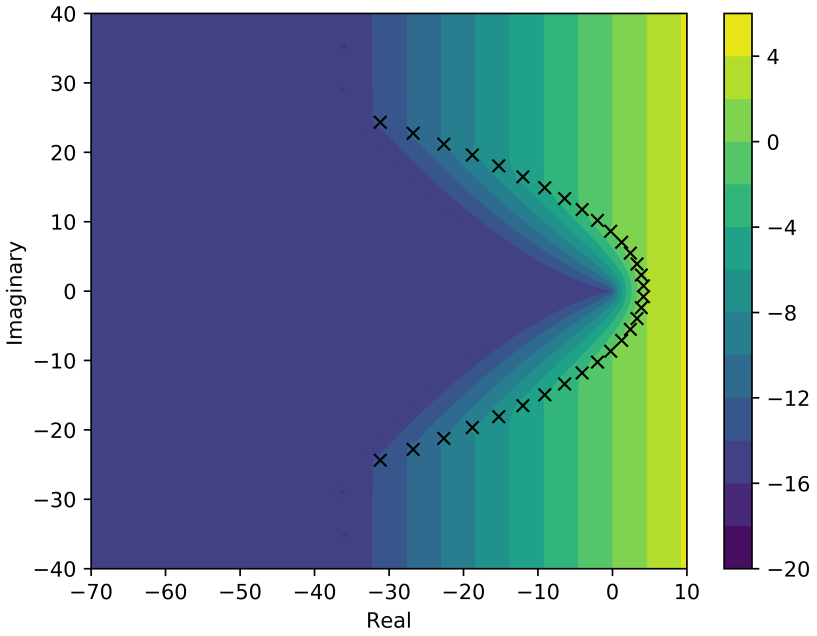

Figure 3.1: $\log_{10}|r(z)-e^z|$ for N = 32 where the contour is defined by a parabola, Equation 3.32. Quadrature points are denoted with x's

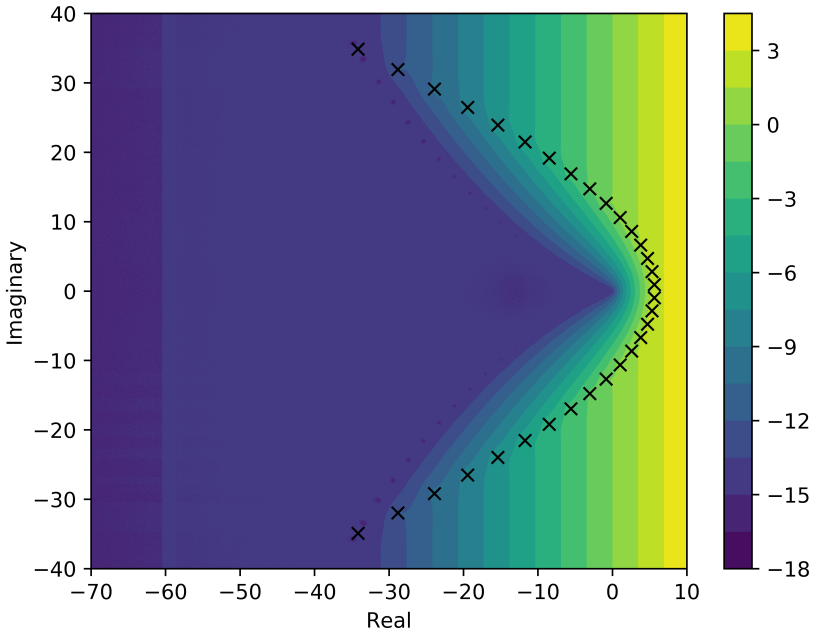

Figure 3.2: $\log_{10}|r(z)-e^z|$ for N = 32 where the contour is defined by a hyperbola, Equation 3.33. Quadrature points are denoted with x's

Best Approximations

A different approach is to choose a function $r(z)$ that is the best approximation of the exponential function on the negative real axis, doing this bypasses the need for a contour function [1] [26]. This method is known as the Chebychev Rational Approximation Method (CRAM) and is done by finding a unique rational function $\hat{r}_{k,k} = \hat{p}_k(x)/\hat{q}_k(x)$ satisfying

$$\epsilon_{k,k} \equiv \sup_{x\in\mathbb{R}_-} |\hat{r}_{k,k}(x) - e^x| = \inf_{r_{k,k}\in\pi_{k,k}} \left\{ \sup_{x\in\mathbb{R}_-} |r_{k,k}(x) - e^x| \right\}. \tag{3.37}$$

Applying this definition leads to the follows form on the rational approximation,

$$e^z \approx r(z) = c_0 + \sum_{k=1}^{N} \frac{c_k}{z - z_k} \tag{3.38}$$

where c_0 is the limit at infinity. This formation is the same as the rational approximation with $c_0 = 0$. The convergence rate for CRAM is of the order $\mathcal{O}(9.28903^{-N})$, which is remarkably faster than those previously shown. With $N = 16$ quadrature points CRAM gives about 15 or more digits of accuracy. Thus the same order of accuracy can be achieved with half the number of quadrature points than the rational approximations defined by contour functions. For a more detailed explanation the CRAM algorithm please refer to Reference [1].

The difficulty with using the CRAM approximation is finding the coefficient for the rational approximation. For CRAM of order 14 and 16 the rational coefficient can be found in Reference [25] up to 20 digits. Figure 3.3 shows the accuracy of CRAM to the function e^z on the complex plane. Because the rational function was built in such a way to be the best approximation on the negative real axis, the accuracy of CRAM is in a more narrow range of the real axis. For a real values matrix the CRAM algorithm leads to the following solution,

$$e^{\boldsymbol{A}} \approx r(\boldsymbol{A}) = c_0 + 2Re\left(\sum_{k=1}^{N/2} c_k(\boldsymbol{A} - z_k\boldsymbol{I})^{-1} \right), \tag{3.39}$$

and for the action of the matrix on a vector,

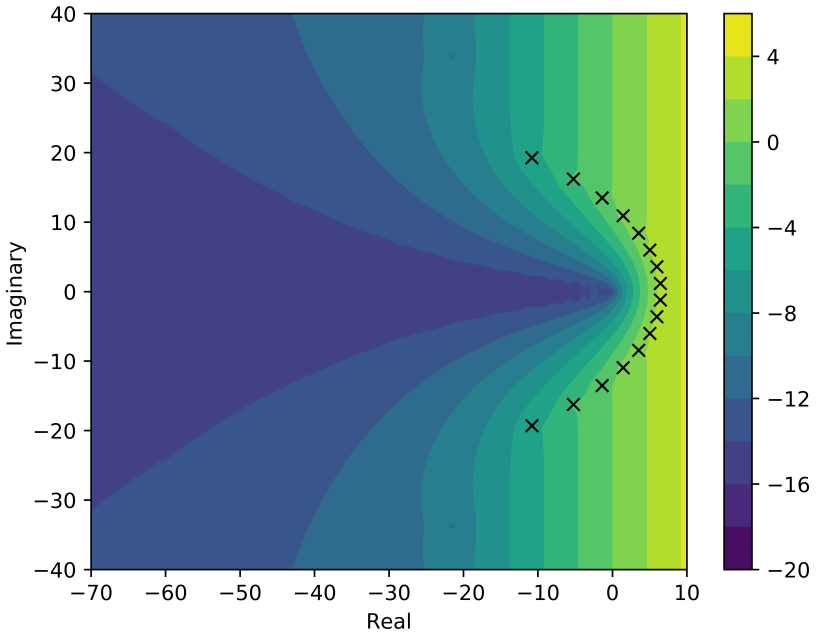

Figure 3.3: $\log_{10}|r(z) - e^z|$ for CRAM with N = 16. Quadrature points are denoted with x's

$$e^{\boldsymbol{A}}\boldsymbol{v} \approx r(\boldsymbol{A})\boldsymbol{v} = c_0\boldsymbol{v} + 2Re\left(\sum_{k=1}^{N/2} c_k(\boldsymbol{A}t - z_k\boldsymbol{I})^{-1}\boldsymbol{v}\right). \tag{3.40}$$

Accuracy of Rational Approximations

While the matrix exponential can be formally defined by Eq. (3.19), when examining the accuracy of rational approximations presented, it is more useful to define the matrix exponential in terms of the transition matrix's eigenvalues. Because the contour Γ must wind around the eigenvalues of $\boldsymbol{A}t$, if $\boldsymbol{A}$ has eigenvalues which have non-trivial imaginary parts then these eigenvalues will scale as a function of t. If these eigenvalues scale to a portion of the complex plane which fall outside the contour, then the accuracy of these methods can be compromised. For traditional burnup matrices it was stated earlier that the eigenvalues are clustered around the negative real axis [6]. Because of this, the accuracy of CRAM as well as the other two rational approximations are not a function of time t [27]. For burnup calculations in MSRs defined by Eq. (2.22) the eigenvalues are not necessarily clustered

around the negative real axis as they have non-trival imaginary parts. Through numerical experiments is was found by the author that these imaginary parts were introduced by the addition of convection terms in the transition matrix. In addition, these imaginary parts were found to be correlated with the ratio the velocity to the spatial discretization as well as the boundary conditions.

While rational approximations such as CRAM have shown fantastic results in burnup calculations, is has been shown that the relative accuracy of CRAM diminishes when the nuclide concentration diminishes significantly over the time step. For a nuclide concentration $n_i(t) << n_i(0)$ the error estimate for CRAM follows [28]:

$$\frac{\delta \hat{n}_i(t)}{n_i(t)} = \hat{\varepsilon}_{k,k} \frac{n_i(0)}{n_i(t)} \tag{3.41}$$

This means that for CRAM of order k concentration smaller than $\hat{\varepsilon}_{k,k} n_i(0)$ is not captured by the rational approximation. Additionally the accuracy of CRAM is diminished when performing calculations on fresh fuel versus depleted fuel. For fresh fuel, the concentrations of nuclides is dependent on transitions corresponding the only a few nuclides in the initial condition. If CRAM introduces large errors in approximating the matrix elements of a few transitions which influence the production of a large number of nuclides then the relative error in calculating nuclides concentrations diminishes. For used fuel depletion these errors are averaged out to provide an overall more accurate calculation. While this error was derived for transition matrices built from solving tradition burnup calculations Eq. (4.2) it is assumed that the same holds true for MSR calculations of the form Eq. (2.22). In order to increase the accuracy of CRAM methods for depletion calculations a substepping approach was taken and introduced into the ORIGEN library [28]. This substepping method is also utilized in our implementation of the CRAM algorithm.

It is important to note that the absolute error in computing rational approximations of the from parabolic or hyperbolic contours do not follow the same error as CRAM. In the scalar case of CRAM, if one plots the absolute error on the negative real axis as $x \to -\infty$, absolute error asymptotically approaches $\hat{\varepsilon}_{k,k}$. This is because the exponential function tends to zero while CRAM stabilizes at $\hat{\varepsilon}_{k,k}$. As $x \to 0$ the absolute error oscillates between $-\hat{\varepsilon}_{k,k}$

and $\hat{\varepsilon}_{k,k}$ [27]. While the errors for rational functions based on quadrature contours do not necessarily follow this same behavior, the substepping method was implemented and is later shown to also increase their accuracy.

Substepping is implemented by scaling the time step size and evaluate the solution from the previous step, shown in Algorithm 1, where m is the number of substeps to be taken.

Algorithm 1 Substeping

1: $\boldsymbol{v}_0 = \boldsymbol{\rho}_0$
2: $t = t/(m+1)$
3: **for** $j = 0, 1, \ldots m$ **do**
4: $\quad \boldsymbol{\rho}_{m+1} = r(\boldsymbol{A}t)\boldsymbol{v}_0$
5: $\quad \boldsymbol{v}_0 = \boldsymbol{\rho}_{m+1}$
6: **end for**

3.4.3 Krylov Subspace Method

Krylov subpace approximations are a class of popular methods utilized in sparse matrix algorithms. The idea of Krylov subspace methods is to project the sparse $n \times n$ $\boldsymbol{A}$ matrix into a lower-dimensional subspace. The new lower dimension projection is of size $m \times m$ where $m < n$. Because the matrix is of lower dimension, calculating its matrix exponential becomes much faster. It is important to note that Krylov subspace methods can only be used as an operation on a vector, the direct calculation of the matrix exponential is not possible [29].

Consider we want to approximate the matrix exponential as a polynomial of order $m-1$, this takes the form,

$$e^{\boldsymbol{A}}\boldsymbol{v} \approx p_{m-1}(A)\boldsymbol{v}. \tag{3.42}$$

This approximation is an element of the Krylov subspace defined by,

$$K_m = \text{span}\{\boldsymbol{v}, \boldsymbol{A}\boldsymbol{v}, \boldsymbol{A}^2\boldsymbol{v}, ..., \boldsymbol{A}^{m-1}\boldsymbol{v}\}. \tag{3.43}$$

For a general non-symmetric matrix the Arnoldi algorithm can be utilized in building the Krylov space [29] [30]. Algorithm 2 constructs an orthonormal bases $\boldsymbol{V}_m = [\boldsymbol{v}_1, \boldsymbol{v}_2, ...\boldsymbol{v}_m]$

of the Krylov subspace, and an $m \times m$ upper Hessenberg matrix. The Arnoldi algorithm produces the following relation,

$$\boldsymbol{A}\boldsymbol{V}_m = \boldsymbol{V}_m\boldsymbol{H}_m + h_{m+1,m}\boldsymbol{v}_{m+1}\boldsymbol{e}_m^T \tag{3.44}$$

Algorithm 2 Arnoldi

```
Compute v_1 = v/||v||_2
for j = 1, 2, ... m do
    Compute w = A v_j
    for i = 1, 2, ... j do
        Compute h_{i,j} = (w, v_i)
        Compute w = w − h_{i,j} v_i
    end for
    Compute h_{j+1,j} = ||w||_2 and v_{j+1} = w/h_{j+1,j}
end for
```

where $\boldsymbol{H}_m = \boldsymbol{V}_m^T\boldsymbol{A}\boldsymbol{V}_m$ and $\boldsymbol{e}_m$ is the unit vector of dimention m. The Hessenberg matrix $\boldsymbol{H}_m$ represents the projection of $\boldsymbol{A}$ on to the Krylov subspace. The approximation in Krylov space is known to be,

$$e^{\boldsymbol{A}}\boldsymbol{v} \approx \beta\boldsymbol{V}_m e^{\boldsymbol{H}_m}\boldsymbol{e}_1 \tag{3.45}$$

where $\beta = ||\boldsymbol{v}||_2$ [30]. The computation of $e^{\boldsymbol{H}_m}$ becomes much easier because $\boldsymbol{H}_m$ is dense and smaller than $\boldsymbol{A}$. After the Krylov approximation is made, a typical method for solving the matrix exponential is used on $e^{\boldsymbol{H}_m}$. The quality of this approximation is exact when $n = m$. This comes from the fact that at step m, $h_{m+1,m} = 0$ and Equation 3.44 becomes,

$$\boldsymbol{A}\boldsymbol{V}_m = \boldsymbol{V}_m\boldsymbol{H}_m. \tag{3.46}$$

The Arnoldi process with be exact after m steps when m is greater to or equal to the degree of the minimal polynomial in Equation 3.42. At this point Equation 3.42 is exact, however this is unlikely to happen until $m = n$ [29] [30].

The general error associated with applying the approximation in Equation 3.45 can be proven to be,

$$||e^{\boldsymbol{A}}\boldsymbol{v} - \beta \boldsymbol{V}_m e^{\boldsymbol{H}_m}\boldsymbol{e}_1||_2 \leq 2\beta \frac{\rho^m e^{\rho}}{m!}, \tag{3.47}$$

where $\rho = ||A||_2$ [29]. The error bounds derived by Saad in [29] can be computed but might not be sharp, especially when the norm of the matrix is large. In practice, a more useful posteriori error can be used to determine the error for applying the Arnoldi algorithm. This error is found after applying the Arnoldi algorithm and is defined by,

$$||e^{\boldsymbol{A}}\boldsymbol{v} - \beta \boldsymbol{V}_m e^{\boldsymbol{H}_m}\boldsymbol{e}_1||_2 \approx h_{m+1,m}|\boldsymbol{e}_m^T \phi(\boldsymbol{H}_m)\beta \boldsymbol{e}_1|, \tag{3.48}$$

where $\phi(\boldsymbol{A}) = \boldsymbol{A}^{-1}[e^{\boldsymbol{A}} - \boldsymbol{I}]$. This error estimate was found to be sufficient enough for practical applications [29].

Krylov subpsace methods are good for approximating the largest eigenvalues of a matrix because of the continued multiplication of $\boldsymbol{A}$ when computing the orthonormal basis [7]. As the spread of the eigenvalues for $\boldsymbol{A}$ increases the accuracy of the Krylove subspace decreases, even as you increase the dimension of the subspace [6].

Chapter 4

Application of Matrix Exponential Methods to Burnup Calculations

Chapter 3 laid out the fundamental mathematical framework on which Libowski is built. Using this framework, the chemical species transport equation is solved for the time dependent solution of radionuclides in liquid fueled molten salt reactors. This chapter explains the practical application of such methods described in Chapter 3 along with the method of lines, to solve the species transport equation. First, the traditional nuclear burnup equations for LWRs are presented, along with their modern day solution in existing reactor depletion codes. Second, the method of lines is presented and used to transform a PDE into a system of coupled ODE's. Finally, some general analysis is conducted on this new governing set of ODE's, to better understand their behavior. This behavior is especially important when choosing one of the matrix exponential solvers described in Chapter 3.

4.1 Application to the Traditional Nuclear Burnup Equations

There are many production level software packages which are used to solve these equations, such as SCALE and Serpent. Both of these codes rely on the CRAM method, the reason for this will be described in further detail.

Nuclear burnup calculations involve solving a set of first order linear ODEs of the form,

$$\frac{dn_i}{dt} = \sum_{j=1}^{N} \left(b_{j\to i}\lambda_j + \sum_{k=1}^{K} \gamma_{j\to i,k}\sigma_{k,j}\phi \right) n_j(t) - \left(\lambda_i + \phi \sum_{k=1}^{K} \sigma_{k,i} \right) n_i(t), \tag{4.1}$$

which can be written in matrix vector form,

$$\frac{d\boldsymbol{n}}{dt} = \boldsymbol{A}\boldsymbol{n}, \quad \boldsymbol{n}(t_0) = \boldsymbol{n}_0, \tag{4.2}$$

where, $\boldsymbol{n}(t)$ is the nuclide concentration vector, $\boldsymbol{A}$ is the transition matrix and $\boldsymbol{n}_0$ is the initial condition vector. Equation 4.2 has the solution $\boldsymbol{n}(t) = e^{\boldsymbol{A}t}\boldsymbol{n}_0$ where $e^{\boldsymbol{A}t}$ was defined in Chapter 3. The transition matrix contains the decay and transmutation coefficients,

$$a_{i,i} = -\left(\lambda_i + \phi \sum_{k=1}^{K} \sigma_{k,i} \right), \tag{4.3}$$

$$a_{i,j\neq i} = b_{j\to i}\lambda_j + \sum_{k=1}^{K} \gamma_{j\to i,k}\sigma_{k,j}\phi. \tag{4.4}$$

While it is possible to order index the nuclides in any order, the transition matrix becomes nearly upper triangular if the nuclides are indexed by ascending order by their ZAI index defined by $ZAI = 10000Z + 10A + I$, where Z is the atomic number, A is the mass number and I is the isomeric state [31].

There are two major matrix properties which influence the accuracy of the matrix exponential algorithms described in Chapter 3, these are the matrix norm and the location of the eigenvalues of the matrix. Series approximations such as Padé are most accurate around the origin, meaning that he norm of the matrix must be small. How small the norm must be depends on the order of the Padé approximation. The ℓ_1 norm the for $\boldsymbol{A}t$ is known to be,

$$||\boldsymbol{A}t||_1 = |t|\ ||\boldsymbol{A}||_1 \geq |t| \max|a_{i,j}|, \tag{4.5}$$

meaning that the norm must be greater than or equal to the absolute value of the max matrix element multiplied by the absolute value of time. Of course, as $t \to \infty$ the matrix norm does so as well. If one includes half lives on the order of 10^{-6}s, this would produce a coefficient

on the order of 10^5, resulting in $||\boldsymbol{A}t|| \geq 10^5$. Because burnup calculations are often taken over long time steps and include isotopes which can greatly increase the norm of the matrix, Padé approximations suffer from complications. In order to utilize these approximations, the matrix needs to be scaled a number of times to reduce the norm to a suitable value.

Solutions based on the Cauchy integral formula do not have a requirement on the norm of the matrix, but do on the eigenvalues. In particular the eigenvalues of the transition matrix need to fall in a region enclosed by the contour function. It was noted by Pusa that the eigenvalues of the transition matrix are clustered around the negative real axis [6]. This makes the CRAM algorithm well suited to solve the system in Equation 4.2. There are a number of papers discussing the accuracy of CRAM vs many commonly used matrix exponential methods for depletion calculations [5] [6]. Both of these papers showed that CRAM outperformed the methods that were tested.

Utilizing the CRAM algorithm to solve Equation 4.2 requires the evaluation of $N/2$ independent systems of equations, where N is the CRAM order. In order to accurately and efficiently solve these linear systems, the mathematical characteristics of the burnup matrix must be understood. If no nuclides are excluded from the calculation then size of the matrix would be about $2,000 \times 2,000$, depending on the library. For example, the SCALE code uses libraries based on ENDF/B-VII.1 which include data for over 2,200 nuclides [32]. The number of nuclides makes the system large, however it is sparse with the matrix density only a few percent [31].

The half lives and microscopic cross sections for nuclides can vary significantly causing the magnitude coefficients in the transition matrix to vary from zero to 10^{21} [31]. For example, radioactive decay results in half lives ranging from 10^{-24} seconds to billions of years [1]. Many iterative solvers with have difficulty dealing with the rounding errors introduced by the coefficients. The resulting system will also have eigenvalues with extremely small and large eigenvalues. Iterative solves that rely on Krylov subspace methods become disadvantageous for solving such systems, because of the spectral properties of the matrix [31]. In order to achieve high order of accuracy and stability, direct solvers are chosen over iterative ones. Such solvers include SuperLU or sparse Gaussian elimination with partial pivoting.

4.2 Application to MSR Burnup Calculations

Molten salt reactor depletion calculations differ greatly from traditional nuclear reactors due to the fact that the fuel salt travels throughout the reactor loop. The MSR depletion equation is a special form of the species transport equations that involves nuclear reactions and possibly nonlinear chemical kinetics. These chemical interactions can come from reactions within the salt, reactions with fission products and reactor loop materials, phase transitions and sparging operations. Equation 2.22 is rewritten by moving all terms to the right hand side and by adding a term for the chemical kinetics.

$$\begin{aligned}\frac{\partial \rho_i}{\partial t} = -\nabla \cdot \rho_i(r,t)\boldsymbol{v} - \nabla \cdot j_i(r) + \sum_{j=1}^{N} \frac{M_i}{M_j}\left(b_{j\to i}\lambda_j + \sum_{k=1}^{K}\gamma_{j\to i,k}\sigma_{k,j}(r)\phi(r,t)\right)\rho_j(r,t) \\ - \left(\lambda_i + \phi(r,t)\sum_{k=1}^{K}\sigma_{k,i}(r)\right)\rho_i(r,t) + f_c(\boldsymbol{\rho}, r, T).\end{aligned} \tag{4.6}$$

where $f_c(\boldsymbol{\rho}, r, T)$ is a function that represents the generation of chemical species i. This generation can include linear and/or nonlinear terms and is a function of chemical species vector $\boldsymbol{\rho}$, space r and temperature T.

The implementation of Equation 4.6 is done on a finite volume basis. Each finite volume element is considered a depletion zone, meaning that the neutron flux and microscopic cross sections that were applied to Equation 2.20 are also applied here. For simplicity, we will assume that $f_c = 0$. After applying the volume average species density Equation 4.6 results in,

$$\begin{aligned}\frac{\partial \overline{\rho}_i}{\partial t} = \frac{-1}{V}\int_V \left(\nabla \cdot \rho_i(r,t)\boldsymbol{v} + \nabla \cdot j_i(r)\right)dV + \sum_{j=1}^{N}\frac{M_i}{M_j}\left(b_{j\to i}\lambda_j + \sum_{k=1}^{K}\gamma_{j\to i,k}\overline{\sigma}_{k,j}\overline{\phi}\right)\overline{\rho}_j(t) \\ - \left(\lambda_i + \overline{\phi}\sum_{k=1}^{K}\overline{\sigma}_{k,i}\right)\overline{\rho}_i(t).\end{aligned} \tag{4.7}$$

Each of the transport terms are more complicated and will be treated with the method of lines by discritizing the spacial variables with a 2D finite volume method.

4.2.1 Method of Lines

The method of lines (MOL) is a general technique for solving PDE's which transforms the spacial dependence into an approximate algebraic expression. All but one dimension is discretized, resulting in a system of ODE's. In many physical problems, the undiscretized variable is time. After MOL is applied then normal analytic or numerical integration techniques can be used to solve the system of ODE's. MOL is a generic scheme and can be applied using methods such as finite difference or finite volume [33].

The finite volume method is a discretization technique which divides the initial spacial domain into smaller volumes. In each volume (cell), the dependent variable is averaged and assumed to be constant. Each volume connects to one another and allows the species to transport between cells. Volumetric generation terms from nuclear and chemical reactions will be a function of the averaged species concentration, neutron flux and microscopic cross sections in that cell. Next each of the transport operators discretized using a 2-D finite volume method. Each cell volume is fixed and of equal size. A visual representation of an internal finite volume cell is shown in Figure 4.1

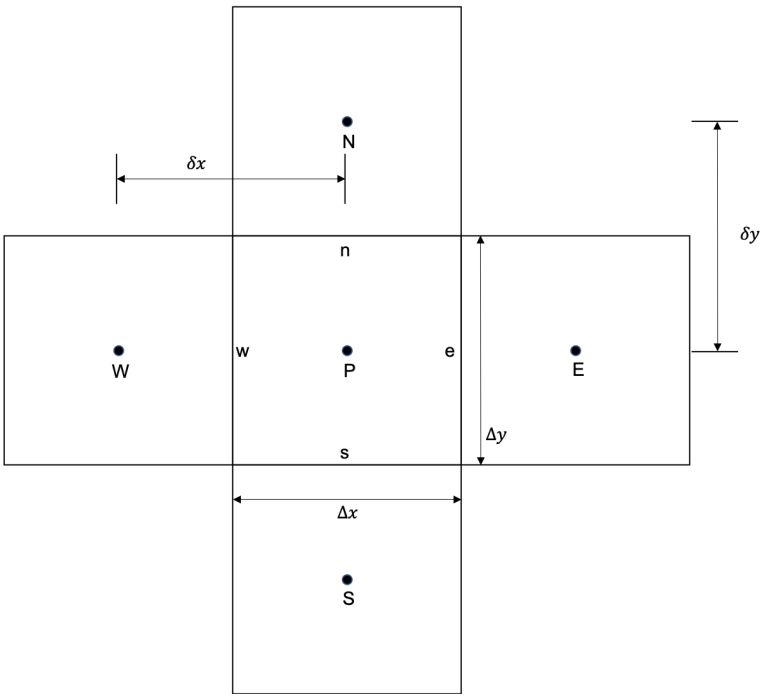

Figure 4.1: 2-D finite volume cell

Diffusion

The diffusive flux is assumed to follow Fick's law of diffusion for an ideal mixture,

$$j_{i,x} = -D_i \frac{d\rho_i}{dx}. \tag{4.8}$$

In a 2-D system, the diffusion terms results in,

$$-\int_V \nabla \cdot j_i dV = -\int_V \frac{\partial}{\partial x}\left(-D_i \frac{\partial \rho_i}{\partial x}\right) dV - \int_V \frac{\partial}{\partial y}\left(-D_i \frac{\partial \rho_i}{\partial y}\right) dV. \tag{4.9}$$

The volume element on a 2-D surface is $dV = dxdy$. To define the diffusive flux into cell P, the diffusion term must be integrated from the West to East wall in the x-direction and from South to North wall in the y-direction [34]. This yields,

$$\begin{aligned} &\int_s^n \int_w^e \frac{\partial}{\partial x}\left(D_i \frac{\partial \rho_i}{\partial x}\right) dxdy + \int_w^e \int_s^n \frac{\partial}{\partial y}\left(D_i \frac{\partial \rho_i}{\partial y}\right) dydx \\ &= D_{e,i}\left(\frac{\partial \rho_i}{\partial x}\right)_e \Delta y - D_{w,i}\left(\frac{\partial \rho_i}{\partial x}\right)_w \Delta y + D_{n,i}\left(\frac{\partial \rho_i}{\partial y}\right)_n \Delta x - D_{s,i}\left(\frac{\partial \rho_i}{\partial y}\right)_s \Delta x. \end{aligned} \tag{4.10}$$

Each of the first order spacial derivatives are assumed to linearly vary between the cell centered points. The diffusion coefficient are defined to be the average between each of pair of points,

$$D_w = \frac{D_W + D_P}{2}, \quad D_e = \frac{D_P + D_E}{2}, \quad D_s = \frac{D_S + D_P}{2}, \quad D_n = \frac{D_P + D_N}{2}. \tag{4.11}$$

The species diffusive flux across each phase are defined as,

$$
\begin{aligned}
D_{e,i}\left(\frac{\partial \rho_i}{\partial x}\right)_e \Delta y &\approx D_{e,i}\left(\frac{\rho_{E,i}-\rho_{P,i}}{\delta x}\right)\Delta y + \mathcal{O}(\Delta x),\\
D_{w,i}\left(\frac{\partial \rho_i}{\partial x}\right)_w \Delta y &\approx D_{w,i}\left(\frac{\rho_{P,i}-\rho_{W,i}}{\delta x}\right)\Delta y + \mathcal{O}(\Delta x),\\
D_{n,i}\left(\frac{\partial \rho_i}{\partial y}\right)_n \Delta x &\approx D_{n,i}\left(\frac{\rho_{N,i}-\rho_{P,i}}{\delta y}\right)\Delta x + \mathcal{O}(\Delta y),\\
D_{s,i}\left(\frac{\partial \rho_i}{\partial y}\right)_s \Delta x &\approx D_{s,i}\left(\frac{\rho_{P,i}-\rho_{S,i}}{\delta y}\right)\Delta x + \mathcal{O}(\Delta y).
\end{aligned}
\tag{4.12}
$$

Plugging these into the diffusion term in the MSR depletion equation gives,

$$
\begin{aligned}
\frac{1}{V}\int_V \nabla \cdot j_i \approx \frac{D_{e,i}}{\Delta x}\left(\frac{\rho_{E,i}-\rho_{P,i}}{\delta x}\right) - \frac{D_{w,i}}{\Delta x}\left(\frac{\rho_{P,i}-\rho_{W,i}}{\delta y}\right)\\
+ \frac{D_{n,i}}{\Delta y}\left(\frac{\rho_{N,i}-\rho_{P,i}}{\delta y}\right) - \frac{D_{s,i}}{\Delta y}\left(\frac{\rho_{P,i}-\rho_{S,i}}{\delta y}\right).
\end{aligned}
\tag{4.13}
$$

Even though each of the derivative approximations for the flux across each surface where first order, the over all order for the diffusion term at point P is second. It can be shown that the approximation to the second order diffusion term used here, is equivalent to the central difference approximation to the second derivative from a finite difference scheme.

Equation 4.13 can be rearranged into a form which represents the coefficients for an interior node of the transition matrix. For species i the coefficients representing diffusion are written as,

$$
\frac{d\rho_P}{dt} = a_E^D \rho_E + a_S^D \rho_S + a_P^D \rho_P + a_W^D \rho_W + a_N^D \rho_N, \tag{4.14}
$$

where,

$$
a_E^D = \frac{D_e}{\Delta x \delta x}, \quad a_S = \frac{D_s}{\Delta y \delta y}, \quad a_W = \frac{D_w}{\Delta x \delta x}, \quad a_N = \frac{D_n}{\Delta y \delta y},
$$

$$
a_P^D = -(a_E^D + a_S^D + a_W^D + a_N^D).
$$

Convection

Convection-diffusion problems are difficult to numerically model because of the relative strengths each of the operators has on the species transport. These relative strengths can be illustrated by the non-dimensional Peclet number [34],

$$Pe = \frac{\text{Convection}}{\text{Diffusion}} = \frac{\rho v}{D/\delta x}. \tag{4.15}$$

As convective forces grow relative to the diffusive, the Peclet number gets larger and larger approaching infinity. On the other hand, if the diffusive forces grow at a rate larger than the convective, the Peclet number approaches zero.

The convective transport term is more complicated and harder to deal with than the diffusion term. Diffusion has no primary direction of flow, it simply cause a species to evenly distribute through a medium through a concentration gradient. Convection on the other hand, has a primary flow direction that is driven by a pressure gradient. One of the major drawbacks of the central differencing scheme is the inability to identify flow direction. In addition to the neglect in identifying the flow direction, the central differencing scheme will cause numerical instability problems for flows with high Peclet number [34]. To combat these numerical problems and to handle flow direction, the upwind differencing scheme can be used.

There are a number of classical differencing schemes such as, first order, power law, and QUICK, each of these will have different orders of accuracy and stability regions. Convection schemes of third order or higher can lead to undershooting or overshooting and applying boundary conditions can be problematic [34]. Because of the issues that higher order schemes can have, much development has been done into deriving a class of second-order schemes called total variation diminishing (TVD) that avoid stability and oscillation issues. TVD schemes have the property of preserving monotonicity, meaning that it must not create local extrema and the value of an existing local minimum must be non-decreasing and that of a local maximum must be non-increasing [34]. One other consequence of monotonicity preserving scheme is that the total variation of the solution should diminish or remain the same with time.

The convection term is discretized and represented using a general second order term. First, flow is defined to be positive from cells West to East and South to North and negative in the opposite directions. For positive and negative flows, the convection operators in the x and y-directions are,

$$\frac{-1}{V}\int_V \frac{\partial}{\partial x}(\rho_i v) \approx \frac{-1}{\Delta x}\big[v_e\rho_{e,i} - v_w\rho_{w,i}\big], \tag{4.16}$$

$$\frac{-1}{V}\int_V \frac{\partial}{\partial y}(\rho_i v) \approx \frac{-1}{\Delta y}\big[v_n\rho_{n,i} - v_s\rho_{s,i}\big]. \tag{4.17}$$

Plugging these in to the convective flux gives,

$$\frac{-1}{V}\int_V \nabla\cdot\rho_i(r,t)\boldsymbol{v} \approx \frac{1}{\Delta x}\big[v_w\rho_{w,i} - v_e\rho_{e,i}\big] + \frac{1}{\Delta y}\big[v_s\rho_{s,i} - v_n\rho_{n,i}\big] \tag{4.18}$$

These equations remain constant no matter the direction of flow, the change in flow direction is taken into account with the generalized species concentration defined at each cell face. The definitions for the concentrations at each cell face are,

East Face

$$\begin{aligned}\rho_e &= \rho_P + \frac{1}{2}\Psi(r_e^+)(\rho_E - \rho_P), \quad v_e, v_w > 0\\ \rho_e &= \rho_E + \frac{1}{2}\Psi(r_e^-)(\rho_P - \rho_E), \quad v_e, v_w < 0\end{aligned} \tag{4.19}$$

West Face

$$\begin{aligned}\rho_w &= \rho_W + \frac{1}{2}\Psi(r_w^+)(\rho_P - \rho_W), \quad v_e, v_w > 0\\ \rho_w &= \rho_P + \frac{1}{2}\Psi(r_w^-)(\rho_W - \rho_P), \quad v_e, v_w < 0\end{aligned} \tag{4.20}$$

North Face

$$\begin{aligned}\rho_n &= \rho_P + \frac{1}{2}\Psi(r_n^+)(\rho_N - \rho_P), \quad v_n, v_s > 0\\ \rho_n &= \rho_N + \frac{1}{2}\Psi(r_n^-)(\rho_P - \rho_N), \quad v_n, v_s < 0\end{aligned} \tag{4.21}$$

South Face

$$\rho_s = \rho_S + \frac{1}{2}\Psi(r_s^+)(\rho_P - \rho_S), \quad v_n, v_s > 0 \tag{4.22}$$
$$\rho_s = \rho_P + \frac{1}{2}\Psi(r_s^-)(\rho_S - \rho_P), \quad v_n, v_s < 0$$

where Ψ is the flux limiter function and r is the ratio of the upwind side gradient and the downwind side gradient [34]. For each cell face, the ratios for positive (r^+) and negative (r^-) flows are defined by,

$$r_e^+ = \left(\frac{\rho_P - \rho_W}{\rho_E - \rho_P}\right), \qquad r_e^- = \left(\frac{\rho_{EE} - \rho_E}{\rho_E - \rho_P}\right), \tag{4.23}$$

$$r_w^+ = \left(\frac{\rho_W - \rho_{WW}}{\rho_P - \rho_W}\right), \qquad r_w^- = \left(\frac{\rho_E - \rho_P}{\rho_P - \rho_W}\right), \tag{4.24}$$

$$r_n^+ \left(\frac{\rho_P - \rho_S}{\rho_N - \rho_P}\right), \qquad r_n^- = \left(\frac{\rho_{NN} - \rho_N}{\rho_N - \rho_P}\right), \tag{4.25}$$

$$r_s^+ = \left(\frac{\rho_S - \rho_{SS}}{\rho_P - \rho_S}\right), \qquad r_s^- = \left(\frac{\rho_N - \rho_P}{\rho_P - \rho_S}\right). \tag{4.26}$$

The criteria for a scheme to be TVD was derived by Sweby [35] using the $\Psi - r$ relationship,

$$\begin{aligned} 0 < r < 1, \quad \Psi(r) \leq 2r \\ r \geq 1, \quad \Psi(r) \leq 2. \end{aligned} \tag{4.27}$$

There are a number of flux limiters which can be applied to Equations 4.19 - 4.22: some of the most popular limiter functions are, Van Leer, Van Albada, Min-Mod, SUPERBEE, QUICK and UMIST. Figure 4.2 shows the grey shaded region for a second order scheme to be TVD, along with common flux limiter functions. In Figure 4.2 many of the limiter

functions overlap. For the region of $r > 1$ QUICK and UMIST are the exact same and for $r > 5$ UMIST, QUICK and SUPERBEE are the same.

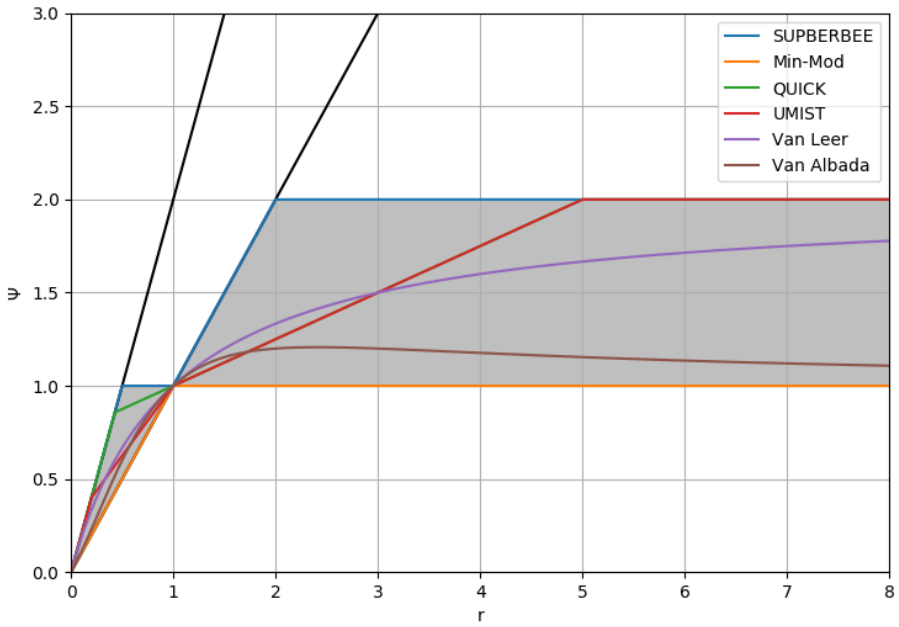

Figure 4.2: Flux limiter functions on the $\Psi - r$ plane

TVD schemes can become unstable due to negative main coefficients and they are often reformulated in a different way to maintain numerical stability. One of the ways in which TVD schemes can be reformulated is by deferred corrections. This is an iterative method which breaks apart the convective flux coefficients into its first order upwind difference component and the second order component. The upwind differencing components will still appear in the usual matrix element however, the second order approximations are moved into a source term. The source term for step n is calculated using species densities from step $n-1$. In this way the source term is differed and the convection flux is corrected. This method can be implemented both explicitly or implicitly. If the implicit formulation is used then a method such as Newton must be used to allow the solution to converge at each time step. The explicit method does not require internal iterations at each step but does require small time steps to increase the accuracy and maintain stability for the second order portion.

The explicit method is utilized in libowski. For smooth functions this formulation is second order, but reverts back to first order for discontinuous functions.

Coefficients for the convection flux can also be rewritten in the same way that was done for the diffusion coefficients. The deferred corrections are placed in a source term coefficient. For an interior node of species i, the transition matrix will have coefficients representing,

$$\frac{d\rho_P}{dt} = a_E^C \rho_E + a_S^C \rho_S + a_P^C \rho_P + a_W^C \rho_W + a_N^C \rho_N + S^C, \tag{4.28}$$

where,

$$S^C = S_E^C + S_S^C + S_W^C + S_N^C$$

$$a_E^C = \max\left(\frac{-v_e}{\Delta x}, 0\right), \quad a_S^C = \max\left(\frac{v_s}{\Delta y}, 0\right),$$
$$a_W^C = \max\left(\frac{v_w}{\Delta x}, 0\right), \quad a_N^C = \max\left(\frac{-v_n}{\Delta y}, 0\right),$$

$$a_P^C = -(a_E^C + a_S^C + a_W^C + a_N^C)$$

$$S_E^C = \frac{v_e}{2\Delta x}\left[(1-\alpha_e)\Psi(r_e^-) - \alpha_e \Psi(r_e^+)\right](\rho_E - \rho_P), \quad \begin{matrix}\alpha_e{=}0, & v_e{<}0 \\ \alpha_e{=}1, & v_e{>}0\end{matrix}$$

$$S_S^C = \frac{v_s}{2\Delta y}\left[(1-\alpha_s)\Psi(r_s^-) - \alpha_s \Psi(r_s^+)\right](\rho_S - \rho_P), \quad \begin{matrix}\alpha_s{=}0, & v_s{<}0 \\ \alpha_s{=}1, & v_s{>}0\end{matrix}$$

$$S_W^C = \frac{v_w}{2\Delta x}\left[(1-\alpha_w)\Psi(r_w^-) - \alpha_w \Psi(r_w^+)\right](\rho_W - \rho_P), \quad \begin{matrix}\alpha_w{=}0, & v_w{<}0 \\ \alpha_w{=}1, & v_w{>}0\end{matrix}$$

$$S_N^C = \frac{v_n}{2\Delta y}\left[(1-\alpha_n)\Psi(r_n^-) - \alpha_n \Psi(r_n^+)\right](\rho_N - \rho_P), \quad \begin{matrix}\alpha_n{=}0, & v_n{<}0 \\ \alpha_n{=}1, & v_n{>}0\end{matrix}$$

4.2.2 Linear Source Terms

Source terms considered thus far are for nuclear reactions. In this transition matrix they are the same as for traditional nuclear burnup equations and the coefficients were shown in Equations 4.3 and 4.4. Diagonal coefficients correspond to the depletion of a species while off diagonal elements are generation terms. For species i is interior cell P the coefficients of the transition matrix for linear sources are,

$$\frac{d\rho_{P,i}}{dt} = a_{P,i}^{LS}\rho_{P,i} + \sum_{j=1,j\neq i}^{N} a_{P,j}^{LS}\rho_{P,j} + \sum S_i^C \tag{4.29}$$

These source terms will also include the deferred corrections from a second order upwind flux approximation represented as constant source terms.

Nuclear reaction rates are calculated in such a way to preserve reaction rates. This is done by integrating over all neutron energies and the volume of the cell. Over a single burn up calculation step all cross sections and the scalar neutron flux is assumed to be constant. Using volume average operators and the multigroup approximation, the neutron flux and microscopic cross section is collapsed into single values for each depletion zone [4]:

$$\sigma_{k,j} = \frac{\int_V \int_0^\infty \sigma_{k,j}(r,E)\phi(r,E)dEdV}{\int_V \int_0^\infty \phi(r,E)dEdV}, \tag{4.30}$$

$$\phi = \frac{1}{V}\int_V \int_0^\infty \phi(r,E)dEdV. \tag{4.31}$$

Source terms for nuclear reactions are not calculated inside of libowski. Thus, for nuclear reactions, an external source must provide the cross sections, neutron flux and decay constants already calculated in the correct manner. These reactions are added to the transition matrix and are expressed as:

$$a_{P,i}^{LS} = \lambda_i + \overline{\phi}\sum_{k=1}^{K}\overline{\sigma}_{k,i}, \quad a_{P,j}^{LS} = \frac{M_i}{M_j}\left(b_{j\to i}\lambda_j + \sum_{k=1}^{K}\gamma_{j\to i,k}\overline{\sigma}_{k,j}(r)\overline{\phi}\right). \tag{4.32}$$

Both of the mass transport models were implemented by added linear coefficients to the transition matrix. If a single species has the ability to transport from the liquid or the wall

or gas phase then that species is duplicated three times. In the cases of Xe, if Xe is allowed to transport to the gas bubbles or the wall then Xe will have three individual species *XeLiq*, *XeGas* and *XeWall*. Xe generated in the liquid from fission would be allowed to transport to the gas bubbles or the wall via terms in the transition matrix which are calculated using the mass transfer models. In addition to the mass transfer coefficient, scalar variables such as temperature, interfacial area, gas void fraction and Henry's Law coefficients are required to compute the transition coefficient.

For species i the following relations represent the mass transfer model for cases when the species exist on the wall or in the liquid:

$$\frac{d\rho_{i,\text{wall}}}{dt} = \frac{kA}{V}(\rho_{i,\text{liquid}} - \rho_{i,\text{wall}}) \qquad \text{Species exist on wall,} \tag{4.33}$$

$$\frac{d\rho_{i,\text{liquid}}}{dt} = \frac{kA}{V}(\rho_{i,\text{wall}} - \rho_{i,\text{liquid}}) \qquad \text{Species exist in liquid,} \tag{4.34}$$

where A is the wall area [m^2], V is the cell volume [m^3] and k is the mass transfer coefficient [m/s]. It is often to express A/V as the interfacial area concentration.The mass transfer coefficient and wall area concentration have to be set by the user when creating the model. In the transition matrix, these coefficients are expressed as:

$$a^{LS}_{P,i,\text{wall}} = -\frac{kA}{V}, \quad a^{LS}_{P,j,\text{liquid}} = \frac{kA}{V}. \tag{4.35}$$

$$a^{LS}_{P,i,\text{liquid}} = -\frac{kA}{V}, \quad a^{LS}_{P,j,\text{wall}} = \frac{kA}{V}. \tag{4.36}$$

In some situations an infinite sink approximation may be appropriate [12]. This assumes that the species build up on the wall will not influence the mass transfer rate to the wall, reducing the model to:

$$\frac{d\rho_{i,\text{wall}}}{dt} = \frac{kA}{V}\rho_{i,\text{liquid}}, \tag{4.37}$$

$$\frac{d\rho_{i,\text{liquid}}}{dt} = -\frac{kA}{V}\rho_{i,\text{liquid}}. \tag{4.38}$$

Gas transport is implemented in a similar way to wall deposition however, the transition coefficient is more complex. Starting with the flux across the bubble interface:

$$j = k_l(\rho_*^l - \rho_{\text{bulk}}^l). \tag{4.39}$$

where ρ_*^l is calculated using the gas phase concentration and ρ_{bulk}^l is the liquid phase concentration. Starting with Henry's law for ideal systems, the concentration of gas species i dissolved in the liquid phase is calculated to be:

$$C_i = H_i P_i, \tag{4.40}$$

where H_i is the Henry's law coefficient [kg/Pa/m^3] and P_i is the partial pressure [Pa] of species i. A gas phase species partial pressure is required to calculate the amount of gas dissolved in the liquid. For an ideal gas, the partial pressure can be related to the gas phase mass density starting with the ideal gas law:

$$P = \frac{nRT}{V}. \tag{4.41}$$

where n is moles of gas [mol], R is ideal gas constant [m^{3}Pa/mole/K], and T is temperature [K]. Dalton's law says that the total in the gas phase is equal to a summation of the partial pressures of each individual gas component:

$$P_{\text{total}} = P_1 + P_2 + P_3 ... = \sum_{i=1}^{N} P_i, \tag{4.42}$$

where partial pressure $P_i = P_{\text{total}} x_i$ and x_i is the mole fraction of species i in the gas phase. Combining Dalton's law and the ideal gas law gives a relation for the partial pressure:

$$P_i = \frac{nRT}{V}\frac{n_i}{n} = \frac{n_i RT}{V}. \tag{4.43}$$

In libowski, species concentrations are solved for mass density in the entire cell volume however, volume in the ideal gas law is for the gas phase. These two volumes are related

using the gas void fraction, $V_{\text{gas}} = V_{\text{cell}}\alpha_{\text{gas}}$. Using the volume relation and the molar mass MM_i [g/mol] of the gas, the partial pressure of gas i in terms of mass density [kg/m^3] is:

$$P_i = \frac{1000RT}{\alpha_{\text{gas}} MM_i}\rho_i^g. \tag{4.44}$$

Plugging this relation into Henry's law gives the equilibrium concentration of gas dissolved in the liquid in terms of mass density in the gas phase:

$$\rho_{*,i}^l = \frac{1000H_iRT}{\alpha_{\text{gas}} MM_i}\rho_i^g \tag{4.45}$$

For species i the following relations represent the mass transfer model for cases when the species exist in the gas bubbles or the liquid:

$$\frac{d\rho_{i,\text{gas}}}{dt} = \frac{kA}{V}\left(\rho_{i,\text{liquid}} - \frac{1000H_iRT}{\alpha_{\text{gas}} MM_i}\rho_{i,\text{gas}}\right) \qquad \text{Species exist in gas,} \tag{4.46}$$

$$\frac{d\rho_{i,\text{liquid}}}{dt} = \frac{kA}{V}\left(\frac{1000H_iRT}{\alpha_{\text{gas}} MM_i}\rho_{i,\text{gas}} - \rho_{i,\text{liquid}}\right) \qquad \text{Species exist in liquid,} \tag{4.47}$$

where k is the mass transfer coefficient [m/s] and A is the interfacial area of the gas phase [m^2]. Again, the interfacial area concentration A/V is set by the user. In the transition matrix, these coefficients are expressed as:

$$a_{P,i,\text{gas}}^{LS} = -\frac{kA}{V}\frac{1000H_iRT}{\alpha_{\text{gas}} MM_i}, \quad a_{P,j,\text{liquid}}^{LS} = \frac{kA}{V}. \tag{4.48}$$

$$a_{P,i,\text{liquid}}^{LS} = -\frac{kA}{V}, \quad a_{P,j,\text{gas}}^{LS} = \frac{kA}{V}\frac{1000H_iRT}{\alpha_{\text{gas}} MM_i}. \tag{4.49}$$

4.2.3 Treatment of Boundary Conditions

Dirichlet and Neumann boundary conditions can be easily implemented by the use of "ghost" cells. Ghost cells are cell which are placed outside the boundary of our domain and will allow for a scalar value or the derivative of a scalar value to be set at the boundary of the domain.

For example, take a 2-D example of the species transport equation. The differential equation for node P would be,

$$\frac{d\rho}{dt} = a_E\rho_E + a_S\rho_S + a_P\rho_P + a_N\rho_N + a_W\rho_W + S \tag{4.50}$$

If P was at the East boundary then a Dirichlet boundary condition means that the value at the cell boundary is fixed ($\rho_e = \rho_{b,r}$). Because there is a ghost cell to the East of cell P, the boundary value can be implemented using a difference across the two cells,

$$\rho_e = \rho_b \approx \frac{\rho_P + \rho_E}{2}. \tag{4.51}$$

The species concentration for the East cell is solved for and plugged back in to the differential equation to yield new coefficients for the matrix elements. This same method is used for the top, bottom, left, right and the corners. Let the subscript b denote a boundary condition value and let r, l, t and b denote the right, left, top and bottom boundaries. Table 4.1 shows the modified coefficients for Dirichlet boundary conditions.

Neumann boundary conditions are applied in a similar way by approximating the derivative at the boundary using a second order central difference approximation at the boundary. For a the bottom boundary this leads to,

$$\rho_b' \approx \frac{\rho_P - \rho_S}{\delta y}. \tag{4.52}$$

The concentration in the south boundary cell is solved for and plugged back into the differential equation. A similar process is done for the top, left, right and corners. The results are shown in Table 4.2 for the modified coefficients. A mixture of both of these boundary conditions can be used for each side, although special care needs to be taken with the corners if the boundary conditions are mixed.

Periodic boundary conditions are applied on either the top and bottom or the left and right. This type of boundary condition can be thought of as folding the rectangular domain creating a cylinder. These are implemented by pointing the cell at one boundary next to the cell at the opposite side. For example, if periodic boundary conditions are applied on the

top and bottom and the flow is positive, then what comes out the top enters at the bottom. In this way, flow loops can be modeled.

Table 4.1: Modified Coefficients for Dirichlet Boundaries

Boundary location	a_E^*	a_S^*	a_P^*	a_N^*	a_W^*	S^*
Bottom	a_E	0	$a_P - a_S$	a_N	a_W	$S + 2a_S\rho_{bb}$
Top	a_E	a_S	$a_P - a_N$	0	a_W	$S + 2a_N\rho_{bt}$
Left	a_E	a_S	$a_P - a_W$	a_N	0	$S + 2a_W\rho_{bl}$
Right	0	a_S	$a_P - a_E$	a_N	a_W	$S + 2a_E\rho_{br}$
Bottom left corner	a_E	0	$a_P - a_S - a_W$	a_N	0	$S + 2a_S\rho_{bb} + 2a_W\rho_{bl}$
Bottom right corner	0	0	$a_P - a_S - a_E$	a_N	a_w	$S + 2a_S\rho_{bb} + 2a_E\rho_{br}$
Top left corner	a_E	a_S	$a_P - a_N - a_W$	0	0	$S + 2a_N\rho_{bt} + 2a_W\rho_{bl}$
Top right corner	0	a_S	$a_P - a_N - a_E$	0	a_W	$S + 2a_N\rho_{bt} + 2a_E\rho_{br}$

Table 4.2: Modified Coefficients for Newmann Boundaries

Boundary location	a_E^*	a_S^*	a_P^*	a_N^*	a_W^*	S^*
Bottom	a_E	0	$a_P + a_S$	a_N	a_W	$S - a_S\rho'_{bb}\delta y$
Top	a_E	a_S	$a_P + a_N$	0	a_W	$S - a_N\rho'_{bt}\delta y$
Left	a_E	a_S	$a_P + a_W$	a_N	0	$S - a_W\rho'_{bl}\delta x$
Right	0	a_S	$a_P + a_E$	a_N	a_W	$S - a_E\rho'_{br}\delta x$
Bottom left corner	a_E	0	$a_P + a_S + a_W$	a_N	0	$S - a_S\rho'_{bb}\delta y - a_W\rho'_{bl}\delta x$
Bottom right corner	0	0	$a_P + a_S + a_E$	a_N	a_w	$S - a_S\rho'_{bb}\delta y - a_E\rho'_{br}\delta x$
Top left corner	a_E	a_S	$a_P + a_N + a_W$	0	0	$S - a_N\rho'_{bt}\delta y - a_W\rho'_{bl}\delta x$
Top right corner	0	a_S	$a_P + a_N + a_E$	0	a_W	$S - a_N\rho'_{bt}\delta y - a_E\rho'_{br}\delta x$

4.2.4 Application of Matrix Exponential Solvers

Five different matrix exponential solver are implemented in libowski with the added ability to use the Krylov subspace approximation. Each of these methods were discussed in Chapter 3 and algorithms are presented in this section. These methods include three based on Cauchys integral formula and two based on Padé's approximation with scaling and squaring. Because of the various magnitudes of the transition matrix coefficients, a direct solver is used to solve the linear systems [31]. The direct solver is Eigens sparse LU decomposition which is based on the SuperLU library [36] [37]. Each of the algorithms listed in this paper are shown in Appendix A.

CRAM

The biggest problem with using CRAM is obtaining the coefficients for the partial fraction decomposition. These coefficients for order 14 and 16 can be found in Reference [25], only order 16 is implemented in libowski. As discussed earlier for Cauchy method of order N, $N/2$ number of complex linear systems need to be solved. The general algorithm for a method based on contour integrals in partial fraction decomposition form is shown in Algorithm 5. Both the Parabolic and Hyperbolic methods used this same algorithm but with different poles and residues. It should be noted that notation for the residues where changed from c_k to α_k and the residues from z_k to θ_k. For complex matrices and scalars α and θ would need to be the same size as the approximation order. Real valued matrices and scalars require half of these coefficients. Because all the matrices here are real, α and θ are all of size $N/2$.

Parabolic and Hyperbolic

Solutions with the parabolic and hyperbolic contours use Algorithm 5 but with different poles and residues. While methods for evaluating the coefficients for CRAM take a long time and need to be precalculated, the poles and residues for contour functions can be computed when the solver is initialized. Calculating the coefficients requires the contour function (ϕ), the function derivative (ϕ') and the order of the approximation (N) Algorithms 3 and 4 in Appendix A show the functions computing the arrays for α and θ with parabolic and hyperbolic contours. While any order is possible, the default order in libowski is 32.

Parallelization of Cauchy Solvers

As mentioned before, for a given order N of the Cauchy solvers $N/2$ linear systems need to be solved. Each of these systems are independent and can be performed separately. Paralelization takes two general forms: shared and distributed memory. Shared memory archetypes involve using multicore CPUs which have access to the same shared memory. Because of this; all processors have access to the same set of variables and can modify them individually. Distributed memory works by connecting multiple CPUs over a network and having each computer solve a portion of the problem. In this case; each computer has its

own copy of the variables required to solve its portion of the problem. Libraries exist which allow programmers the ability to parallelize their code on a high level for each of these two architectures. Open multiprocessing (OpenMP) is used for shared memory and message massing interface (MPI) for distributed memory [38] [39]. In general OpenMP is easier to implement but less flexible because of the memory requirement. This issue with MPI is that using it on a shared memory device will unnecessarily increase the memory consumption. Because of its inherent flexibility MPI is utilized in parallizing the Cauchy solver in libowski.

Padé - Method 1

The first method bast on the Padé approximation was developed by Nicholas J. Higham and can be found in Reference [23]. Higham developed this algorithm in a similar manner which was done before by Moler and Van Loan [21] by choosing the optimal number of matrix scalings required for the Padé of a certain order. Higham noted that the backward error analysis done by Moler and Van Loan was simple and elegant, however not sharp. Instead of only developing an algorithm based on a numerical error bound, Highman also considered computational cost. The resulting algorithm developed by Highman can be found in Reference [23] and the reader should refer to Highman's paper for further detail in its development. Algorithm 14 is the algorithm that was developed in the paper, and shows its implementation in libowski. From now it shall be referred to as Padé Method 1. Padé Method's 1 and 2 both rely on functions that are shown in Appendix A. Unlike methods based on contour functions, the Paé method directly computes the matrix exponential.

Padé - Method 2

The second Padé method is a modification of the first Padé method that addresses the weakness in overscaling. Overscaling occurs when a large matrix norm causes a larger than necessary α to be used, leading to decreased accuracy [24]. Al-Mohy and Higham developed this algorithm by inroducing a new sharper truncation error bound, which is likely to help correct the overscaling problem. More information on the new algorithm can be found in Referenence [24]. Algorithm 6.1 from said reference is implemented in libowski and is shown

in Algorithm 15. There are two helper functions that are used in Algorithm 15, these function are defined Appendix A along with Algorithm 15.

Taylor

The Taylor algorithms implemented in libowski was taken from Reference [22] for computing $F \approx e^{\boldsymbol{A}t}\boldsymbol{B}$ over a single step, where $\boldsymbol{A} \in \mathbb{C}^{n\times n}$ and $\boldsymbol{B} \in \mathbb{C}^{n\times n_0}$. Unlike the Padé methods, the Taylor algorithm does not require the need for linear solves. This was done in an attempt to reduce the computational cost. Three key ideas were employed when developing this algorithm:

1. Careful choice of the Taylor series order and scaling parameter, exploiting estimates of $||t\boldsymbol{A}^p||^p$, to keep the backward error suitably bounded while minimizing the computational cost.

2. Shifting, and optional balancing, to reduce the norm of $\boldsymbol{A}t$.

3. Premature termination of the truncated Taylor series evaluations.

One of the major components of this algorithm is in picking the optimal Taylor order and scaling parameter. This is accomplished by minimizing the computational cost while choosing the appropriate scaling and squaring parameter to obtain a backwards error below a minimal tolerance. For this algorithm, the parameters function accomplishes this and is shown in Algorithm 16.

4.2.5 Time Marching Schemes

There are two different approaches to evaluating the solution as a function of time. Each of these schemes will have different impacts on the error and computation time. The first scheme involves evaluating the solution at each time step starting from the initial condition. In this case the step length is not constant and the length is equal to the time at the time step assuming that the solution begins from $t_0 = 0$. The solution for this time marching scheme is,

$$\boldsymbol{\rho}(t_{n+1}) = e^{\boldsymbol{A}\Delta t}\boldsymbol{\rho}(t_0), \quad \Delta t = t_{n+1} - 0, \quad \text{Time stepping method 1.} \tag{4.53}$$

One of the advantages of this time marching scheme is that the error is based on the error for a single time step because the initial condition is considered exact. As Δt becomes larger then the norm of $\boldsymbol{A}\Delta t$ also grows, this can become a problem for methods based on Padé. One other consequence is that the eigenvalues of the transition matrix will grow and possibly spread apart, causing problems with some of the numerical methods previously discussed. If the eigenvalue has a large imaginary part and a small real part then any $\Delta t > 1$ could move then eigenvalue into a region where Cauchy's solution will be inaccurate. Krylov subspace methods will also not work well if the eigenvalues are spread apart.

The second method involves calculating the solution at t_{n+1} is calculated using the $\rho(t_n)$ as the initial condition where $\rho(t_n)$ was calculated from the previous time step and thus not the exact solution. The solution for this time marching scheme is,

$$\boldsymbol{\rho}(t_{n+1}) = e^{\boldsymbol{A}\Delta t}\boldsymbol{\rho}(t_n), \quad \Delta t = t_{n+1} - t_n, \quad \text{Time stepping method 2} \tag{4.54}$$

The error with this calculation will be larger because of the compounding errors from taking multiple time steps instead of one. Solutions using matrix exponential methods do not behave in the same way that normal numerical integration methods i.e. its not guaranteed that smaller time steps give more accurate solutions. There are however, a few of advantages to this scheme. The first is that it has been shown that the accuracy of CRAM is increased by dividing long depletion steps into smaller substeps if the species concentration diminishes significantly during a single time step [28]. If methods based on Padé are used, then dividing the total depletion length into smaller time steps will help with the size of the matrix norm. Another consequence of the Padé method can be exploited if time steps of constant size are taken. Because Padé methods compute the matrix exponential and not the action on a vector, the matrix exponential will not change for constant time steps. This is not true in the case where other elements such as flow or reactions rates change moving from each time step.

Chapter 5

Results

In chapters 2, 3, 4 the theoretical framework and application for solving mass transport problems in MSRs were developed on a finite volume mesh. Components of the solution strategy are broken in to three general categories:

1. Diffusion operator
2. Convection operator
3. Time integration

The first two categories deal with the spatial discretization of the PDE. The third involves the time variance in the solution as its ability to handle volumetric source terms. A number of test are conducted to access the numerical solution of libowski with these three key features in mind. These test aim to explore the accuracy of the libowski and the computation time required to solve such problems. A set of progression problems are defined which explore, diffusion, convection, linear source terms and the combination of each. After this small case studies are performed with a nuclear reactor design in mind which looks at libowskis' ability to handle the types of problems which arrive in the field. These problems include the set of neutron precursors as well as depletion and mass transport with a small selection of radio nuclides.

In order to test the validity of libowski, the problems which are conducted must have reference solutions to test against. Many of the problems do in fact have analytical solutions

while some do not. For those that do not MATLAB is utilized in generating the reference solution. In these cases the transition matrix and initial condition that is generated in libowski is save to a CSV file and read by a MATLAB script. This script then uses the Symbolic tool box to calculate the exponential of the transition matrix using variable precision arithmetic with 64 digits of accuracy. The solution is then converted to IEEE double precision and read in to libowski as the reference solution. While this method is not able to test the accuracy of the diffusion or convection operators, it is able to access the time integration thus overall accuracy of the matrix exponential algorithms.

Many of the case studies involve a set of isotopes that are referred to as the small and medium sets. These set of isotopes were chosen based on their inclusion in TRITON for cross section evaluation with depletion [32]. The small case corresponds to *addnux* = -2 and the medium case to *addnux* = 2. Additional isotopes are added to both of these sets to fully include the initial condition for the MSRE salt [40]. The list of isotopes for the small and medium cases are shown in Appendix , Tables . One important note about these tables is that indicate which species will have addition mass transport models for wall depositon and gas sparging. In some cases no mass transport is included, resulting in a subset of nuclides which do not include the additional isotopes required to model addition mass transport models. For example if one were to model three species, ^{235}U, ^{135}Xe and ^{109}Ag with no mass transport then the system would only contain these three isotopes. If wall deposition and gas sparging is added for Ag and Xe then the system would contain two addition species ^{135}XeGas and ^{109}AgWall. Making the total number of species five.

All problems show an error based on a reference solution. For the following results the relative errors are defined as:

$$E_\infty = \max\left(|\hat{u}_i - u_i|\right) \quad \text{For } i = 1, 2, ..., N,$$

$$E_1 = \frac{1}{N}\sum_{i=1}^{N}|\hat{u}_i - u_i|, \qquad E_2 = \frac{1}{N}\sqrt{\sum_{i=1}^{N}\left(|\hat{u}_i - u_i|\right)^2},$$

where N is the number of elements in the solution domain. Some times it is more meaningful to shown an absolute error instead of a relative error. It is explicitly stated in the results weather a relative or absolute difference is used. Run time is also reported for some test and is reported as the wall time for calling the *solve* function. This includes the time to build the matrix, run the solution algorithm and unpack the solution. For problems where multiple time steps are taken, the matrix is rebuilt before each time step to update the deferred correction source term. While these run times are reported with no standard deviation, some changes are to be expected by running the problems multiple times or on different machines.

Earlier, it was discussed that substepping can increase the accuracy of Cauchy based solvers. For all results shown, unless other wise noted, no substeps are used for either the CRAM, Parabolic or Hyperbolic solvers. For some of the reaction-diffusion-convection problems, substepping does not play a role in increasing the accuracy of the solution but will increase the run time. Exceptions to this will be further discussed in the results. The default orders for the CRAM, Hyperbolic and Parabolic solvers in libowski are 16, 32 and 32 respectively.

5.1 Progression Problems

5.1.1 Problem 1

The first diffusion problem consist of a 2-D system shown as:

$$\frac{\partial U}{\partial t} = k\frac{\partial^2 U}{\partial x^2} + k\frac{\partial^2 U}{\partial y^2}, \tag{5.1}$$

on the domain $x \in [0,1]$, $y \in [0,1]$, subject to periodic boundary conditions and initial condition,

$$U(x,y,0) = \sin(2\pi x)\sin(2\pi y), \tag{5.2}$$

and solution,

$$U(x, y, t) = e^{-t} \sin(2\pi x) \sin(2\pi y), \tag{5.3}$$

with $k = 1/(8\pi^2)$. The Problem is ran for a total time of 2.0 seconds with the number of cells in the x and y direction being the same [10, 20, 40, 80]. These results are shown in Table 5.1. These results show good convergence rates for the l_∞ and l_1 error functions previous described, but not l_2. While each of the solvers maintained the same error, the run times were drastically different. Run times for each of the solvers is shown in Figure (5.1).

From Figure (5.1) each of the solvers show a monomial relation between the problem size and run time. Both the Parabolic and Hyperbolic solvers have almost the exact same solve time. This is because they have to solve the same number of linear systems. The CRAM solver requires half the number of linear solves, making it about twice as fast. For this example the Taylor solver is the fastest but only slightly beats each of the Cauchy solvers. The Padé solvers scale much more poorly than the other ones. Starting out each of the solvers have a similar run time but as the problem size increases, the Padé methods run times grow at a faster rate. Each of these solvers maintain the same numerical error do the the error being dominated by the spatial discretization error of the diffusion operator.

The eigenvalues for this problem were found to be clustered on the negative real axis with zero imaginary parts. For the 400 x 400 case, the eigenvalues are shown in Figure 5.2. On exception to this is a positive eigenvalue which is very close to the origin at 6.0396e-14. This positive value only appears in the 400 x 400 case. The other discretizations show the same behavior with the eigenvalues being clustered on the negative real axis but do not show the positive valued eigenvalue a the origin. Due to the relatively long solve times, particularly for the Padé solvers, the Krylov subspace approximation was used to analyze the error and run time. For a spatial resolution of 160 cells in both the x and y direction, results for various subspace dimensions M are shown in Table 5.2. Interestingly, the error associated with reducing the overall dimension of the problem did not change, even though the run time drastically decreased leading to the conclusion that the dimension of the true space is much smaller.

Table 5.1: Convergence Rate for Diffusion Problem 1 with Absolute Error

Solver	Cells	E_∞ Rate	E_1 Rate	E_2 Rate	E_∞ Error	E_1 Error	E_2 Error	Solve Time (sec)
CRAM	100	-	-	-	4.39e-03	1.84e-03	2.20e-04	8.06e-03
-	400	2.02	2.02	2.98	1.08e-03	4.53e-04	2.77e-05	5.07e-02
-	1600	1.97	2.01	3.00	2.76e-04	1.13e-04	3.48e-06	5.37e-01
-	6400	1.99	2.00	3.00	6.94e-05	2.82e-05	4.35e-07	5.62e+00
Parabolic	100	-	-	-	4.39e-03	1.84e-03	2.20e-04	1.08e-02
-	400	2.02	2.02	2.98	1.08e-03	4.53e-04	2.77e-05	9.71e-02
-	1600	1.97	2.01	3.00	2.76e-04	1.13e-04	3.48e-06	1.06e+00
-	6400	1.99	2.00	3.00	6.94e-05	2.82e-05	4.35e-07	1.10e+01
Hyperbolic	100	-	-	-	4.39e-03	1.84e-03	2.20e-04	1.07e-02
-	400	2.02	2.02	2.98	1.08e-03	4.53e-04	2.77e-05	9.70e-02
-	1600	1.97	2.01	3.00	2.76e-04	1.13e-04	3.48e-06	1.05e+00
-	6400	1.99	2.00	3.00	6.94e-05	2.82e-05	4.35e-07	1.10e+01
Pade-method1	100	-	-	-	4.39e-03	1.84e-03	2.20e-04	6.19e-03
-	400	2.02	2.02	2.98	1.08e-03	4.53e-04	2.77e-05	5.27e-01
-	1600	1.97	2.01	3.00	2.76e-04	1.13e-04	3.48e-06	6.70e+01
-	6400	1.99	2.00	3.00	6.94e-05	2.82e-05	4.35e-07	5.52e+03
Pade-method2	100	-	-	-	4.39e-03	1.84e-03	2.20e-04	1.62e-02
-	400	2.02	2.02	2.98	1.08e-03	4.53e-04	2.77e-05	1.23e+00
-	1600	1.97	2.01	3.00	2.76e-04	1.13e-04	3.48e-06	1.19e+02
-	6400	1.99	2.00	3.00	6.94e-05	2.82e-05	4.35e-07	9.48e+03
Taylor	100	-	-	-	4.39e-03	1.84e-03	2.20e-04	4.83e-03
-	400	2.02	2.02	2.98	1.08e-03	4.53e-04	2.77e-05	2.20e-02
-	1600	1.97	2.01	3.00	2.76e-04	1.13e-04	3.48e-06	1.20e-01
-	6400	1.99	2.00	3.00	6.94e-05	2.82e-05	4.35e-07	7.38e-01

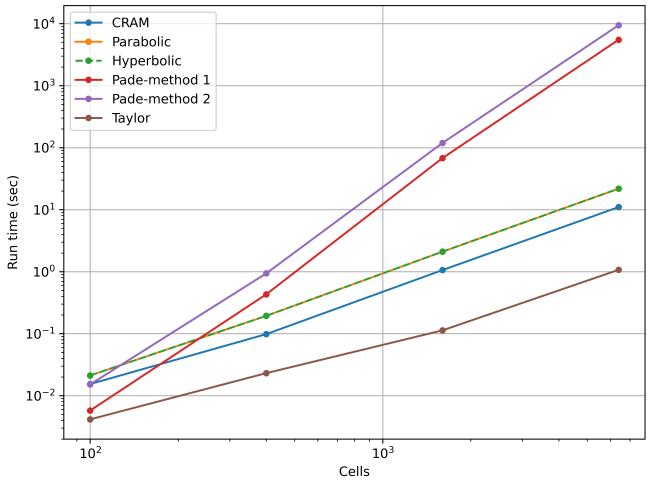

Figure 5.1: Run time performance for diffusion problem one

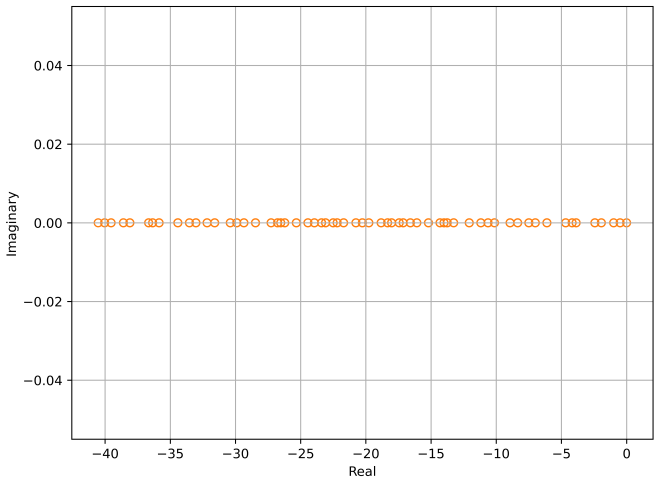

Figure 5.2: Eigenvalues for the 400 x 400 case in diffusion problem one

Table 5.2: Error and Run Times for Different Krylov Subspace Dimensions

Solver	M	l_∞ Error	l_1 Error	l_2 Error	Run time (sec)
Padé-method 1	5	1.74e-05	7.05e-06	1.12e-02	1.13e-02
-	10	1.74e-05	7.05e-06	1.12e-02	1.47e-02
-	25	1.74e-05	7.05e-06	1.12e-02	3.31e-02
-	50	1.74e-05	7.05e-06	1.12e-02	9.27e-02
-	100	1.74e-05	7.05e-06	1.12e-02	3.33e-01
-	150	1.74e-05	7.05e-06	1.12e-02	7.30e-01
-	200	1.74e-05	7.05e-06	1.12e-02	1.33e+00
Padé-method 2	5	1.74e-05	7.05e-06	1.12e-02	9.15e-03
-	10	1.74e-05	7.05e-06	1.12e-02	1.18e-02
-	25	1.74e-05	7.05e-06	1.12e-02	2.96e-02
-	50	1.74e-05	7.05e-06	1.12e-02	8.95e-02
-	100	1.74e-05	7.05e-06	1.12e-02	3.28e-01
-	150	1.74e-05	7.05e-06	1.12e-02	8.20e-01
-	200	1.74e-05	7.05e-06	1.12e-02	1.55e+00

5.1.2 Problem 2

In the second example, a 1D reaction-diffusion problem was modeled. This problem was taken from work performed by Chou et al. ([41]). The partial differential equations (PDEs) are as follows:

$$\begin{aligned} \frac{\partial U}{\partial t} &= d\frac{\partial^2 U}{\partial x^2} - aU + V, \\ \frac{\partial V}{\partial t} &= d\frac{\partial^2 V}{\partial x^2} - bV, \end{aligned} \tag{5.4}$$

on the domain $x \in [0, \pi/2]$. The system is subject to the following boundary conditions:

$$\frac{\partial U}{\partial x}(0,t) = 0, \quad \frac{\partial V}{\partial x}(0,t) = 0, \quad U(\frac{\pi}{2},t) = 0, \quad V(\frac{\pi}{2},t) = 0, \tag{5.5}$$

with the following initial condition:

$$U(x,0) = 2\cos(x), \quad V(x,0) = (a-b)\cos(x). \tag{5.6}$$

The exact solution is given by

$$\begin{aligned} U(x,t) &= \left(e^{-(a+d)t} + e^{-(b+d)t}\right)\cos(x), \text{ and} \\ V(x,t) &= (a-b)e^{-(b+d)t}\cos(x). \end{aligned} \tag{5.7}$$

The same three test problems that were conducted in the work by Chou et al. ([41]) were also conducted in libowski. These test results correspond to changes in coefficients $[a, b, d]$, which produced a diffusion-dominated system [0.1, 0.01, 1.0], a reaction-dominated system [2.0, 1.0, 0.001], and a stiff reaction system [100, 1.0, 0.001]. Each test case was run with one time step to $t = 1$, so $\Delta t = 1$. The number of cells in the x direction was varied from 10 to 320. Results for the spatial convergence for the Taylor solver for the diffusion-dominated, reaction-dominated and stiff reaction–dominated cases are shown in Tables 5.3, 5.4, and 5.5.

Each solver produced the same error at up to four decimal places for most of the n values; therefore, only the Taylor results are shown. Full results for these test are shown in Appendix C Tables 3, 4 and 5. As expected, the convergence rate for this problem is second order or better for each of the error metrics. While there is little difference between the errors from these solvers for this first problem, there is a notable difference in run times for the Padé and Taylor solvers. The E_∞ error and run time for each sub-problem is shown in Figures 5.3, 5.4 and 5.5.

Figures 5.3-5.5 show that CRAM is the fastest solver, followed by the parabolic and hyperbolic solvers. Both of the Padé solvers had drastically much longer run times for the diffusion-dominated cases, with Method 2 being the longest. Interestingly, the Taylor solver showed dramatically different run times based on the physical process dominating the problem. In the diffusion-dominated cases, the Taylor solver took the longest, but in the reaction-dominated cases, the run time was on the order of the CRAM, parabolic, and hyperbolic solvers.

The Krylov subspace approximation was applied to each sub-problem with 1,000 cells in the x direction. The Krylov dimensions were varied from 2 to 100, and the results are shown in Figure 5.6 . Unlike the previous diffusion example, a much larger Krylov dimension was required to reach a converged error in the diffusion-dominated case. Each solver showed the same error behavior, but with slightly different converged errors due to numerical accuracy in the algorithms. Each solver also showed similar behavior, with run time scaling as a function of the Krylov dimension; however, each axis was scaled differently. Applying the Krylov subspace to each Padé solver reduced the many orders of magnitude, but it failed to reduce the run time of the Taylor solver to the same order in the diffusion-dominated case. For each of the reaction-dominated cases, the Krylov subspace dimension required to reach a converged solution is much smaller than the diffuion-dominated case.

Table 5.3: Convergence Rate for Diffusion-Dominated Problem Using Absolute Error

n	E_∞ Rate	E_1 Rate	E_2 Rate	E_∞ Error	E_1 Error	E_2 Error
20	2.00	2.00	2.50	1.87e-04	1.19e-04	2.97e-05
40	2.00	2.00	2.50	4.69e-05	2.99e-05	5.24e-06
80	2.00	2.00	2.50	1.17e-05	7.46e-06	9.27e-07
160	2.00	2.00	2.50	2.93e-06	1.87e-06	1.64e-07
320	2.00	2.00	2.50	7.33e-07	4.66e-07	2.90e-08

Table 5.4: Convergence Rate for Reaction-Dominated Problem Using Absolute Error

n	E_∞ Rate	E_1 Rate	E_2 Rate	E_∞ Error	E_1 Error	E_2 Error
20	2.00	2.00	2.50	2.23e-04	1.42e-04	3.53e-05
40	2.00	2.00	2.50	5.58e-05	3.55e-05	6.24e-06
80	2.00	2.00	2.50	1.40e-05	8.88e-06	1.10e-06
160	2.00	2.00	2.50	3.49e-06	2.22e-06	1.95e-07
320	2.00	2.00	2.50	8.72e-07	5.55e-07	3.45e-08

Table 5.5: Convergence Rate for Stiff Reaction–Dominated Problem Using Absolute Error

n	E_∞ Rate	E_1 Rate	E_2 Rate	E_∞ Error	E_1 Error	E_2 Error
20	2.00	2.00	2.50	9.42e-03	6.00e-03	1.49e-03
40	2.00	2.00	2.50	2.36e-03	1.50e-03	2.63e-04
80	2.00	2.00	2.50	5.89e-04	3.75e-04	4.66e-05
160	2.00	2.00	2.50	1.47e-04	9.38e-05	8.23e-06
320	2.00	2.00	2.50	3.68e-05	2.34e-05	1.46e-06

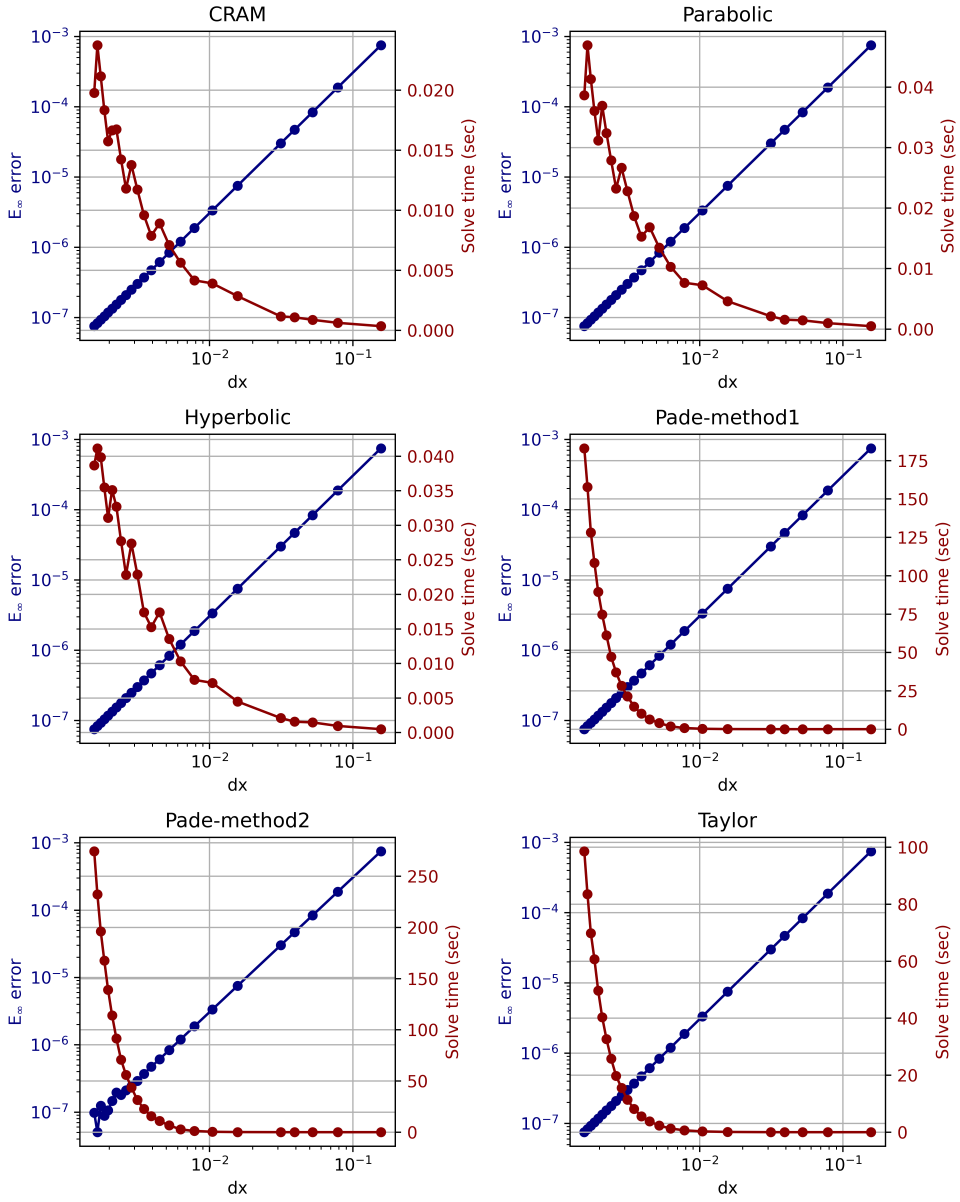

Figure 5.3: Results for the diffusion-dominated case

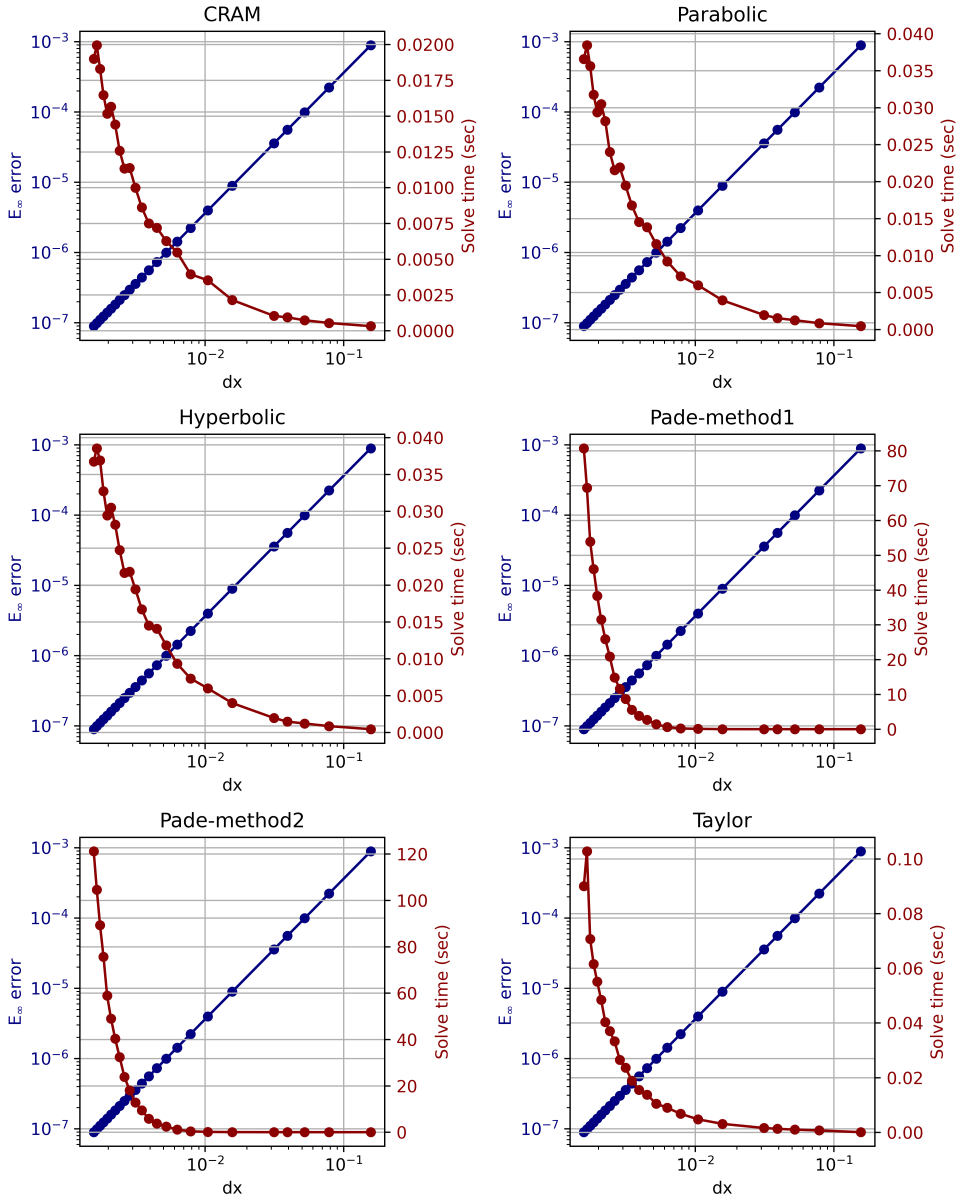

Figure 5.4: Results for the reaction-dominated case

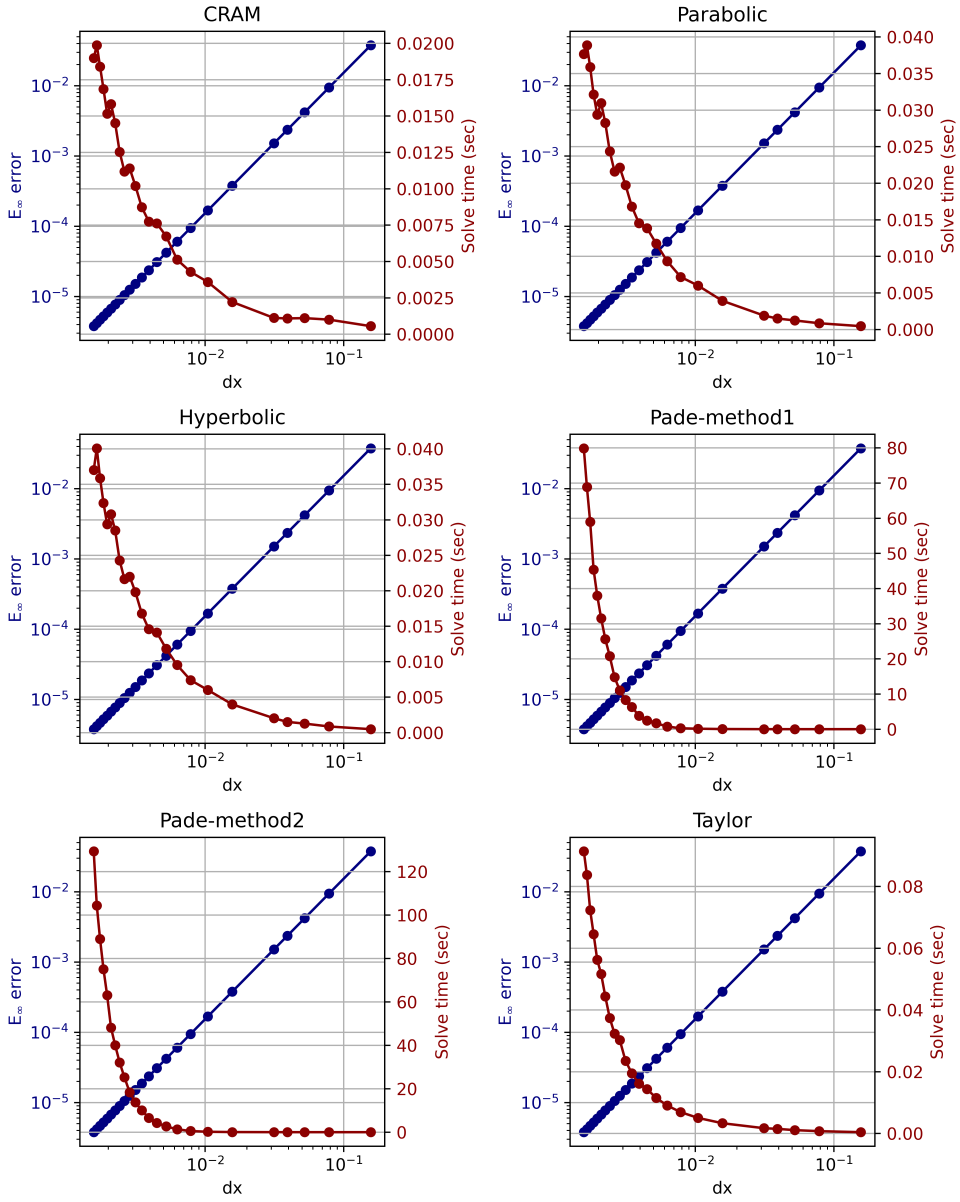

Figure 5.5: Results for the stiff reaction-dominated case

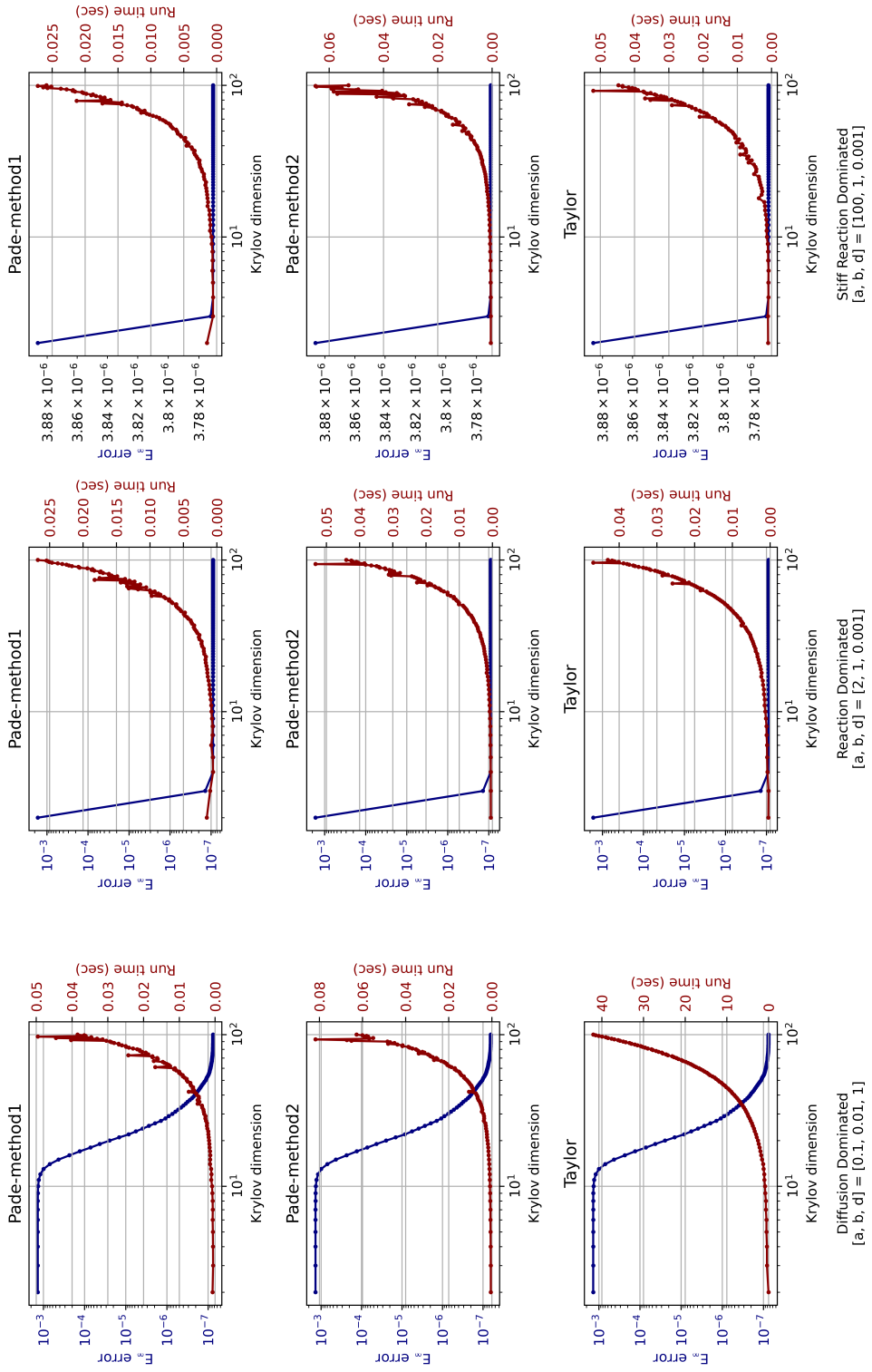

Figure 5.6: Results for the Krylov subspace approximation for each sub-problem

5.1.3 Problem 3

Convection was tested using PDE of the form

$$\frac{\partial U}{\partial t} = -v\frac{\partial U}{\partial x}, \tag{5.8}$$

where v is velocity. Equation (5.11) was applied to a system on the domain $x \in [0, 100]$, $t \in [0, 20]$ subject to the following boundary conditions:

$$U(0, t) = 1.0, \quad \frac{\partial U}{\partial x}(100, t) = 0.0, \tag{5.9}$$

and the initial condition:

$$U(x, 0) = 0.0. \tag{5.10}$$

What is most important about this problem is that it exposes an instability with Cauchy solvers which seems to be specific to this problem. This is something that is not seen with the series solvers such as Padé or Taylor. First, results for the first order upwind differencing scheme are shown in Figure 5.7 for a first-order backward differencing formula (BDF1) and the ETD scheme using the series matrix exponential solvers. While only the Taylor solver is shown, results for the Padé solvers are the same. Figure 5.7 shows the typical reduction in numerical diffusion with decreasing dx. In figure 5.7 we see that as the time step size decreases for the explicit BDF1 solver it approaches the accuracy of the ETD method with a single time step.

The instability seen with Cauchy solvers is shown in figure 5.8 for a zoomed in and zoomed out view of each solution. In figure 5.8 it is seen that for dx values 0.5 and 0.4, the solution blows up and oscillates. What is important to note that even know the solution for dx = 1.0 is stable it does contain small negative values as it approaches zero. The matrix exponential is the analytical solution to the ODE system and should not show any instability, so the instability shown here is due to the numerical computation of the exponential itself. To combat this instability the order of the method can be increased, this is seen in figure 5.9. Figure 5.9 shows the Hyperbolic and Parabolic solvers at different order of N. As the order in

increased, the solution becomes more stable. Another method to increase the stability of the solution is to compute the solution with multiple steps and not just at the final time step. This is equivalent to using the substep method for the Cauchy solvers. Figure 5.10 shows results for each of the Cauchy solver for different time step sizes. Even with large time step sizes the solution becomes stable. Discuss accuracy related to the Jordan canonical form.

To illustrate the enhanced accuracy of TVD non-linear flux limiters, the same problem is conducted with a variety of limiter functions. This method works to approximate the convection term to at most second order in the case of smooth solutions and reduce numerical diffusion seen in the first order case. Figure 5.11 shows results from each of the flux limiter functions in libowski using the Taylor solver. Each of the flux limiter functions reduces the numerical diffusion, seen in the first order upwind case with superbee being the best. While the TVD scheme is designed to remove unphysical oscillations seen in higher order convection schemes, these oscillations are present because of the explicit implementation. Figure 5.12 shows these oscillations for six flux limiter functions. This behavior is only present when large time step sizes are used. For some time integration methods, explicit schemes are known to be unstable if large time step sizes are used. In this case the second order correction being calculated from the previous time step causes the solution to become unstable if large enough time steps are taken. This same behavior is seen in all the ETD matrix exponential solvers.

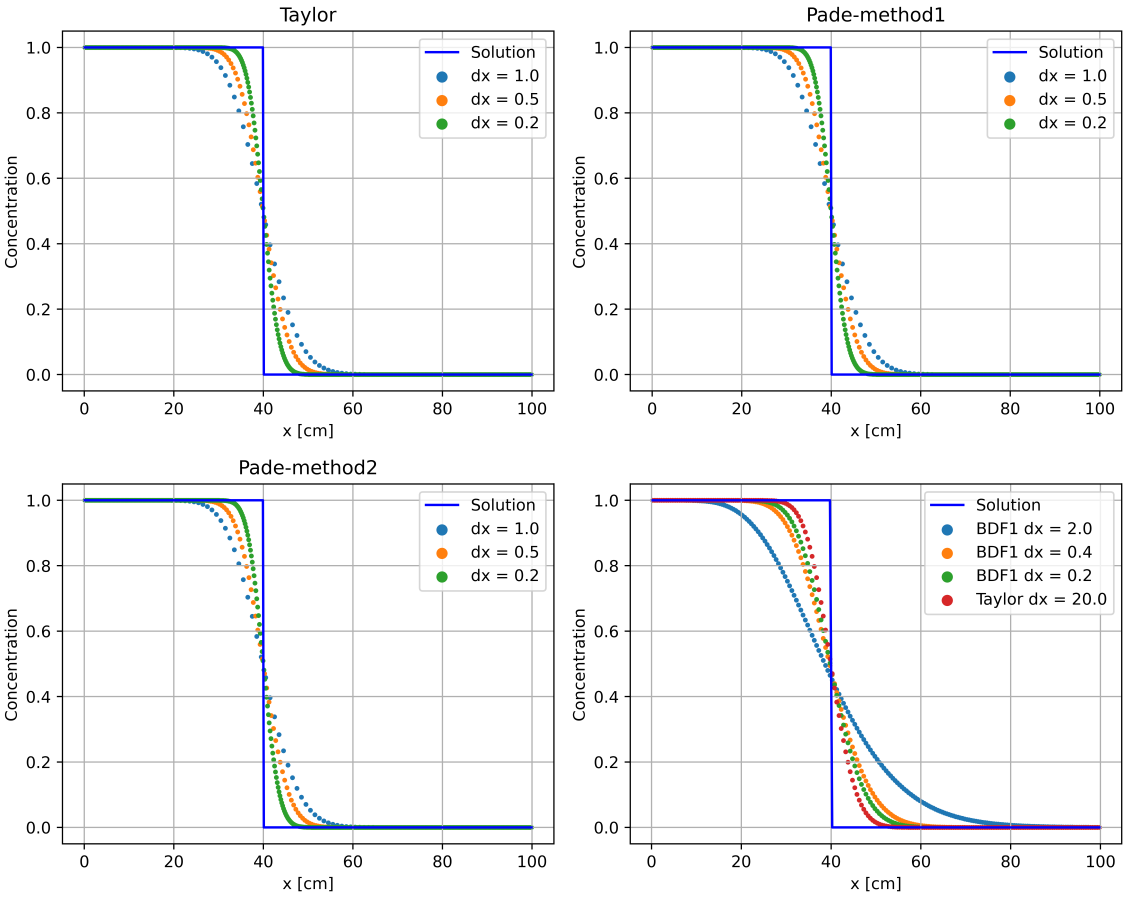

Figure 5.7: First-order upwind differencing with series solvers and BDF1

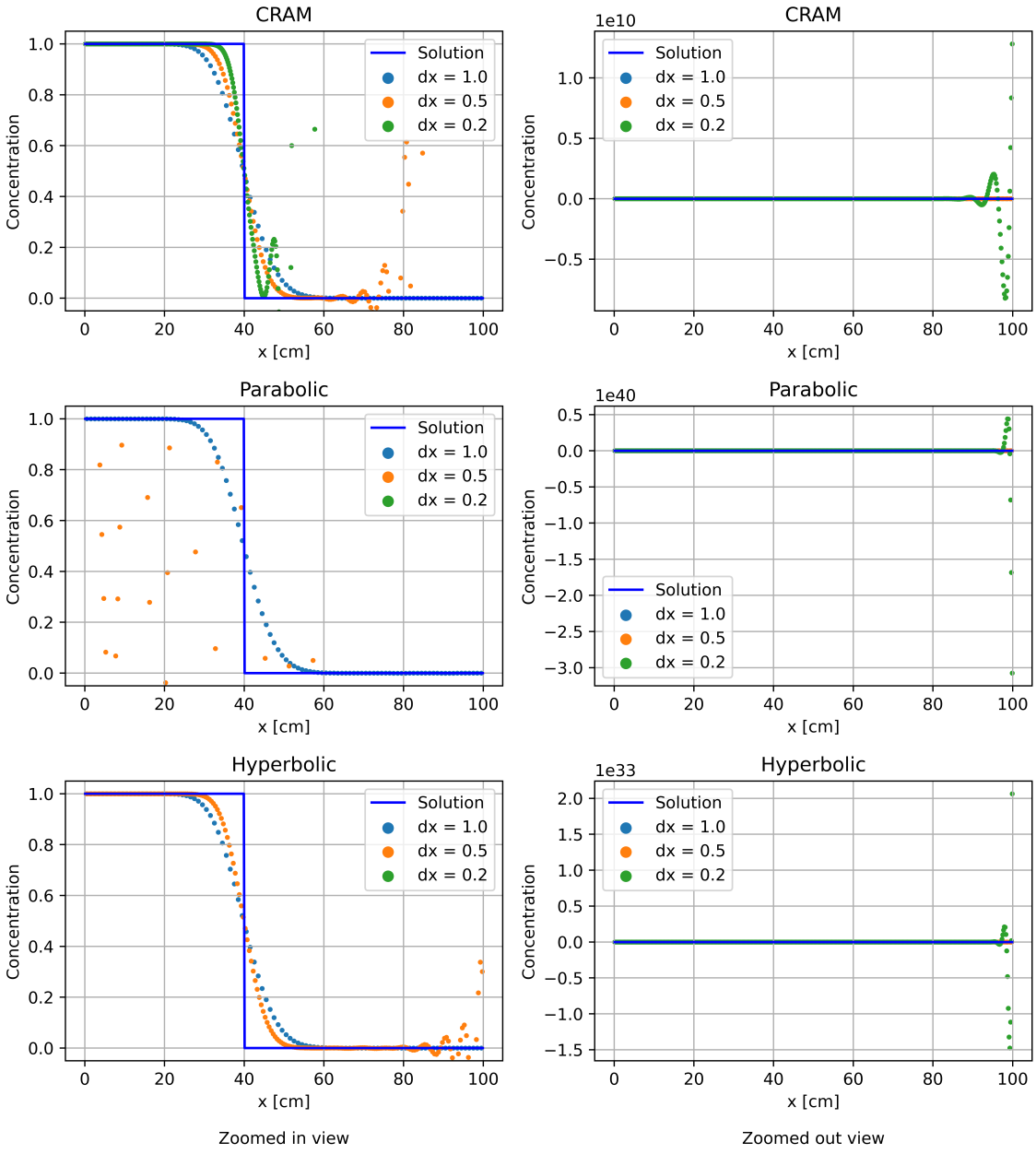

Figure 5.8: First-order upwind differencing instability for Cauchy solvers

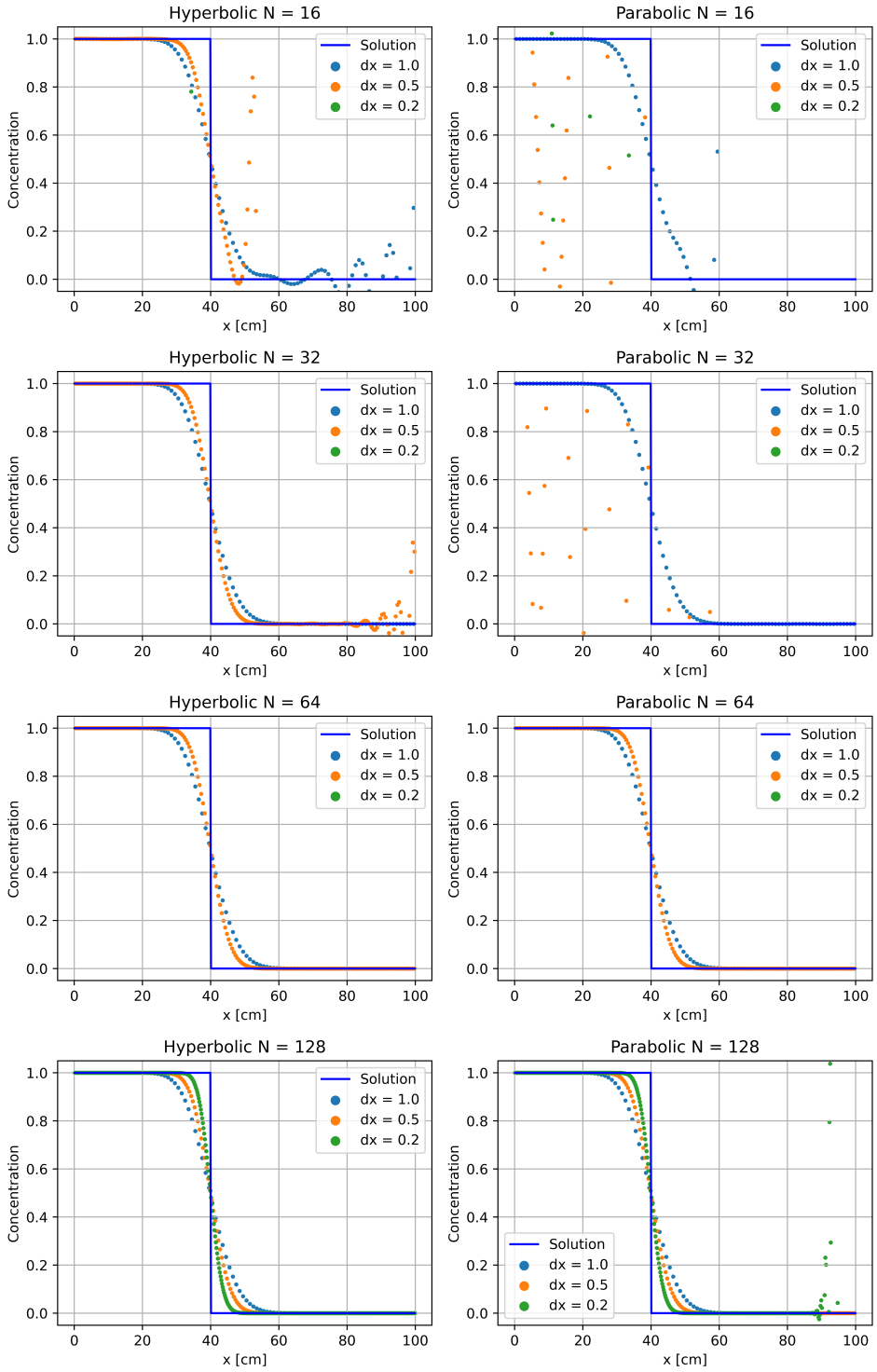

Figure 5.9: First-order upwind differencing instability for Cauchy solvers at different orders

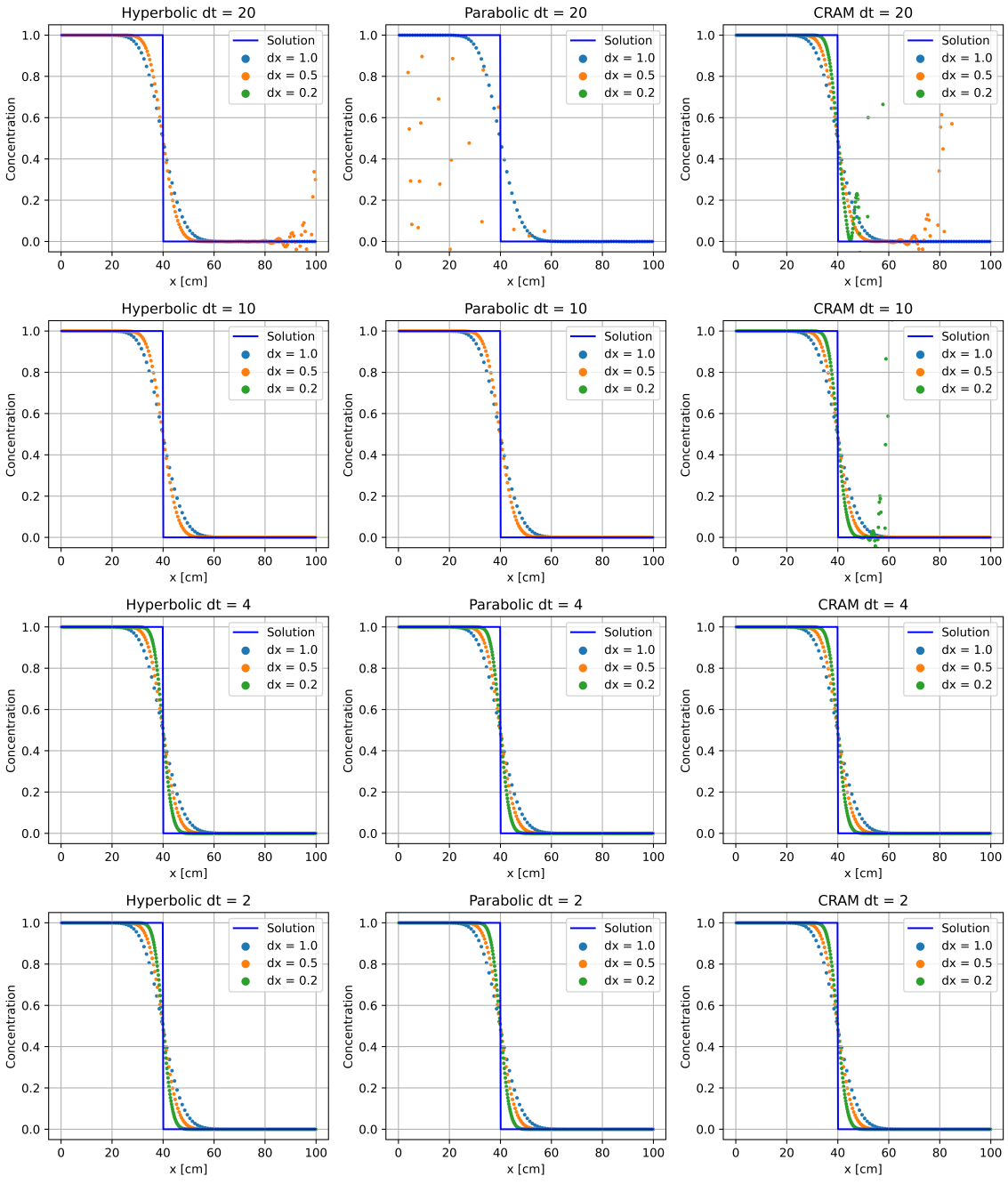

Figure 5.10: First-order upwind differencing instability for Cauchy solvers at different time step sizes

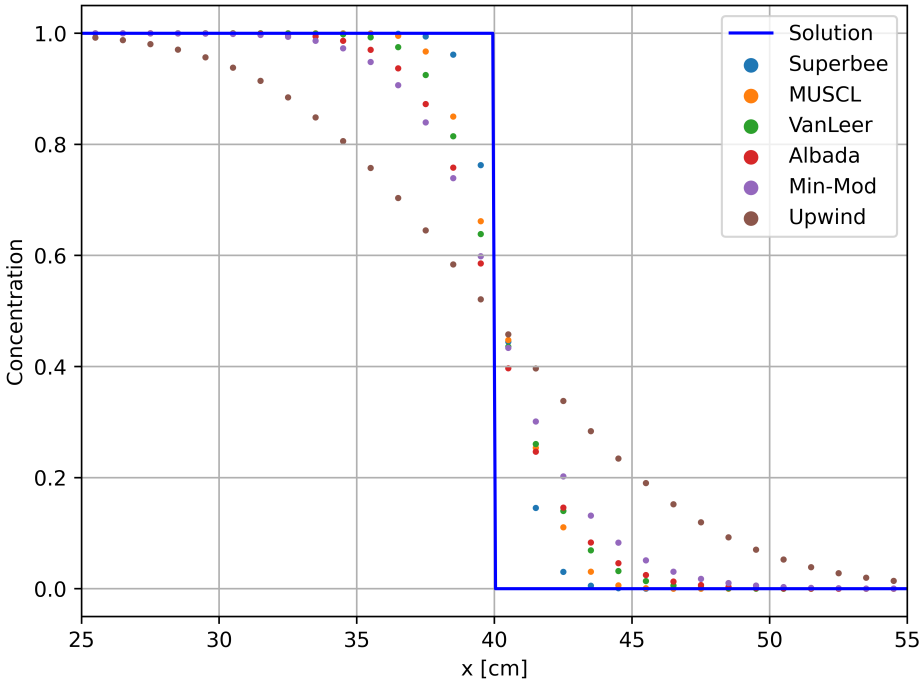

Figure 5.11: Flux limiter function applied to problem three, dt = 0.1, dx = 0.2

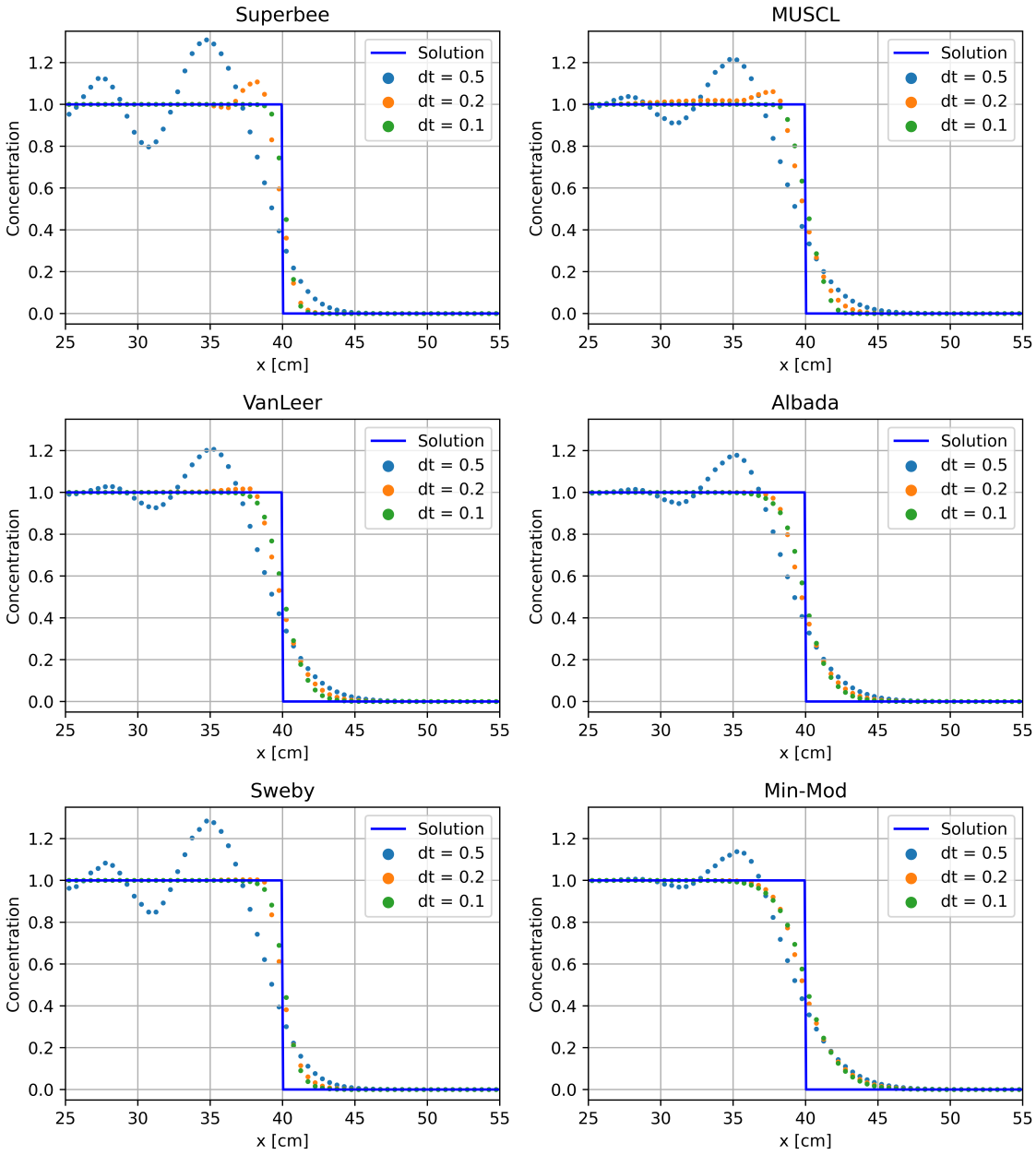

Figure 5.12: Instability of flux limiter functions with the Taylor solver

5.1.4 Problem 4

The same PDE that was shown in the previous problem is tested again:

$$\frac{\partial U}{\partial t} = -v\frac{\partial U}{\partial x}, \tag{5.11}$$

on the domain $x \in [0, 100]$, $t =\in [0, 5]$ subject to periodic boundary conditions:

$$U(0, t) = U(100, t), \tag{5.12}$$

and the initial condition:

$$U(x, 0) = \exp\left\{ -\left(\frac{(x - 30)}{10}\right)^2\right\}. \tag{5.13}$$

This problem produces a smooth solution which shows the convergence order of the flux limiters. Figures 5.13 and 5.14 show results for each flux limiter function using the Taylor solver. Figure 5.13 shows l_1 error behavior as a function of spatial discretization at various time step sizes. Figure 5.14 shows the same error behavior as a function of temporal discretization at various number of spatial cells. Figure 5.13 shows that as the number of time steps take is increased the higher order flux limiters begin to show a linear relationship with spatial discretization. This can be readily seen with the number of steps greater than 40. The upwind limiter is unaffected by decreasing the time step size, which is to be expected. This is because increasing the number of times steps do nothing to increase its accuracy. Figure 5.14 shows that increasing the number of time steps has little to no affect on the error for large spatial discretization. As the number of cells increases, decreasing the time step size has a greater affect on reducing the error. This figure also reiterates the fact that the first order upwind error is not affected by increasing the number of time steps taken. Results for the l_∞ and l_2 errors are shown in Appendix C figures 1, 2, 3 and 4. Figure 1 shows a similar trend with spatial refinement using the l_∞ error to the l_1 error however, the error function in figure 1 does not maintain the same linear rate. This relation is further represented in Figure 2 which shows the error with temporal refinement. For large spatial refinement, the error remains constant. For smaller refinement, the error takes a dip then

increases. The l_2 error shown in figures 3 and 4 shows a trend resembling the l_1 error, in which the error convergence rate remains semi linear as a function of spatial discretization for most flux limiters with larger number of time steps.

Figure 5.15 shows convergence rates for each of the higher order flux limiter functions as a function of space and time using the l_1 error. Each flux limiter shows a dark zone (representing low convergence rate) with high spatial discretization and low temporal discretization. At 320 spatial cells, the convergence rate of each flux limiter increases until reaching an apex at 40 time steps, then slightly decreases to a constant-ish value. At low spatial resolution, increases the number of time steps taken does not have a large affect on changing the convergence rate. Results for the l_∞ and l_2 error metrics are shown in Appendix C figures 5 and 6. Behavior of the l_2 error shows similar trends to l_1, low to negative convergence rates for high spatial resolution and low temporal, with most limiters showing peak convergence with 40 time steps and 320 cells. Convergence rates for l_∞ error are shown in figure 5. These results show convergence rates which are slightly worse than the other two errors. The peak convergence rate for most flux limiters has also shifted from 320 cells and 40 time steps to 320 cells and 8 time steps. All limiter functions also show a general trend of having larger convergence rates at lower spatial resolution.

Figure 5.16 shows the average run time results for each of the solvers. Each flux limiter function call requires the same time meaning that small run time differences for each solve is due to the computers computation. For the small cases, the Taylor and Padé solvers out perform the Cauchy solvers. As the number of cells increase, the Padé solver run times begin to approach values greater than the Cauchy solvers. Hyperbolic and Parabolic remain about the same as seen before, with CRAM being the lowest of the three. The Taylor solver remains the lowest for all problem sizes.

Unlike the previous example, not instabilities were noticed while conducting the numerical test. Even at large spatial resolutions and small time steps, the flux limiter functions remained stable. Each of the Cauchy solvers also remained stable during testing. While only results for the Taylor solver are shown, all other solvers show the same errors and convergence rates.

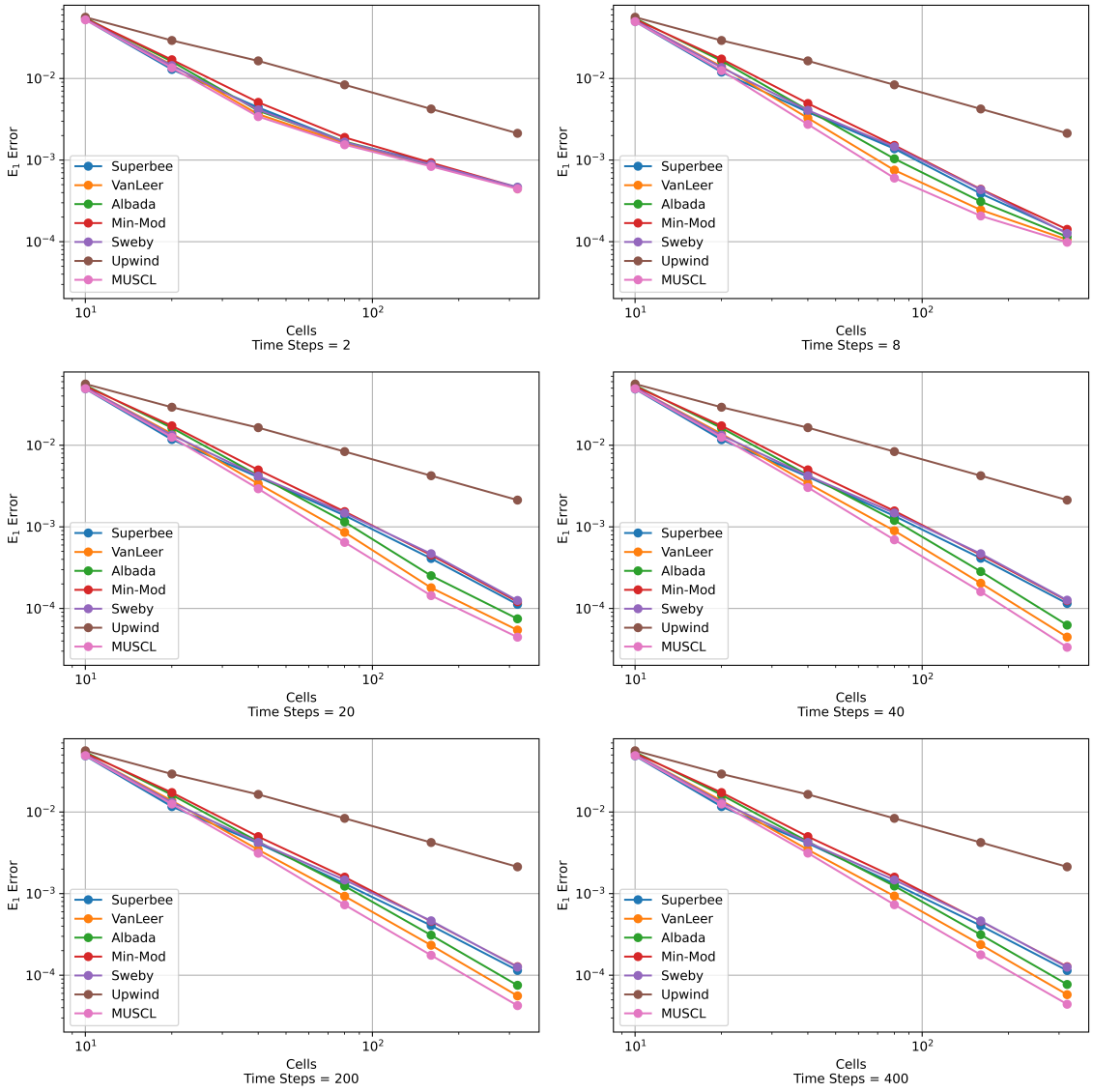

Figure 5.13: Problem 4 absolute l_1 error with spatial discretization at various time steps

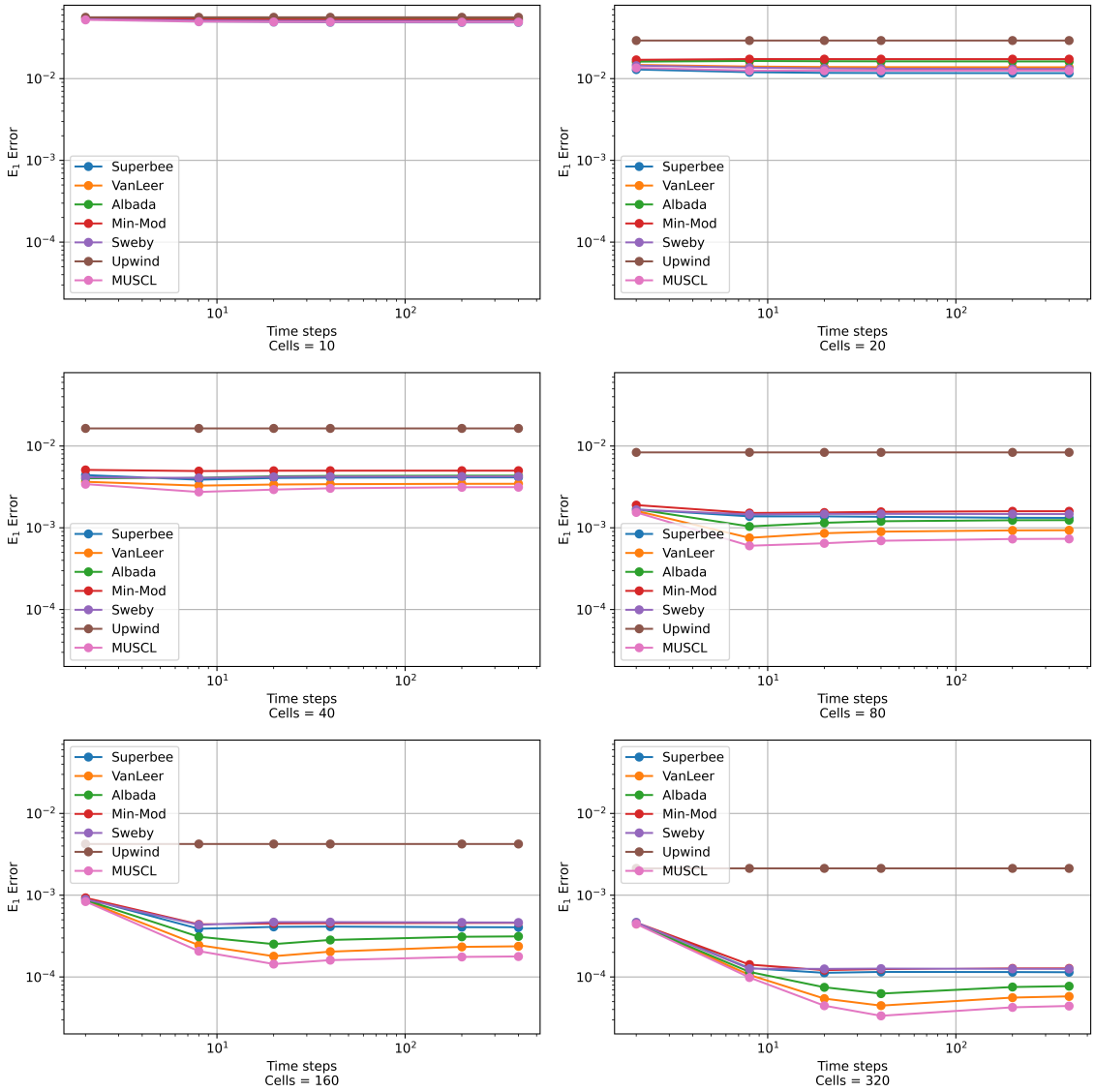

Figure 5.14: Problem 4 absolute l_1 error with temporal discretization at various number of spatial cells

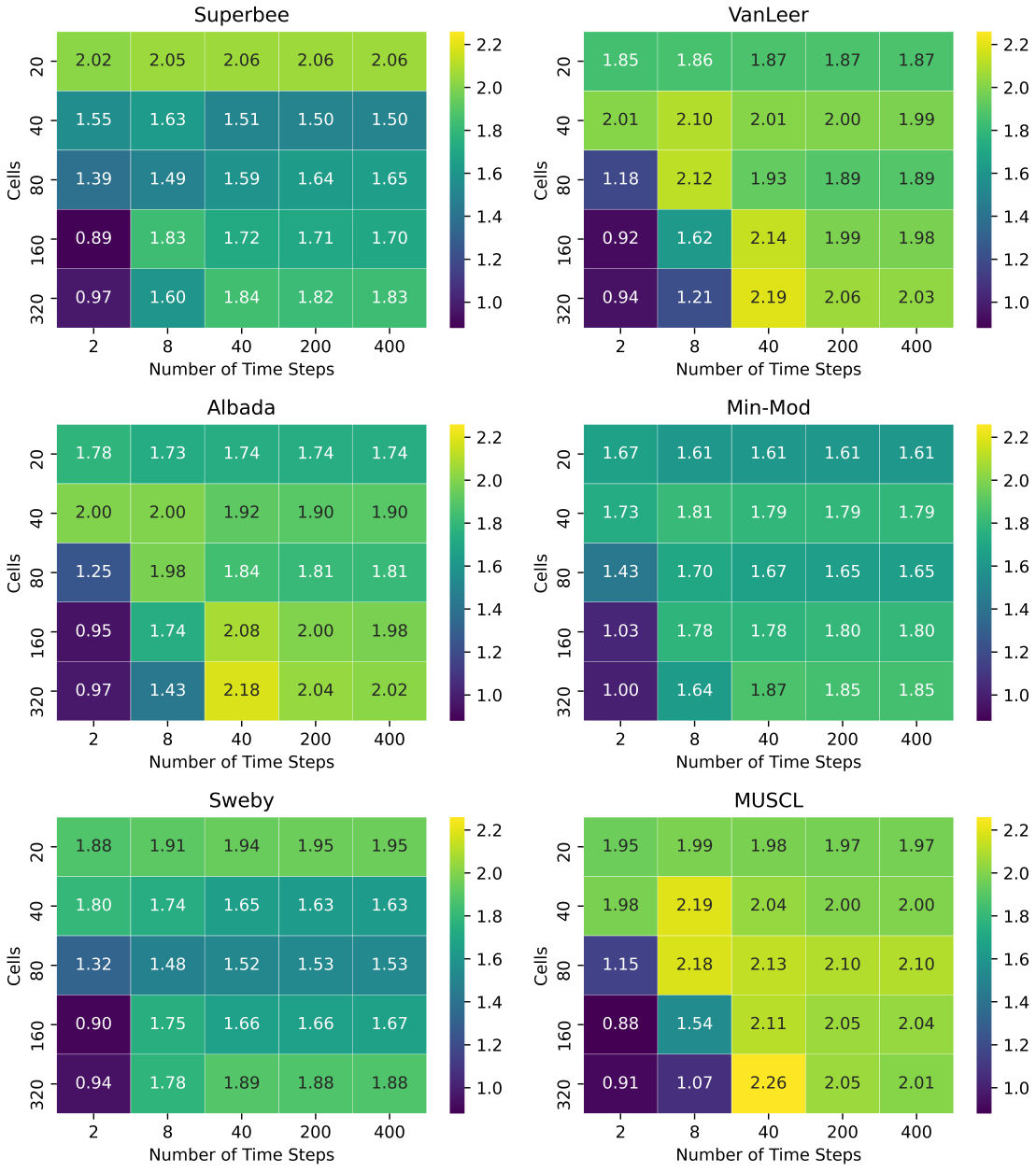

Figure 5.15: Problem 4 flux limiter l_1 convergence rate using absolute error

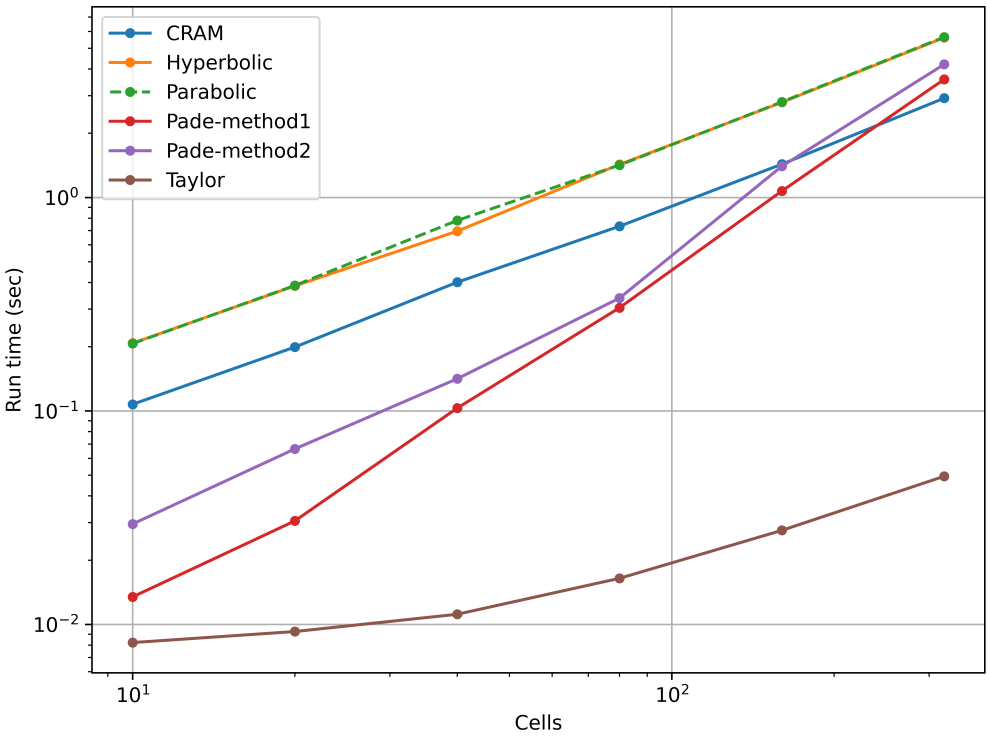

Figure 5.16: Problem 4 average run times for each solver, dt = 0.01

5.1.5 Problem 5

This problem consist on a convection-reaction problem with a single species represented by:

$$\frac{\partial C}{\partial t} = -v\frac{\partial C}{\partial x} - \lambda C, \tag{5.14}$$

on the domain $x \in [0, 100]$, $t \in [0, 20]$ where $v = 2$ and $\lambda = 0.01$. Subject to the following conditions:

$$C(x, 0) = 1000, \quad C(0, t) = 1000, \quad \frac{dC}{dx}(100, t) = 0. \tag{5.15}$$

This leads to the analytic solution:

$$C(x, t) = \begin{cases} C(x, 0)e^{-\frac{x\lambda_i}{v}}, & x < vt \\ C(x, 0)e^{-t\lambda_i}, & x \geq vt \end{cases} \tag{5.16}$$

This problem is a 1D convection driven flow where the species decays at a constant rate. The solution to this leads to a function that is not smooth, causing the higher order flux limiter functions to reduced to first order accuracy. Figures 5.17 and 5.18 show results for each of the flux limiter functions for the l_1 relative error for spatial and temporal discretization. Starting with figure 5.17, each of the flux limiters show about the same convergence rate at the first order upwind function. This agrees with the previous statement about TVD schemes reducing to first order convergence rates for non smooth solutions. While the TVD schemes do not have second order convergence rates, they do increase the accuracy of the solution when compared to first order upwind. This is most prevalent with the Albada limiter, which does show are higher convergence rate with large number of cells and time steps. This limiter also maintains the lowest error for all test shown if figures 5.17 and 5.18. Albada also benefits the most from increasing the number of time steps, this is easly seen in figure 5.18. Figure 5.18 also shows that each of the other flux limiter functions are not greatly affected by increasing the number of time steps. For smaller spatial discretizations, increasing the number of time steps does slightly increase the accuracy of the other TVD schemes. Results using the l_∞ and l_2 errors are found in Appendix C figures 7, 8,

9 and 10. Figure 7 shows that the l_∞ error behaves a little worse than l_1, showing different flux limiters achieve minimal error at higher spatial resolution. While Albada is still one of the best, it does not have the lowest error for higher spatial and temporal resolutions. The l_2 error, shown in figure 9, shows interesting results, as the error appears to get worse as the number of cells is increased for almost all time step sizes. The notable exception to this is the Albada limiter, which shows reduction in error for low dt values.

Figure 5.19 shows convergence rates as a function of time and space. Unlike problem 4, which showed minimal convergence rate in the bottom left corner, these results show the minimal to be in the top left. This corresponds to low spatial and temporal resolution. For each limiter function, the convergence rate is increased by increasing the number of time steps taken. Just as with figures 5.17 and 5.17, figure 5.19 shows the the Albada limiter has the highest convergence rate, reaching a sweet spot at 80 cells and 400 time steps. Figures 11 and 12 shows convergence rates for l_∞ and l_2 errors. The l_∞ error shows high convergence rates for for low cells and high time steps for all but the Albada and Min-Mod limiters. As expected, the l_2 error shows negative convergence rates for all but the Albada limiter.

Figure 5.20 shows how the problem behaves as a function of v and λ. These coefficients are changed to induce and reaction or velocity dominated system. In a reaction dominated system, all of the flux limiter functions converge to a single point. This is because the accuracy of the solution is dominated by the integration of the reaction rate of the time interval. As the velocity is increased, the flux limiters begin to diverge to difference accuracy's. The Albada limiter diverges the most, decreasing the error 2 orders of magnitude below all the others. What is also interesting is that the error follows a hump shape, increasing up until it reaches a max value where the coefficient are the same. After this, the error decreases. This is also the point where the Albada separates from the rest.

Figure 5.21 show how each of the solvers scale with number of cells. On the log-log scale, the Cauchy and Taylor solvers scale linearly while the Padé solvers begin low then increase to a point above the others. This indicates that the Padé solvers scale much more poorly. As with problem 3, the Cauchy solvers did how instability. These instabilities were also remedied by increasing the order of the method and by adding substeps.

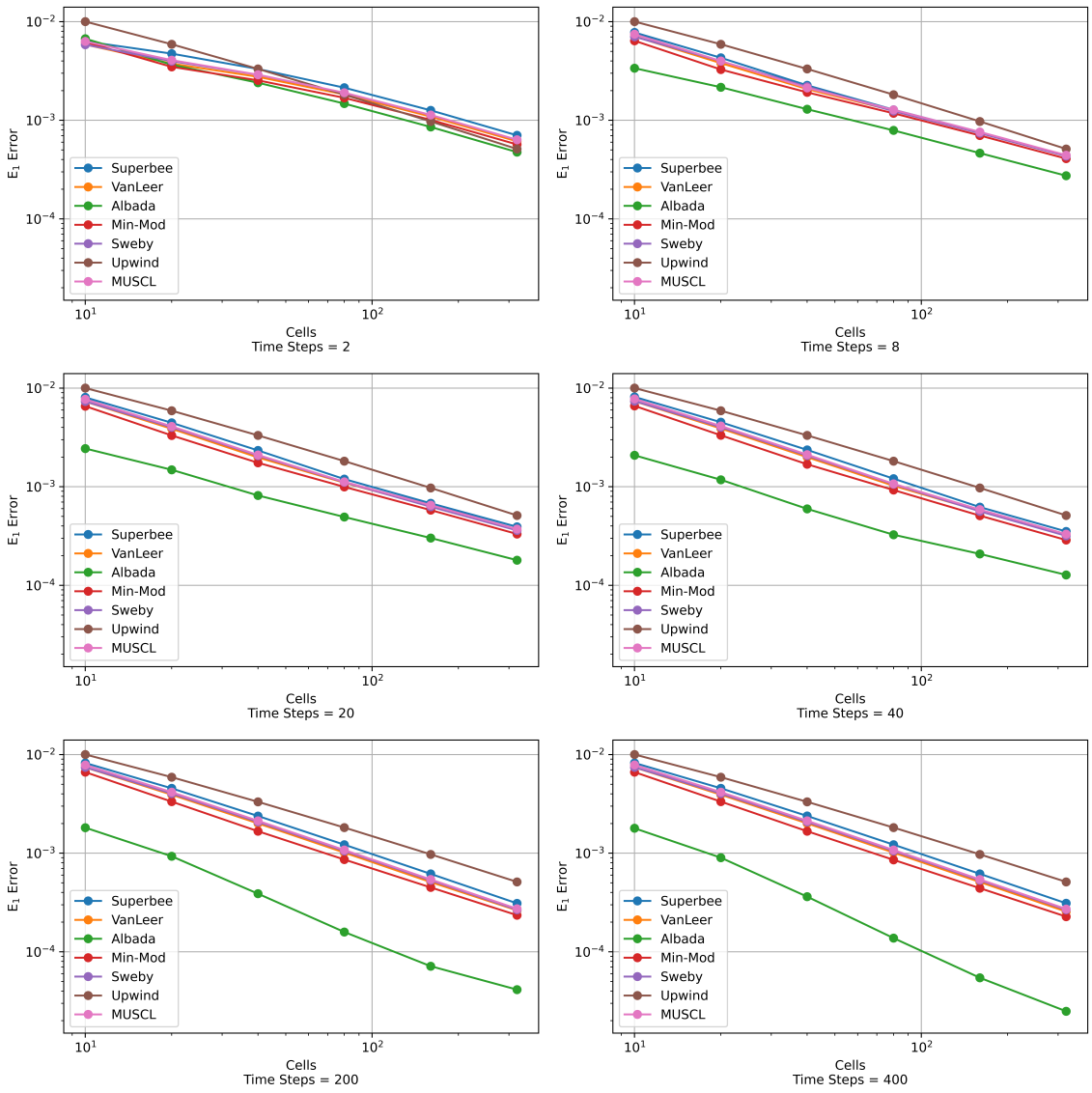

Figure 5.17: Problem 5 absolute l_1 error with spatial discretization at various time steps

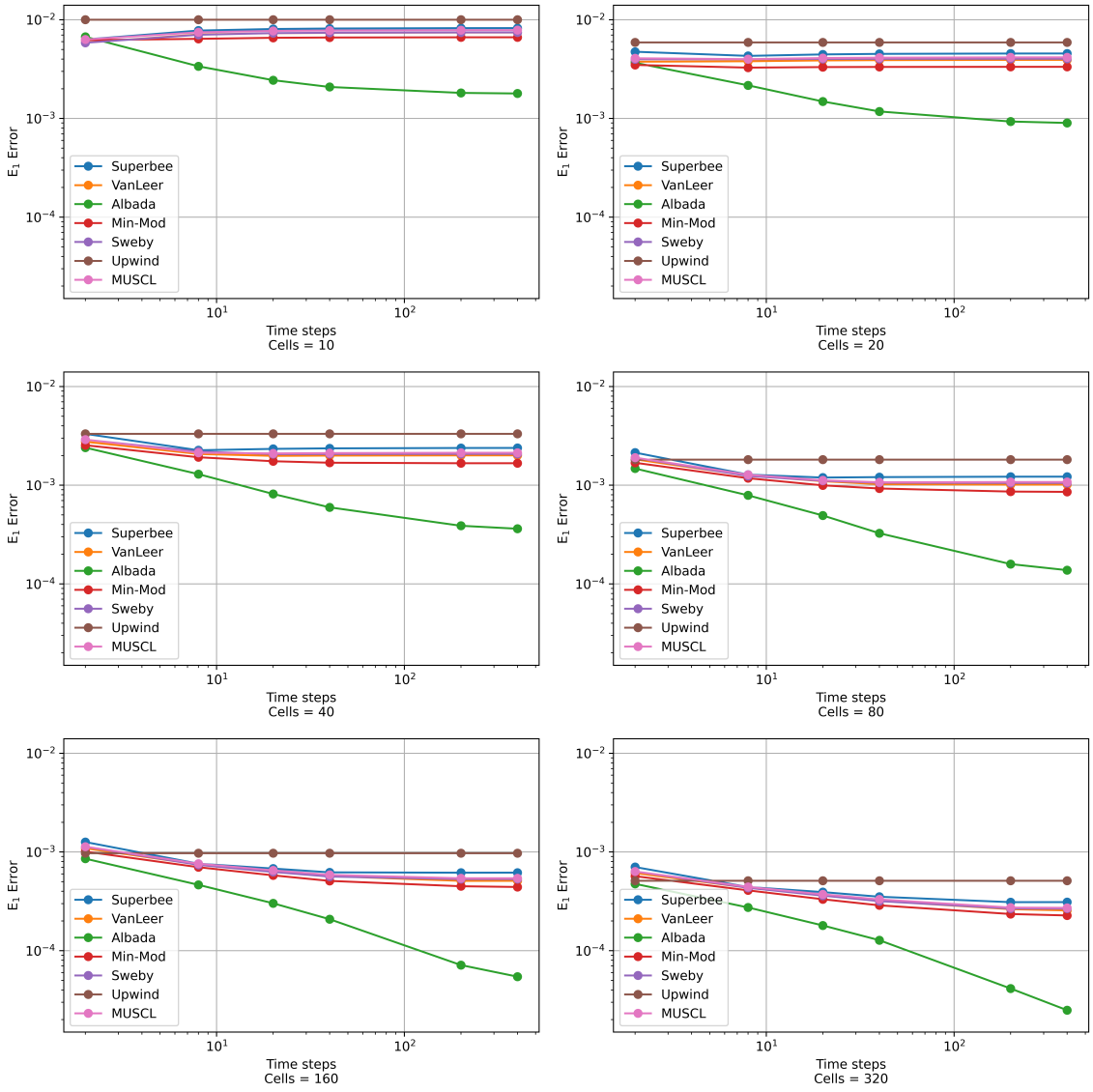

Figure 5.18: Problem 5 absolute l_1 error with temporal discretization at various number of spatial cells

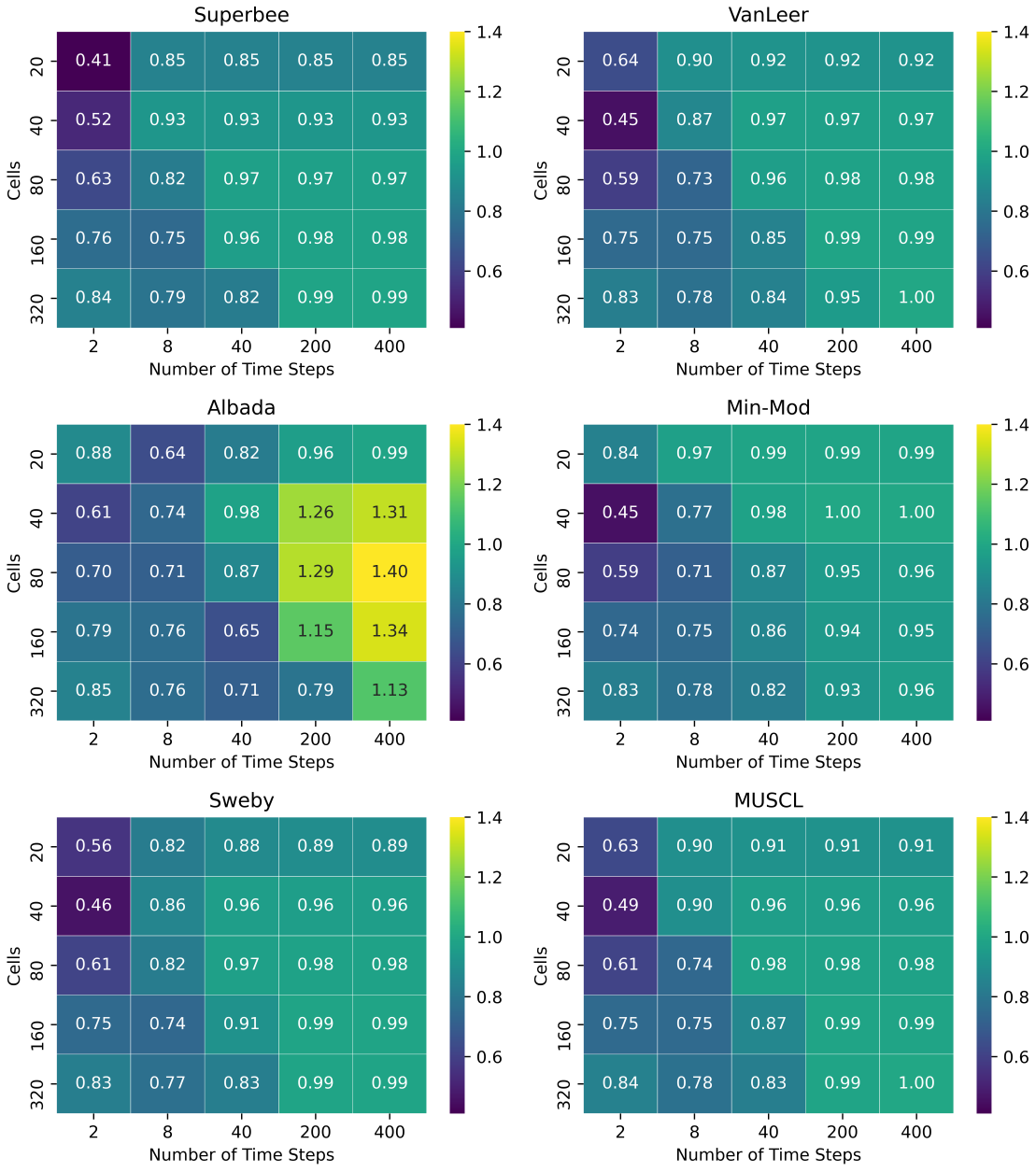

Figure 5.19: Problem 5 flux limiter l_1 convergence rate using absolute error

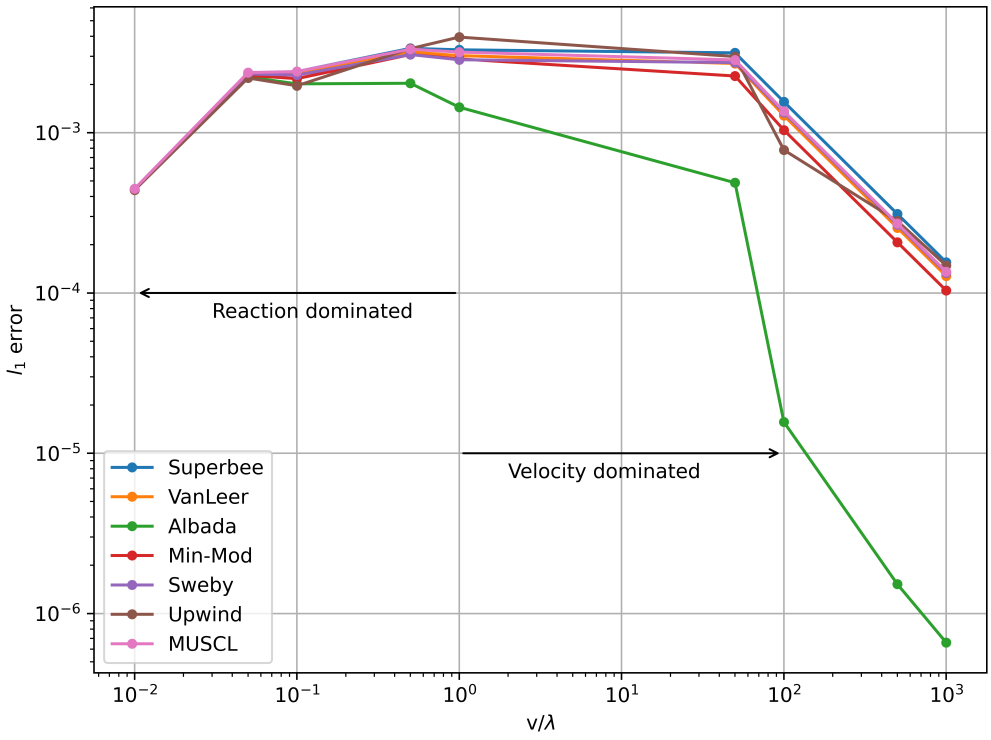

Figure 5.20: Problem 5 flux limiter l_1 behavior with coefficient changes using absolute error

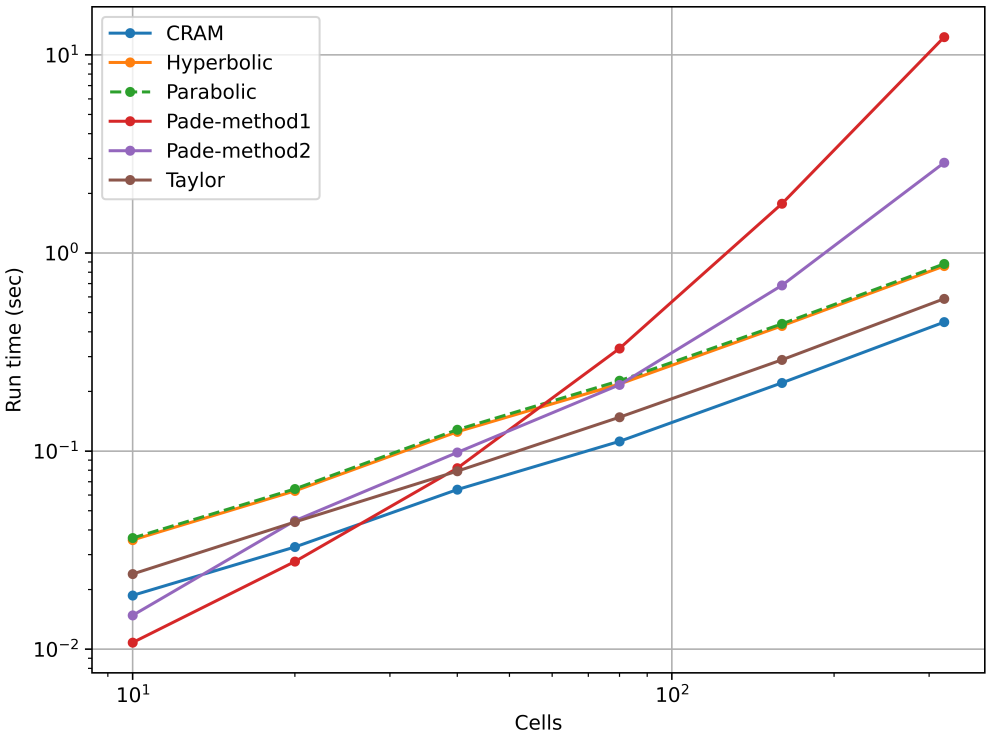

Figure 5.21: Problem 5 average run times for each solver, dt = 0.1

5.1.6 Problem 6

This is an extension of problem 6 with two species, one that exist in the liquid and one on the wall. The species in the liquid transports and depositions on the wall where it sticks and does not move. This system is represented by:

$$\begin{aligned} \frac{\partial C_l}{\partial t} &= -v\frac{\partial C_l}{\partial x} - \frac{kA}{V}C_l, \\ \frac{\partial C_w}{\partial t} &= \frac{kA}{V}C_l, \end{aligned} \tag{5.17}$$

on the domain $x \in [0, 100]$, $t \in [0, 20]$ where $v = 2$ and $kA/V = \lambda = 0.01$. Subject to the following conditions:

$$\begin{aligned} C_l(x,0) = 1000, \quad C_l(0,t) = 1000, \quad \frac{dC_l}{dx}(100,t) = 0, \\ C_w(x,0) = 0, \quad C_w(0,t) = 0, \quad \frac{dC_l}{dx}(100,t) = 0. \end{aligned} \tag{5.18}$$

Let $\lambda = kA/V$, equation 5.17 the following solution:

$$C_l(x,t) = \begin{cases} C_l(x,0)e^{-\frac{x\lambda_i}{v}}, & x < vt \\ C_l(x,0)e^{-t\lambda_i}, & x \geq vt \end{cases} \tag{5.19}$$

$$C_w(x,t) = \begin{cases} C_l(x,0)\Big(1 - e^{-\frac{x\lambda_i}{v}} + \lambda\big(t - x/v\big)e^{-\lambda x/v}\Big), & x < vt \\ C_l(x,0)\Big(1 - e^{-t\lambda_i}\Big), & x \geq vt \end{cases} \tag{5.20}$$

This problem is conducted in the same manor as problem 4 and 5. As with problem 5, the solution for C_l

5.1.7 Problem 7

This problem models the primary uranium isotopes along with Pu-239 and Np-239 in a system which contains a neutron flux. Coefficients in the transition matrix include source

and sink terms from both neutron induced reactions and radioactive decay. This system is represented by:

$$\frac{dC_i}{dt} = \sum_{j=1}^{9} A_{ij} C_j(x,t) \tag{5.21}$$

$$i = \begin{cases} 1, & ^{233}\mathrm{U} \\ 2, & ^{234}\mathrm{U} \\ 3, & ^{235}\mathrm{U} \\ 4, & ^{236}\mathrm{U} \\ 5, & ^{237}\mathrm{U} \\ 6, & ^{238}\mathrm{U} \\ 7, & ^{239}\mathrm{U} \\ 8, & ^{239}\mathrm{Pu} \\ 9, & ^{239}\mathrm{Np} \end{cases} \tag{5.22}$$

On the domain from $t \in [0, 500]$ subject to the following neutron flux $\phi = 1e13$. Each isotope is given an initial concentration of $C_{i,0} = 1e10$. The transition matrix is built using information from the SCALE ORIGEN library. The analytical solution is given by the exponential of the transition matrix:

$$\boldsymbol{C}(t) = e^{\boldsymbol{A}t}, \tag{5.23}$$

where $\boldsymbol{C}$ is a vector of isotope concentrations. The solution is calculated using in MATLAB using the method previously described.

5.1.8 Problem 8

This is an extension of problem 9 but for a fictitious "pipe" reactor which allows for the flow of isotopes due to a velocity. This is represented by:

$$\frac{dC_i}{dt} = -v\frac{\partial C_i}{\partial x} + \sum_{j=1}^{9} A_{ij}C_j(x,t) \tag{5.24}$$

$$i = \begin{cases} 1, & ^{233}\mathrm{U} \\ 2, & ^{234}\mathrm{U} \\ 3, & ^{235}\mathrm{U} \\ 4, & ^{236}\mathrm{U} \\ 5, & ^{237}\mathrm{U} \\ 6, & ^{238}\mathrm{U} \\ 7, & ^{239}\mathrm{U} \\ 8, & ^{239}\mathrm{Pu} \\ 9, & ^{239}\mathrm{Np} \end{cases} \tag{5.25}$$

On the domain $x \in [0, 400]$, $t \in [0, 500]$ where $v = 25$ and $\phi(x) = 1e13$. Subject to the periodic boundary conditions:

$$C_i(0,t) = C_i(400,t). \tag{5.26}$$

5.2 Case Studies

The following are a selection of case studies which aim to explore libowskis' accuracy and performance with problems which come up in modeling MSRs.

5.2.1 Neutron precursors

This problem examines a convection-driven flow with the 6 neutron precursor groups, as shown in Eqs. (5.27) and (5.28):

$$\frac{\partial C_i}{\partial t} = -v_x\frac{\partial C_i}{\partial x} - v_y\frac{\partial C_i}{\partial y} + \beta_i\Psi(x,y) - \lambda_i C_i, \tag{5.27}$$

$$\Psi(x,y) = \begin{cases} \psi_0 \sin\left(\frac{\pi x}{50}\right) \sin\left(\frac{\pi y}{100}\right), x \in [0,50], y \leq 100 \\ 0 \text{ , otherwise,} \end{cases} \tag{5.28}$$

where $i \in [1,6]$, $x \in [0,50]$, $y \in [0,400]$, $t \in [0,60]$, $v_x = 0$, and $v_y = 25$, subject to the following boundary conditions:

$$C_i(x,0) = C_i(x,400), \quad \frac{dC_i}{dx}(0,y) = 0, \quad \frac{dC_i}{dx}(50,y) = 0. \tag{5.29}$$

Coefficients for the system are shown in Table 5.6 [**?**]. Each precursor had the same initial condition of zero, and the source term for each precursor was scaled in the x and y directions by the sine function. The spatial domain was modeled to mimic an MSR with a core region extending from $y \in [0,100]$ and $x \in [0,50]$, with a core exterior loop modeled from $y \in [100,400]$.

Table 5.6: Parameters for Neutron Precursors

Group	λ	β
1	0.0127	0.0006
2	0.0317	0.00364
3	0.115	0.00349
4	0.311	0.00628
5	1.4	0.00179
6	3.87	0.0007

While there is no analytic solution for this example problem, a reference solution was generated in Matlab using the symbolic tool box to solve for the matrix exponential. A small, spatially discretized problem was set up with 5 cells in the x direction and 20 in the y direction. The transition matrix that was built was then exported to Matlab, and the matrix exponential was solved at various times and saved as the reference solution. While the spatial accuracy was not tested in this case, this method will access the accuracy of each matrix exponential algorithm. To further simplify the problem, the first-order upwind difference scheme was applied to the convective flux.

One important feature for many of these solvers was the location of the eigenvalues for the transition matrix. The spectrum was calculated using the Matlab symbolic tool box and

is plotted in Figure 5.22 for dx = 10, dy = 20, and various values of dt. Figure 5.22 show 6 elliptical rings, one at each dt, each one representing a precursor group. For a given time step size, the eigenvalues shifted along lines with slopes that are the ratio of their real and imaginary parts [27]. For a system in which the eigenvalues are located in a region where solutions based on Cauchy's integral break down, these eigenvalues can be shifted into a region where the solutions hold. Figure 5.22 shows how changing the time step size can shrink the real and imaginary parts of the eigen spectrum.

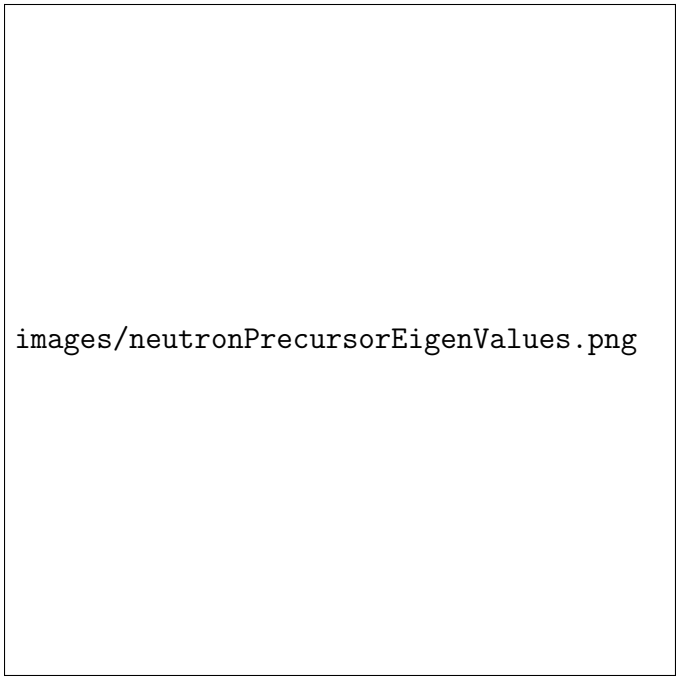

Figure 5.22: Spectrum for the Neutron Precursors

To understand how sub-stepping can work to improve the accuracy of a Cauchy-based solver, the neutron precursors problem was computed at 10-second time step intervals. The location of the eigenvalues is a function of the ratio v_x/dx in the transition matrix; therefore, at the prescribed discretization, this ratio was manipulated by changing the flow velocity. For each Cauchy solver, the eigenvalues at two different flow velocities, 25 cm/s ($v_x/dx = 2.5$) and 60 cm/s ($v_x/dx = 6$), are shown in Figure **??**. As shown in Figure **??**, as the ratio increased, so did the spread of the eigenvalues on the real and imaginary axis. This led to a limitation of the velocity to discretization size for convection problems when using Cauchy solvers. One solution, which is also shown in Figure **??**, is to use sub-stepping to reduce

the time step size, thus confining the eigenvalues. As the number of substeps is increased, the eigenvalues become confined in a region in which the contour encloses the spectrum, theoretically increasing the solver's accuracy. Each plot in Figure **??** shows how the the spectrum was confined using 0, 2 and 6 substeps. As the number of substeps increased, the spectrum shrank into the confines of the contour.

5.2.2 Small case lump depletion mass transport

This is a depletion problem using the selected isotopes in Table 1 represented by the following equations:

5.2.3 Medium case lump depletion mass transport

Matlab analytical solution

5.2.4 Small case 2D transport

Matlab analytical solution for small case

5.2.5 Medium case 2D transport

The final boss. No analytical solution required

Chapter 6

Conclusions

Bibliography

[1] Maria Pusa. *Numerical methods for nuclear fuel burnup calculations: Dissertation*. PhD thesis, Aalto University, Finland, 2013. ix, 5, 6, 7, 9, 13, 25, 29, 37

[2] James J. Duderstadt and Louis J. Hamilton. *Nuclear Reactor Analysis*. John Wiley and Sons, 1976. 5, 6, 8, 11

[3] E.E. Lewis and W.F. Miller. *Computational methods of neutron transport*. John Wiley & Sons, Ltd, 1 1984. 10, 11

[4] Aarno Isotalo. *Computational Methods for Burnup Calculations with Monte Carlo Neutronics*. PhD thesis, Aalto University, Finland, 2013. 12, 13, 47

[5] A.E. Isotalo and P.A. Aarnio. Comparison of depletion algorithms for large systems of nuclides. *Annals of Nuclear Energy*, 38(2):261 – 268, 2011. 12, 37

[6] Maria Pusa and Jaakko Leppänen. Computing the matrix exponential in burnup calculations. *Nuclear Science and Engineering*, 164(2):140–150, 2010. 12, 22, 24, 30, 34, 37

[7] Akio YAMAMOTO, Masahiro TATSUMI, and Naoki Sugimura. Numerical solution of stiff burnup equation with short half lived nuclides by the krylov subspace method. *Journal of Nuclear Science and Technology - J NUCL SCI TECHNOL*, 44:147–154, 02 2007. 12, 34

[8] R. Byron Bird, Warren E. Stewart, and Edwin N. Lightfoot. *Transport Phenomena*. John Wiley and Sons, Inc., New York, revised 2 edition, 2006. 13, 15, 16

[9] Robert Zachary Taylor. Implementation of multi-phase species transport into vera-cs for molten salt reactor analysis. Master's thesis, University of Tennessee, Knoxville, 2019. 13

[10] E. L. Compere, S. S. Kirslis, E. G. Bohlmann, F. F. Blankenship, and W. R. Grimes. *Fission Product Behavior in the Molten Salt Reactor Experiment*. Oak Ridge National Laboratory, Oak Ridge, TN, 1975. ORNL–4865. 14

[11] Don W. Green and Robert H. Perry. *Perry's Chemical Engineers' Handbook*. McGraw Hill Professional, New York, 8th edition, 2007. 15

[12] R. J. Kedl. *The Migration of a Class of Fission Products (Noble Metals) in the Molten-Salt Reactor Experiment*. Oak Ridge National Laboratory, Oak Ridge, TN, 1972. ORNL–3884. 15, 16, 48

[13] H.A. Ashi, L.J. Cummings, and P.C. Matthews. Comparison of methods for evaluating functions of a matrix exponential. *Applied Numerical Mathematics*, 59(3):468 – 486, 2009. Selected Papers from NUMDIFF-11. 18, 19, 21

[14] Kendall Atkinson, Weimin Han, and David Stewart. *Stiff differential equations*, chapter 8, pages 127–148. John Wiley & Sons, Ltd, 2011. 18

[15] S.M. Cox and P.C. Matthews. Exponential time differencing for stiff systems. *Journal of Computational Physics*, 176(2):430 – 455, 2002. 18, 21

[16] A.G. Bratsos and A.Q.M. Khaliq. An exponential time differencing method of lines for burgersâ-fisher and coupled burgers equations. *Journal of Computational and Applied Mathematics*, 356:182 – 197, 2019. 18

[17] Aly khan Kassam and Lloyd N. Trefethen. Fourth-order time stepping for stiff pdes. *SIAM J. SCI. COMPUT*, 26:1214–1233, 2005. 18, 19

[18] A. Bratsos and A Q. Khaliq. A conservative exponential time differencing method for the nonlinear cubic schrödinger equation. *International Journal of Computer Mathematics*, 94:1–25, 09 2015. 20

[19] D.G. Zill and W.S. Wright. *Differential Equations with Boundary-Value Problems*. Cengage Learning, 2012. 21

[20] Roger Sidje. Expokit: A software package for computing matrix exponentials. *ACM Trans. Math. Softw.*, 24:130–156, 03 1998. 22, 24

[21] C. Moler and C. Van Loan. Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later. *SIAM Review*, 45(1):3–49, 2003. 22, 23, 24, 54

[22] Awad H. Al-Mohy and Nicholas J. Higham. Computing the action of the matrix exponential, with an application to exponential integrators. *SIAM Journal on Scientific Computing*, 33(2):488–511, 2011. 23, 55

[23] Nicholas J. Higham. The scaling and squaring method for the matrix exponential revisited. *SIAM Journal on Matrix Analysis and Applications*, 26(4):1179–1193, 2005. 24, 25, 54

[24] Awad Al-Mohy and Nicholas Higham. A new scaling and squaring algorithm for the matrix exponential. *SIAM Journal on Matrix Analysis and Applications*, 31, 01 2009. 24, 25, 54, 109

[25] Maria Pusa. Rational approximations to the matrix exponential in burnup calculations. *Nuclear Science and Engineering*, 169(2):155–167, 2011. 25, 27, 29, 53

[26] L. N. Trefethen, J. A. C. Weideman, and T. Schmelzer. Talbot quadratures and rational approximations. *BIT Numerical Mathematics*, 46(3):653–670, Sep 2006. 25, 26, 29

[27] Maria Pusa. Accuracy considerations for chebyshev rational approximation method (cram) in burnup calculations. In *Proceedings*, pages 973–984, 2013. International Conference on Mathematics and Computational Methods Applied to Nuclear Science and Engineering, M&C 2013, M&C 2013 ; Conference date: 05-05-2013 Through 09-05-2013. 30, 32, 96

[28] Aarno Isotalo and Maria Pusa. Improving the accuracy of the chebyshev rational approximation method using substeps. *Nuclear Science and Engineering*, 183, 05 2016. 31, 56

[29] Y. Saad. Analysis of some krylov subspace approximations to the matrix exponential operator. *SIAM Journal on Numerical Analysis*, 29(1):209–228, 1992. 32, 33, 34

[30] Efstratios Gallopoulos and Yousef Saad. On the parallel solution of parabolic equations. In *ICS '89*, 1989. 32, 33

[31] Maria Pusa and Jaakko Leppänen. Solving linear systems with sparse gaussian elimination in the chebyshev rational approximation method. *Nuclear Science and Engineering*, 175(3):250–258, 2013. 36, 37, 52

[32] W. A. Wieselquist, R. A. Lefebvre, M. A. Jessee, and Eds. *SCALE Code System*. Oak Ridge National Laboratory, Oak Ridge, Tennessee, 2020. ORNL/TM-2005/39, Version 6.2.4, Available from Radiation Safety Information Computational Center as CCC-834. 37, 58

[33] S. Hamdi, W. E. Schiesser, and G. W Griffiths. Method of lines. *Scholarpedia*, 2(7):2859, 2007. revision #124335. 39

[34] H K Versteeg and W Malalasekera. *An Introduction to Computational Fluid Dynamics The Finite Volume Method*. Pearson Prentice Hall, 2nd edition, 2007. 40, 42, 44

[35] P. K. Sweby. High resolution schemes using flux limiters for hyperbolic conservation laws. *SIAM Journal on Numerical Analysis*, 21(5):995–1011, 1984. 44

[36] Gaël Guennebaud, Benoît Jacob, et al. Eigen v3. http://eigen.tuxfamily.org, 2010. 52

[37] James W. Demmel, Stanley C. Eisenstat, John R. Gilbert, Xiaoye S. Li, and Joseph W. H. Liu. A supernodal approach to sparse partial pivoting. *SIAM J. Matrix Analysis and Applications*, 20(3):720–755, 1999. 52

[38] Leonardo Dagum and Ramesh Menon. Openmp: an industry standard api for shared-memory programming. *Computational Science & Engineering, IEEE*, 5(1):46–55, 1998. 54

[39] Lyndon Clarke, Ian Glendinning, and Rolf Hempel. The mpi message passing interface standard. In Karsten M. Decker and René M. Rehmann, editors, *Programming Environments for Massively Parallel Distributed Systems*, pages 213–218, Basel, 1994. Birkhäuser Basel. 54

[40] Massimiliano Fratoni, Dan Shen, Germina Ilas, and Jeffrey Powers. Molten salt reactor experiment benchmark evaluation. Technical Report Project 16-10240, University of California Berkeley and Oak Ridge National Laboratory, 2020. 58

[41] Ching-Shan Chou, Yong-Tao Zhang, Rui Zhao, and Qing Nie. Numerical methods for stiff reaction-diffusion systems. *Discrete & Continuous Dynamical Systems - B*, 7:515, 2007. 64

Appendix

A Matrix Exponential Algorithms

A.1 Cauchy

Algorithm 3 Parabolic Contour Coefficients

procedure PARABOLICCONTOURCOEFFS(N)
    i = 1 ▷ Index counter for θ array
    for $k = 1, 3, 5, 7, \ldots N-1$ **do** ▷ Builds array of quadrature points
        $\theta_i = \pi k/N$
        i = i+1
    end for
    $\phi = N(0.1309 - 0.1194\theta^2 + 0.2500\theta i$ ▷ Calculate ϕ array
    ϕ' = N(-2*0.1194θ + 0.2500i) ▷ Calculate ϕ' array
    α = i/N*ElementWiseExp(ϕ)*ϕ' ▷ Calculate α array
    $\alpha_0 = 0$
    return θ, α, α_0
end procedure

Algorithm 4 Hyperbolic Contour Coefficients

procedure HYPERBOLICCONTOURCOEFFS(N)
    i = 1 ▷ Index counter for θ array
    for $k = 1, 3, 5, 7, \ldots N-1$ **do** ▷ Builds array of quadrature points
        $\theta_i = \pi k/N$
        i = i+1
    end for
    $\phi = 2.246N(1 - \sin(1.1721 - 0.3443i\theta))$ ▷ Calculate ϕ array
    $\phi' = N\cos((3443i\theta)/10000 - 11721/10000)3866489i/5000000$ ▷ Calculate ϕ' array
    α = i/N*ElementWiseExp(ϕ)*ϕ' ▷ Calculate α array
    $\alpha_0 = 0$
    return θ, α, α_0
end procedure

Algorithm 5 Matrix Exponential Approximation from Contour Integrals

procedure CAUCHYSOLVEMATRIXEXPONENTIAL($\boldsymbol{A}$, $\boldsymbol{v}_0$, t, θ, α)
    $At = \boldsymbol{A}$*t
    identity = setIdentity(At.rows(), At.cols()) ▷ Builds the identity matrix
    LUSolver.analyzePattern(At) ▷ Analyze the sparsity pattern
    $S = \theta$.size()
    for $k = 1, 2, \ldots S$ **do**
        tempAt = $At - \theta_k$*identity
        tempB = α_k*$\boldsymbol{v}_0$
        LUSolver.factorize(tempAt) ▷ Compute LU decomposition
        v = v + LUSolver.solve(tempB) ▷ Solve linear systems
    end for
    v = 2*v.real()
    v = v + α_0*v_0 ▷ Add limit at infinity
    return v
end procedure

A.2 Padé

Algorithm 6 Padé of Order 3

procedure PADE3(A, $A2$)
    identity = setIdentity(A.rows(), A.cols())
    b = [120, 60, 12, 1]
    temp = b[3]*A2 + b[1]*identity
    U = A*temp
    V = b[2]*A2 + b[0]*identity
    return U, V
end procedure

Algorithm 7 Padé of Order 5

```
1: procedure PADE5(A, A2, A4)
2:     identity = setIdentity(A.rows(), A.cols())
3:     b = [30240, 15120, 3360, 420, 30, 1]
4:     temp = b[5]*A4 + b[3]*A2 + b[1]*identity
5:     U = A*temp
6:     V = b[4]*A4 + b[2]*A2 + b[0]*identity
7:     return U, V
8: end procedure
```

Algorithm 8 Padé of Order 7

```
1: procedure PADE7(A, A2, A4, A6)
2:     identity = setIdentity(A.rows(), A.cols())
3:     b = [17297280, 8648640, 1995840, 277200, 25200, 1512, 56, 1]
4:     temp = b[7]*A6 + b[5]*A4 + b[3]*A2 + b[1]*identity
5:     U = A*temp
6:     V = b[6]*A6 + b[4]*A4 + b[2]*A2 + b[0]*identity
7:     return U, V
8: end procedure
```

Algorithm 9 Padé of Order 9

```
1: procedure PADE9(A, A2, A4, A6, A8)
2:     identity = setIdentity(A.rows(), A.cols())
3:     b = [17643225600, 8821612800, 2075673600, 302702400, 30270240, 2162160, 110880,
4:         3960, 90, 1]
5:     temp = b[9]*A8 + b[7]*A6 + b[5]*A4 + b[3]*A2 + b[1]*identity
6:     U = A*temp
7:     V = b[8]*A8 + b[6]*A6 + b[4]*A4 + b[2]*A2 + b[0]*identity
8:     return U, V
9: end procedure
```

Algorithm 10 Padé of Order 13

```
1: procedure PADE13(A, A2, A4, A6)
2:     identity = setIdentity(A.rows(), A.cols())
3:     b = [64764752532480000, 32382376266240000, 7771770303897600,
4:         1187353796428800, 129060195264000, 10559470521600, 670442572800,
5:         33522128640, 1323241920, 40840800, 960960, 16380, 182, 1]
6:     V = b[13]*A6 + b[11]*A4 + b[9]*A2
7:     temp = A6*V + b[7]*A6 + b[5]*A4 + b[3]*A2 + b[1]*identity
8:     U = A*temp
9:     temp = b[12]*A6 + b[10]*A4 + b[8]*A2
10:    V = A6*temp + b[6]*A6 + b[4]*A4 + b[2]*A2 + b[0]*identity
11:    return U, V
12: end procedure
```

Algorithm 11 Normest of Multiple Matrices

```
1: procedure NORMEST(A_1, A_2, ..., A_k)
2:     l1norm = ||A_1, A_2, ..., A_k||_{l_1}   ▷ Although Reference [24] uses an estimate,
3:     return l1norm                            ▷ the exact is used here
4: end procedure
```

Algorithm 12 Normest of Matrix Power

```
1: procedure NORMEST(A, m)
2:     l1norm = ||A^m||                         ▷ Although Reference [24] uses an estimate,
3:     return l1norm                            ▷ the exact is used here
4: end procedure
```

Algorithm 13 ell

```
1: procedure ELL(A, m)
2:     p = 2*m + 1
3:     c = 2p!/((2p)!(2p + 1)!)                 ▷ Leading coefficient
4:     u = 2^{-53}                              ▷ Unit round off for IEE double
5:     α = c*normest(|A|, p)/|A|_{l_1}
6:     value = ⌈log_2(α/u)/(2*m)⌉
7:     return max(value, 0)
8: end procedure
```

Algorithm 14 Padé Method 1

```
1: procedure padeMethod1(A, t)
2:     α = 0                                   ▷ Number of times to square the matrix
3:     A = A*t
4:     norm = ||A||_1                          ▷ Compute the l_1 norm of the matrix
5:     A2 = A^2
6:     if norm < 1.495585217958292e-002 then
7:         U, V = pade3(A, A2)                 ▷ Compute terms to build pade order 3
8:     else if norm < 2.539398330063230e-001 then
9:         A4 = A2*A2
10:        U, V = pade5(A, A2, A4)             ▷ Compute terms to build pade order 5
11:    else if norm < 9.504178996162932e-001 then
12:        A4 = A2*A2
13:        A6 = A4*A2
14:        U, V = pade7(A, A2, A4, A6)         ▷ Compute terms to build pade order 7
15:    else if norm < 2.097847961257068 then
16:        A4 = A2*A2
17:        A6 = A4*A2
18:        A8 = A6*A2
19:        U, V = pade9(A, A2, A4, A6, A8)     ▷ Compute terms to build pade order 9
20:    else
21:        maxnorm = 5.371920351148152
22:        α = max(0, ⌈(log_2(norm/maxnorm)⌉)  ▷ Calculate number of squarings
23:        A = A/2^α                           ▷ Scale the matrix
24:        A2 = A*A
25:        A4 = A2*A2
26:        A6 = A4*A2
27:        U, V = pade13(A, A2, A4, A6)        ▷ Compute terms to build pade order 13
28:    end if
29:    denominator = -U + V                    ▷ Build the denominator
30:    numerator = U + V                       ▷ Build the numerator
31:    LUSolver.analyzePattern(denominator)    ▷ Analyze sparsity patern
32:    LUSolver.factorize(denominator)         ▷ Compute LU factorization
33:    R = LUSolver.solve(numerator)           ▷ Solve for the matrix exponential
34:    for k=1,2,3...α do                      ▷ Unscale the matrix
35:        R = R*R
36:    end for
37:    return R
38: end procedure
```

Algorithm 15 Padé Method 2

1: **procedure** PADEMETHOD2($\boldsymbol{A}$, t)
2: $\alpha = 0$ ▷ Number of times to square the matrix
3: A = $\boldsymbol{A}$*t
4: norm = $||\mathrm{A}||_{l_1}$ ▷ Compute the l_1 norm of the matrix
5: A2 = A^2
6: d6 = normest(A2,3)$^{1/6}$, η_1 = max(normest(A2,3)$^{1/4}$, d6)
7: **if** $\eta_1 <$ 1.495585217958292e-002 **and** ell(A,3) = 0 **then** ▷ Try Padé 3
8: U, V = pade3(A, A2)
9: Evaluate R using lines 29-33 of Algorithm Padé Method 1
10: **return** R
11: **end if**
12: A4 = A2*A2
13: d4 = $||\mathrm{A4}||_{l_1}^{1/4}$, η_2 = max(d4, d6)
14: **if** $\eta_2 <$ 2.539398330063230e-001 **and** ell(A,5) = 0 **then** ▷ Try Padé 5
15: U, V = pade5(A, A2, A4)
16: Evaluate R using lines 29-33 of Algorithm Padé Method 1
17: **return** R
18: **end if**
19: A6 = A4*A2
20: d6 = $||\mathrm{A6}||_{l_1}^{1/6}$, d8 = normest(A2,2)$^{1/8}$, η_3 = max($d6, d8$)
21: **if** $\eta_3 <$ 9.504178996162932e-001 **and** ell(A, 7) = 0 **then** ▷ Try Padé 7
22: U, V = pade7(A, A2, A4, A6)
23: Evaluate R using lines 29-33 of Algorithm Padé Method 1
24: **return** R
25: **end if**
26: **if** $\eta_3 <$ 2.097847961257068e+000 **and** ell(A, 9) = 0 **then** ▷ Try Padé 9
27: A8 = A6*A2
28: U, V = pade9(A, A2, A4, A6, A8)
29: Evaluate R using lines 29-33 of Algorithm Padé Method 1
30: **return** R
31: **end if**
32: d10 = normest(A4, A6)$^{1/10}$
33: η_4 = max(d8, d10)
34: $\eta_5 = \min(\eta_3, \eta_4)$
35: $\alpha = \max(\lceil \log_2(\eta_5/4.25) \rceil, 0)$
36: $\alpha = \alpha$+ell(A/s^α) ▷ Compute the number of squarings
37: A = A/s^α, A2 = A2/$s^{2\alpha}$, A4 = A4/$s^{4\alpha}$, A6 = A6/$s^{6\alpha}$
38: U, V = pade13(A, B2, B4, B6) ▷ Compute Padé 13
39: Evaluate R using lines 29-33 of Algorithm Padé Method 1
40: **for** k=1,2,3...α **do** ▷ Unscale the matrix
41: R = R*R
42: **end for**
43: **return** R
44: **end procedure**

A.3 Taylor

Algorithm 16 parameters

procedure PARAMETERS($\boldsymbol{A}$, t, tol)
    $A = \boldsymbol{A}t$
    norm $= ||A||_1$
    $l_{lim} = \frac{4}{n_0}\frac{\theta_{m_{max}}}{m_{max}}p_{max}(p_{max}+3)$
    if norm $\leq l_{lim}$ **then**
        $m_* = \text{argmin}_{1\leq m\leq m_{max}}\ m\lceil ||A||_1/\theta_m\rceil$
        $s = \lceil ||A||_1/\theta_m$
    else
        Cost $= \min[m\lceil \alpha_p(A)/\theta_m\rceil : 2 \leq p \leq p_{max}, \quad p(p-1)-1 \leq m \leq m_{max}]$
        Let m_* to be the smallest m achieving the minimum in Cost.
        $s = \max$ (Cost$/m_*, 1)$
    end if
    return m_*, s
end procedure

Algorithm 17 Taylor

```
1: procedure TAYLOR(A, t, b)
2:     A = A, b = b
3:     μ = trace(A)/n
4:     A = A − μI
5:     [m*, s] = parameters(A, t, tol)
6:     F = b, η = e^{tμ}
7:     for i = 1 : s do
8:         c1 = ||b||∞
9:         for j = 1 : m* do
10:            b = tAb/(sj), c2 = ||b||∞
11:            F = F + B
12:            if c1 + c2 ≤ tol||F||∞ then
13:                break
14:            end if
15:            c1 = c2
16:        end for
17:        F = ηF, b = F
18:    end for
19:    return F
20: end procedure
```

B Case Study Information

This section contains a summary of information used in case studies. Tables 1 and 2 are list of the radio nuclides used as the species for depletion in 2D MSR cases. Tables and contain information for the mass transport models.

Table 1: Isotopes include in small case

^{1}H	^{10}B	^{11}B	^{14}N
^{6}Li	^{7}Li	^{9}Be	^{19}F
^{16}O	^{83}Kr	^{93}Nb	^{90}Zr
^{91}Zr	^{92}Zr	^{94}Zr	^{96}Zr
^{95}Mo	^{99}Tc	^{103}Rh	^{105}Rh
^{106}Ru	^{109}Ag	^{126}Sn	^{135}I
^{131}Xe	^{135}Xe	^{133}Cs	^{134}Cs
^{135}Cs	^{137}Cs	^{143}Pr	^{144}Ce
^{143}Nd	^{145}Nd	^{146}Nd	^{147}Nd
^{147}Pm	^{148}Pm	^{149}Pm	^{148}Nd
^{147}Sm	^{149}Sm	^{150}Sm	^{151}Sm
^{152}Sm	^{151}Eu	^{153}Eu	^{154}Eu
^{155}Eu	^{152}Gd	^{154}Gd	^{155}Gd
^{156}Gd	^{157}Gd	^{158}Gd	^{160}Gd
^{234}U	^{235}U	^{236}U	^{238}U
^{237}Np	^{238}Pu	^{239}Pu	^{240}Pu
^{241}Pu	^{242}Pu	^{241}Am	^{242}Am
^{243}Am	^{242}Cm	^{243}Cm	^{244}Cm

* When gas sparging or wall deposition models are used isotopes colored red or blue indicate that addition species are added for gas sparing and wall deposition respectfully. Total 72 or 86

Table 2: Additional isotopes added for medium case

^{93}Zr	^{95}Zr	^{95}Nb	^{97}Mo
^{98}Mo	^{99}Mo	^{100}Mo	^{101}Ru
^{102}Ru	^{103}Ru	^{104}Ru	^{105}Pd
^{107}Pd	^{108}Pd	^{113}Cd	^{115}In
^{127}I	^{129}I	^{133}Xe	^{139}La
^{140}Ba	^{141}Ce	^{142}Ce	^{143}Ce
^{141}Pr	^{144}Nd	^{153}Sm	^{156}Eu
^{242m}Am			

* 29 additional isotopes in addition to the small case. When gas sparging or wall deposition models are used isotopes colored red or blue indicate that addition species are added for gas sparing and wall deposition respectfully. Total 101 or 128

C Addition Results

C.1 Progression Problem 2

Table 3: Convergence Rate for Diffusion-Dominated Problem 2 with Absolute Error

Solver	Cells	E_∞ Rate	E_1 Rate	E_2 Rate	E_∞ Error	E_1 Error	E_2 Error
CRAM	10	-	-	-	7.47e-04	4.78e-04	1.68e-04
-	20	2.00	2.00	2.50	1.87e-04	1.19e-04	2.97e-05
-	40	2.00	2.00	2.50	4.69e-05	2.99e-05	5.24e-06
-	80	2.00	2.00	2.50	1.17e-05	7.46e-06	9.27e-07
-	160	2.00	2.00	2.50	2.93e-06	1.87e-06	1.64e-07
-	320	2.00	2.00	2.50	7.33e-07	4.67e-07	2.90e-08
Parabolic	10	-	-	-	7.47e-04	4.78e-04	1.68e-04
-	20	2.00	2.00	2.50	1.87e-04	1.19e-04	2.97e-05
-	40	2.00	2.00	2.50	4.69e-05	2.99e-05	5.24e-06
-	80	2.00	2.00	2.50	1.17e-05	7.46e-06	9.27e-07
-	160	2.00	2.00	2.50	2.93e-06	1.87e-06	1.64e-07
-	320	2.00	2.00	2.50	7.33e-07	4.67e-07	2.90e-08
Hyperbolic	10	-	-	-	7.47e-04	4.78e-04	1.68e-04
-	20	2.00	2.00	2.50	1.87e-04	1.19e-04	2.97e-05
-	40	2.00	2.00	2.50	4.69e-05	2.99e-05	5.24e-06
-	80	2.00	2.00	2.50	1.17e-05	7.46e-06	9.27e-07
-	160	2.00	2.00	2.50	2.93e-06	1.87e-06	1.64e-07
-	320	2.00	2.00	2.50	7.33e-07	4.67e-07	2.90e-08
Pade-method1	10	-	-	-	7.47e-04	4.78e-04	1.68e-04
-	20	2.00	2.00	2.50	1.87e-04	1.19e-04	2.97e-05
-	40	2.00	2.00	2.50	4.69e-05	2.99e-05	5.24e-06
-	80	2.00	2.00	2.50	1.17e-05	7.46e-06	9.27e-07
-	160	2.00	2.00	2.50	2.93e-06	1.87e-06	1.64e-07
-	320	2.00	2.00	2.50	7.33e-07	4.66e-07	2.90e-08
Pade-method2	10	-	-	-	7.47e-04	4.78e-04	1.68e-04
-	20	2.00	2.00	2.50	1.87e-04	1.19e-04	2.97e-05
-	40	2.00	2.00	2.50	4.69e-05	2.99e-05	5.24e-06
-	80	2.00	2.00	2.50	1.17e-05	7.46e-06	9.27e-07
-	160	2.00	2.00	2.50	2.93e-06	1.87e-06	1.64e-07
-	320	2.00	2.00	2.50	7.33e-07	4.66e-07	2.90e-08
Taylor	10	-	-	-	7.47e-04	4.78e-04	1.68e-04
-	20	2.00	2.00	2.50	1.87e-04	1.19e-04	2.97e-05
-	40	2.00	2.00	2.50	4.69e-05	2.99e-05	5.24e-06
-	80	2.00	2.00	2.50	1.17e-05	7.46e-06	9.27e-07
-	160	2.00	2.00	2.50	2.93e-06	1.87e-06	1.64e-07
-	320	2.00	2.00	2.50	7.33e-07	4.66e-07	2.90e-08

Table 4: Convergence Rate for Reaction-Dominated Problem 2 with Absolute Error

Solver	Cells	E_∞ Rate	E_1 Rate	E_2 Rate	E_∞ Error	E_1 Error	E_2 Error
CRAM	10	-	-	-	8.90e-04	5.69e-04	2.00e-04
-	20	2.00	2.00	2.50	2.23e-04	1.42e-04	3.53e-05
-	40	2.00	2.00	2.50	5.58e-05	3.55e-05	6.24e-06
-	80	2.00	2.00	2.50	1.40e-05	8.88e-06	1.10e-06
-	160	2.00	2.00	2.50	3.49e-06	2.22e-06	1.95e-07
-	320	2.00	2.00	2.50	8.72e-07	5.55e-07	3.45e-08
Parabolic	10	-	-	-	8.90e-04	5.69e-04	2.00e-04
-	20	2.00	2.00	2.50	2.23e-04	1.42e-04	3.53e-05
-	40	2.00	2.00	2.50	5.58e-05	3.55e-05	6.24e-06
-	80	2.00	2.00	2.50	1.40e-05	8.88e-06	1.10e-06
-	160	2.00	2.00	2.50	3.49e-06	2.22e-06	1.95e-07
-	320	2.00	2.00	2.50	8.72e-07	5.55e-07	3.45e-08
Hyperbolic	10	-	-	-	8.90e-04	5.69e-04	2.00e-04
-	20	2.00	2.00	2.50	2.23e-04	1.42e-04	3.53e-05
-	40	2.00	2.00	2.50	5.58e-05	3.55e-05	6.24e-06
-	80	2.00	2.00	2.50	1.40e-05	8.88e-06	1.10e-06
-	160	2.00	2.00	2.50	3.49e-06	2.22e-06	1.95e-07
-	320	2.00	2.00	2.50	8.72e-07	5.55e-07	3.45e-08
Pade-method1	10	-	-	-	8.90e-04	5.69e-04	2.00e-04
-	20	2.00	2.00	2.50	2.23e-04	1.42e-04	3.53e-05
-	40	2.00	2.00	2.50	5.58e-05	3.55e-05	6.24e-06
-	80	2.00	2.00	2.50	1.40e-05	8.88e-06	1.10e-06
-	160	2.00	2.00	2.50	3.49e-06	2.22e-06	1.95e-07
-	320	2.00	2.00	2.50	8.72e-07	5.55e-07	3.45e-08
Pade-method2	10	-	-	-	8.90e-04	5.69e-04	2.00e-04
-	20	2.00	2.00	2.50	2.23e-04	1.42e-04	3.53e-05
-	40	2.00	2.00	2.50	5.58e-05	3.55e-05	6.24e-06
-	80	2.00	2.00	2.50	1.40e-05	8.88e-06	1.10e-06
-	160	2.00	2.00	2.50	3.49e-06	2.22e-06	1.95e-07
-	320	2.00	2.00	2.50	8.72e-07	5.55e-07	3.45e-08
Taylor	10	-	-	-	8.90e-04	5.69e-04	2.00e-04
-	20	2.00	2.00	2.50	2.23e-04	1.42e-04	3.53e-05
-	40	2.00	2.00	2.50	5.58e-05	3.55e-05	6.24e-06
-	80	2.00	2.00	2.50	1.40e-05	8.88e-06	1.10e-06
-	160	2.00	2.00	2.50	3.49e-06	2.22e-06	1.95e-07
-	320	2.00	2.00	2.50	8.72e-07	5.55e-07	3.45e-08

Table 5: Convergence Rate for Stiff Reaction-Dominated Problem 2 with Absolute Error

Solver	Cells	E_∞ Rate	E_1 Rate	E_2 Rate	E_∞ Error	E_1 Error	E_2 Error
CRAM	10	-	-	-	3.76e-02	2.40e-02	8.43e-03
-	20	2.00	2.00	2.50	9.42e-03	6.00e-03	1.49e-03
-	40	2.00	2.00	2.50	2.36e-03	1.50e-03	2.63e-04
-	80	2.00	2.00	2.50	5.89e-04	3.75e-04	4.66e-05
-	160	2.00	2.00	2.50	1.47e-04	9.38e-05	8.23e-06
-	320	2.00	2.00	2.50	3.68e-05	2.34e-05	1.46e-06
Parabolic	10	-	-	-	3.76e-02	2.40e-02	8.43e-03
-	20	2.00	2.00	2.50	9.42e-03	6.00e-03	1.49e-03
-	40	2.00	2.00	2.50	2.36e-03	1.50e-03	2.63e-04
-	80	2.00	2.00	2.50	5.89e-04	3.75e-04	4.66e-05
-	160	2.00	2.00	2.50	1.47e-04	9.38e-05	8.23e-06
-	320	2.00	2.00	2.50	3.68e-05	2.34e-05	1.46e-06
Hyperbolic	10	-	-	-	3.76e-02	2.40e-02	8.43e-03
-	20	2.00	2.00	2.50	9.42e-03	6.00e-03	1.49e-03
-	40	2.00	2.00	2.50	2.36e-03	1.50e-03	2.63e-04
-	80	2.00	2.00	2.50	5.89e-04	3.75e-04	4.66e-05
-	160	2.00	2.00	2.50	1.47e-04	9.38e-05	8.23e-06
-	320	2.00	2.00	2.50	3.68e-05	2.34e-05	1.46e-06
Pade-method1	10	-	-	-	3.76e-02	2.40e-02	8.43e-03
-	20	2.00	2.00	2.50	9.42e-03	6.00e-03	1.49e-03
-	40	2.00	2.00	2.50	2.36e-03	1.50e-03	2.63e-04
-	80	2.00	2.00	2.50	5.89e-04	3.75e-04	4.66e-05
-	160	2.00	2.00	2.50	1.47e-04	9.38e-05	8.23e-06
-	320	2.00	2.00	2.50	3.68e-05	2.34e-05	1.46e-06
Pade-method2	10	-	-	-	3.76e-02	2.40e-02	8.43e-03
-	20	2.00	2.00	2.50	9.42e-03	6.00e-03	1.49e-03
-	40	2.00	2.00	2.50	2.36e-03	1.50e-03	2.63e-04
-	80	2.00	2.00	2.50	5.89e-04	3.75e-04	4.66e-05
-	160	2.00	2.00	2.50	1.47e-04	9.38e-05	8.23e-06
-	320	2.00	2.00	2.50	3.68e-05	2.34e-05	1.46e-06
Taylor	10	-	-	-	3.76e-02	2.40e-02	8.43e-03
-	20	2.00	2.00	2.50	9.42e-03	6.00e-03	1.49e-03
-	40	2.00	2.00	2.50	2.36e-03	1.50e-03	2.63e-04
-	80	2.00	2.00	2.50	5.89e-04	3.75e-04	4.66e-05
-	160	2.00	2.00	2.50	1.47e-04	9.38e-05	8.23e-06
-	320	2.00	2.00	2.50	3.68e-05	2.34e-05	1.46e-06

C.2 Progression Problem 4

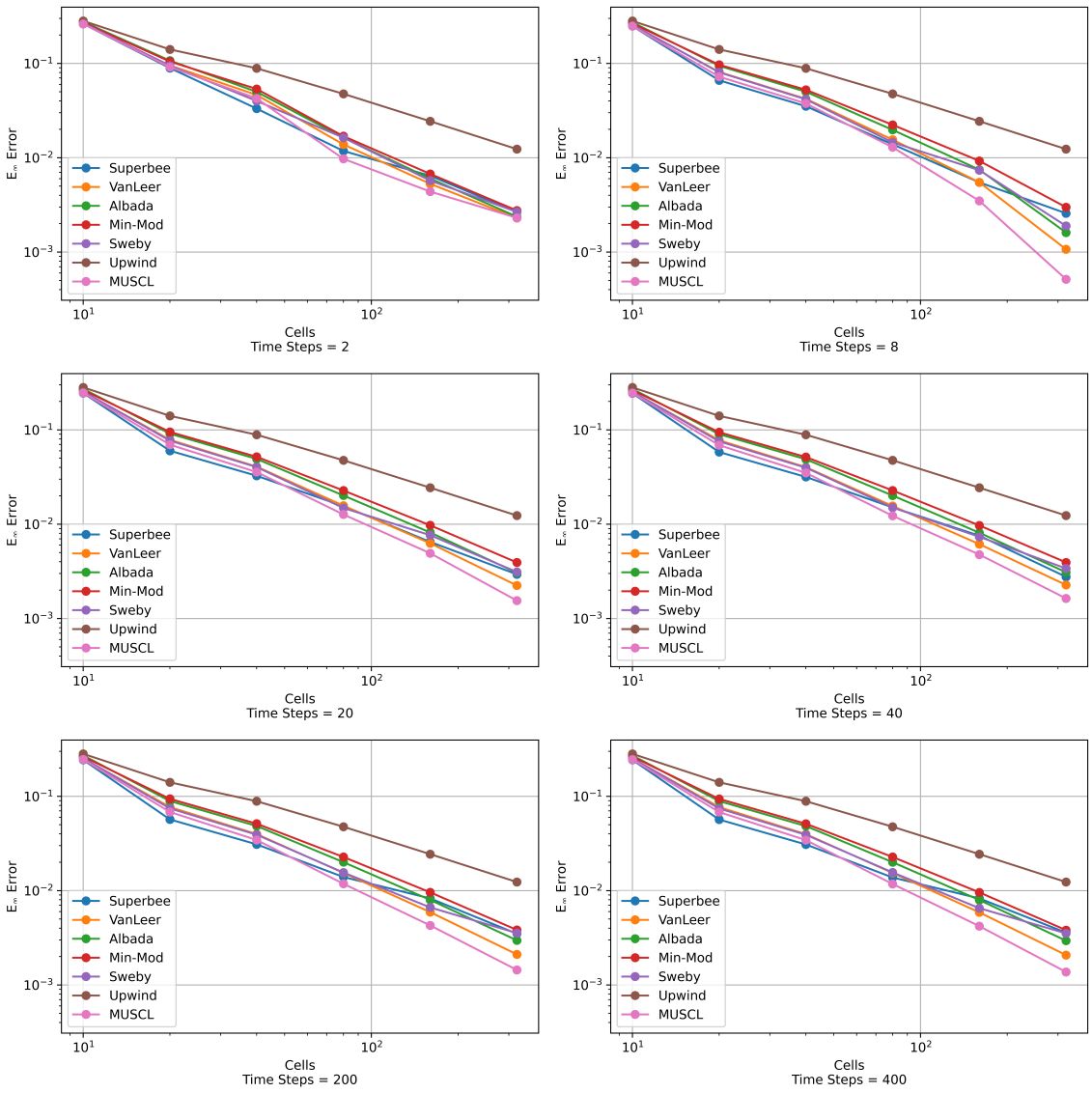

Figure 1: Problem 4 absolute E_∞ error with spatial discretization at various time steps

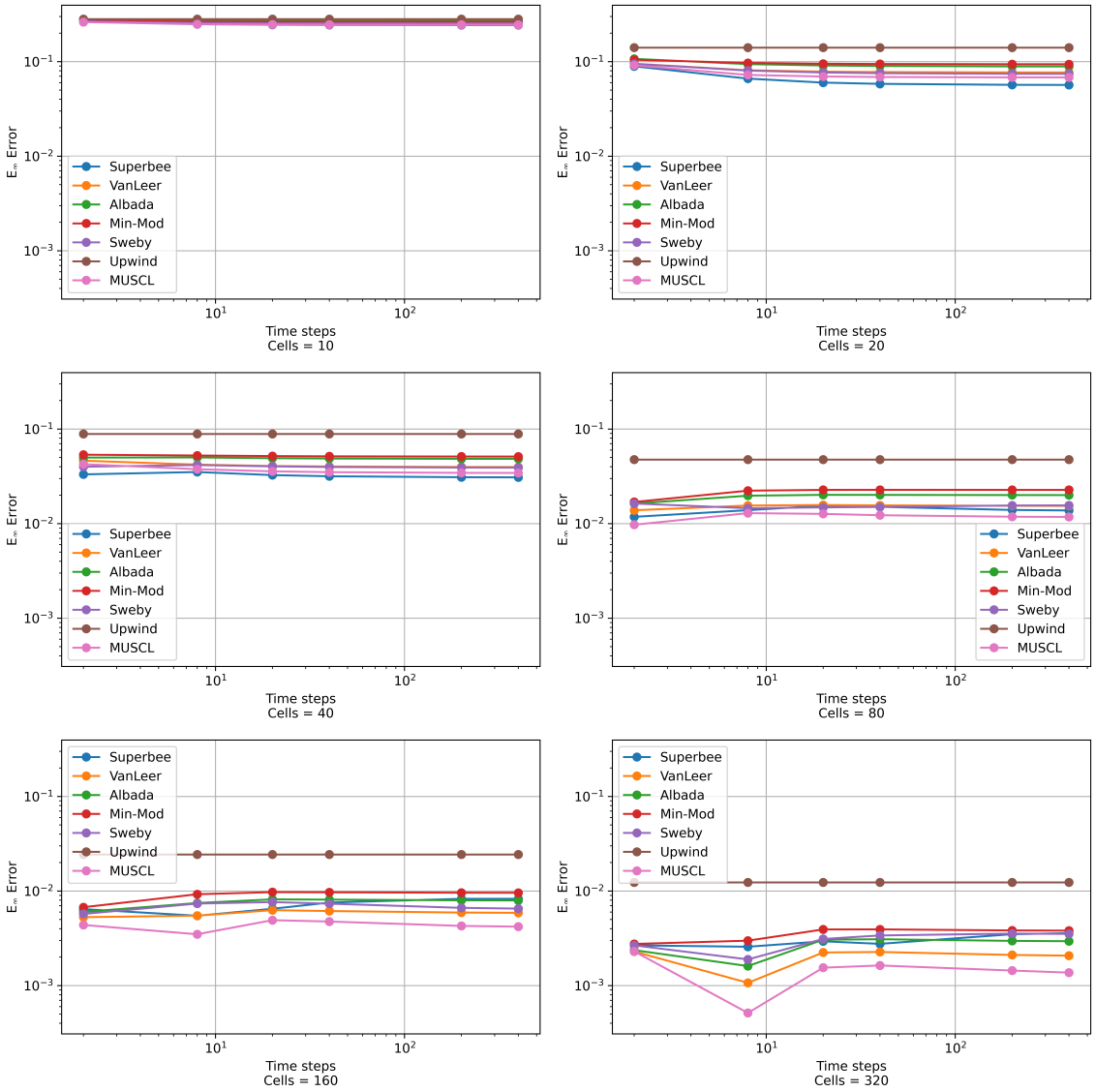

Figure 2: Problem 4 absolute E_∞ error with temporal discretization at various number of spatial cells

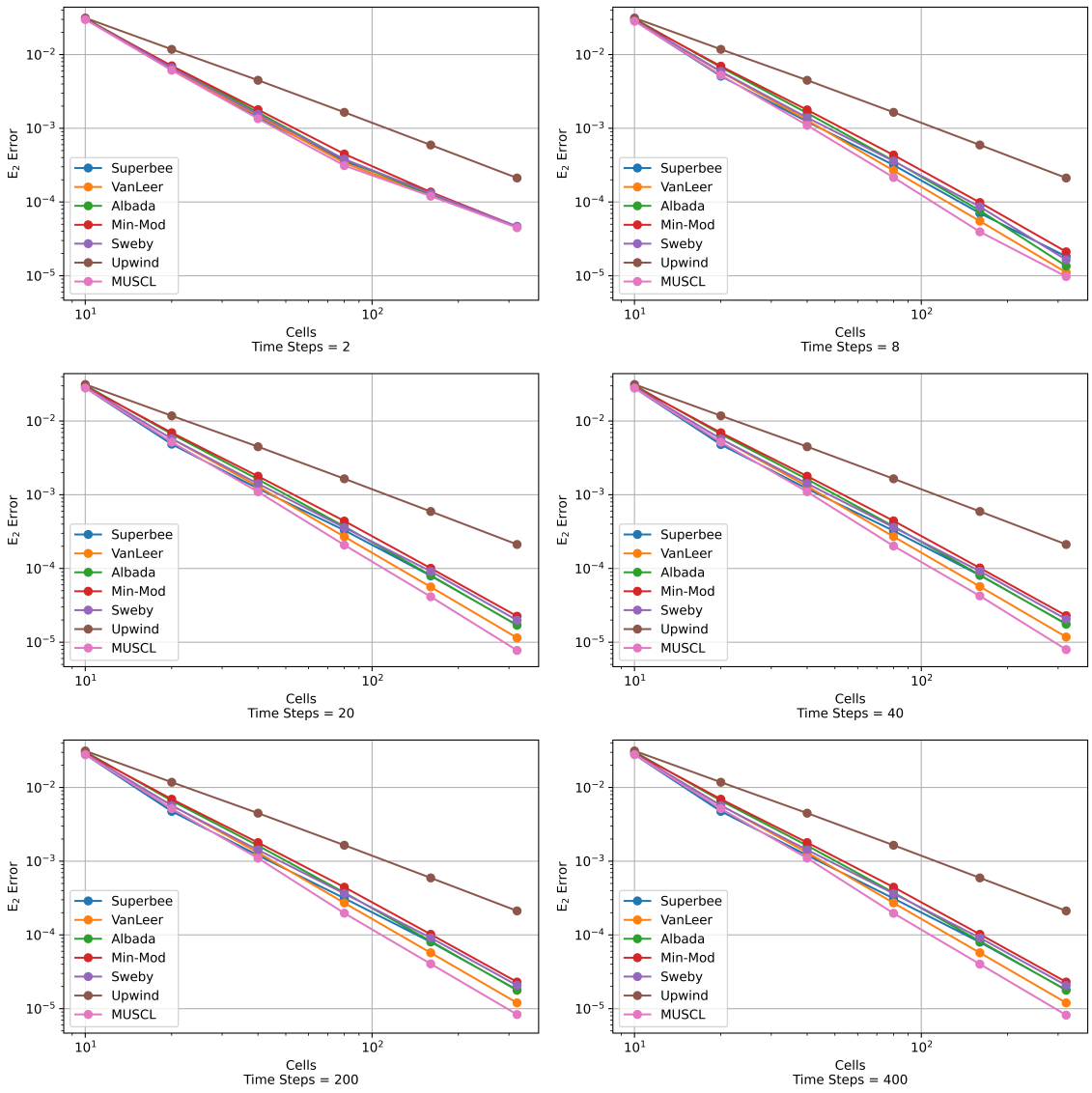

Figure 3: Problem 4 absolute E_2 error with spatial discretization at various time steps

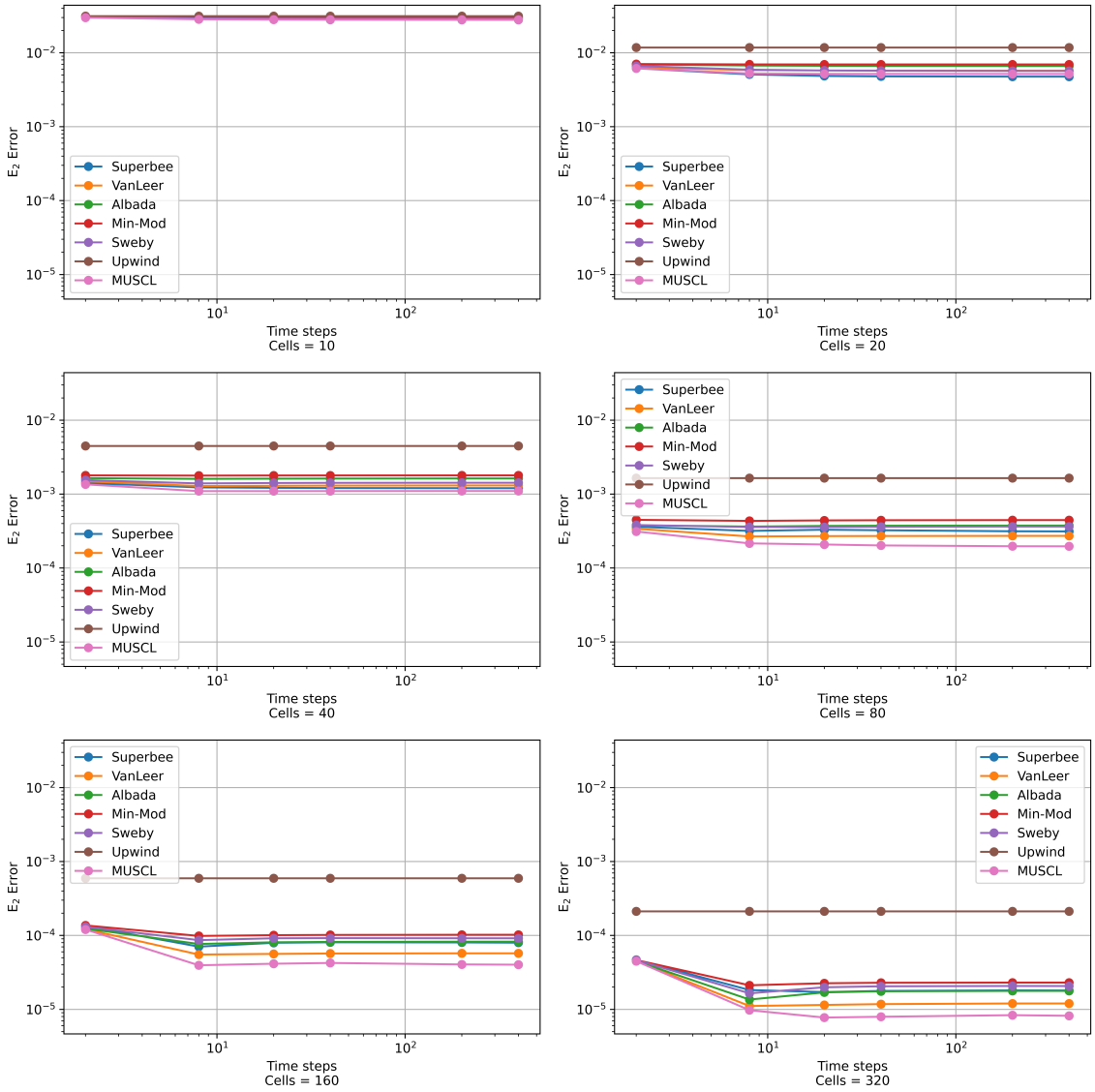

Figure 4: Problem 4 absolute E_2 error with temporal discretization at various number of spatial cells

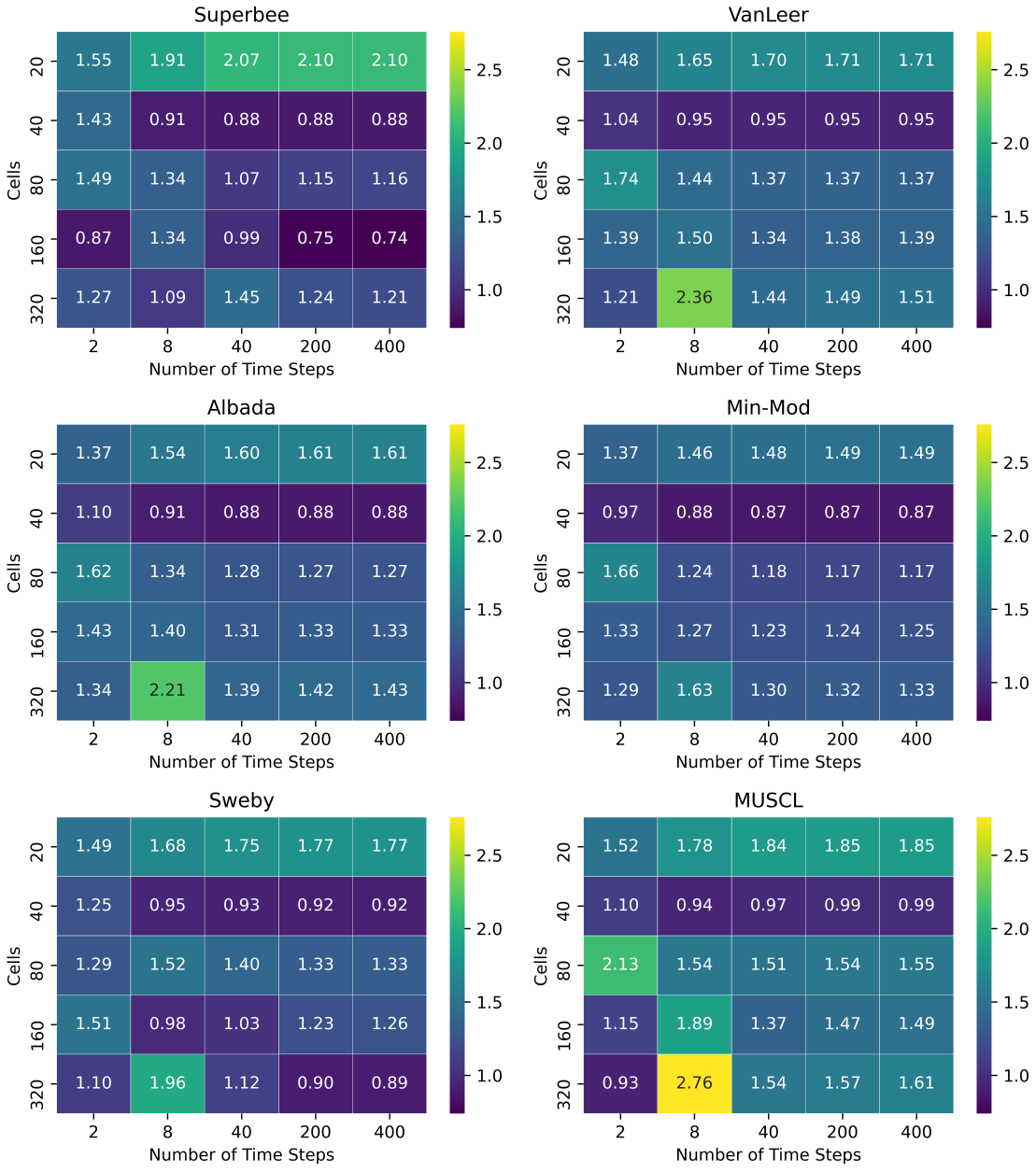

Figure 5: Problem 4 flux limiter E_∞ convergence rate using absolute error

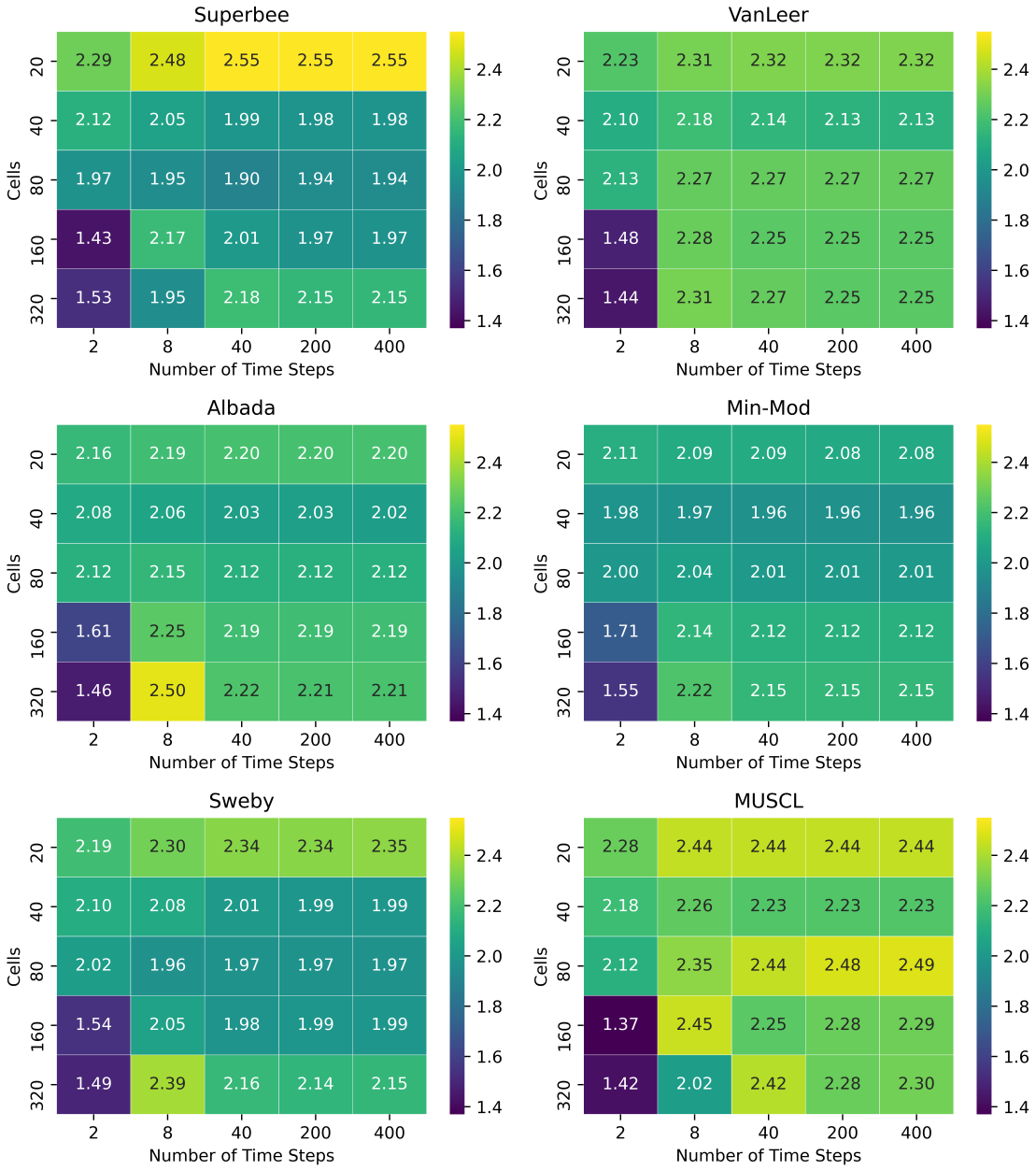

Figure 6: Problem 4 flux limiter E_2 convergence rate using absolute error

C.3 Progression Problem 5

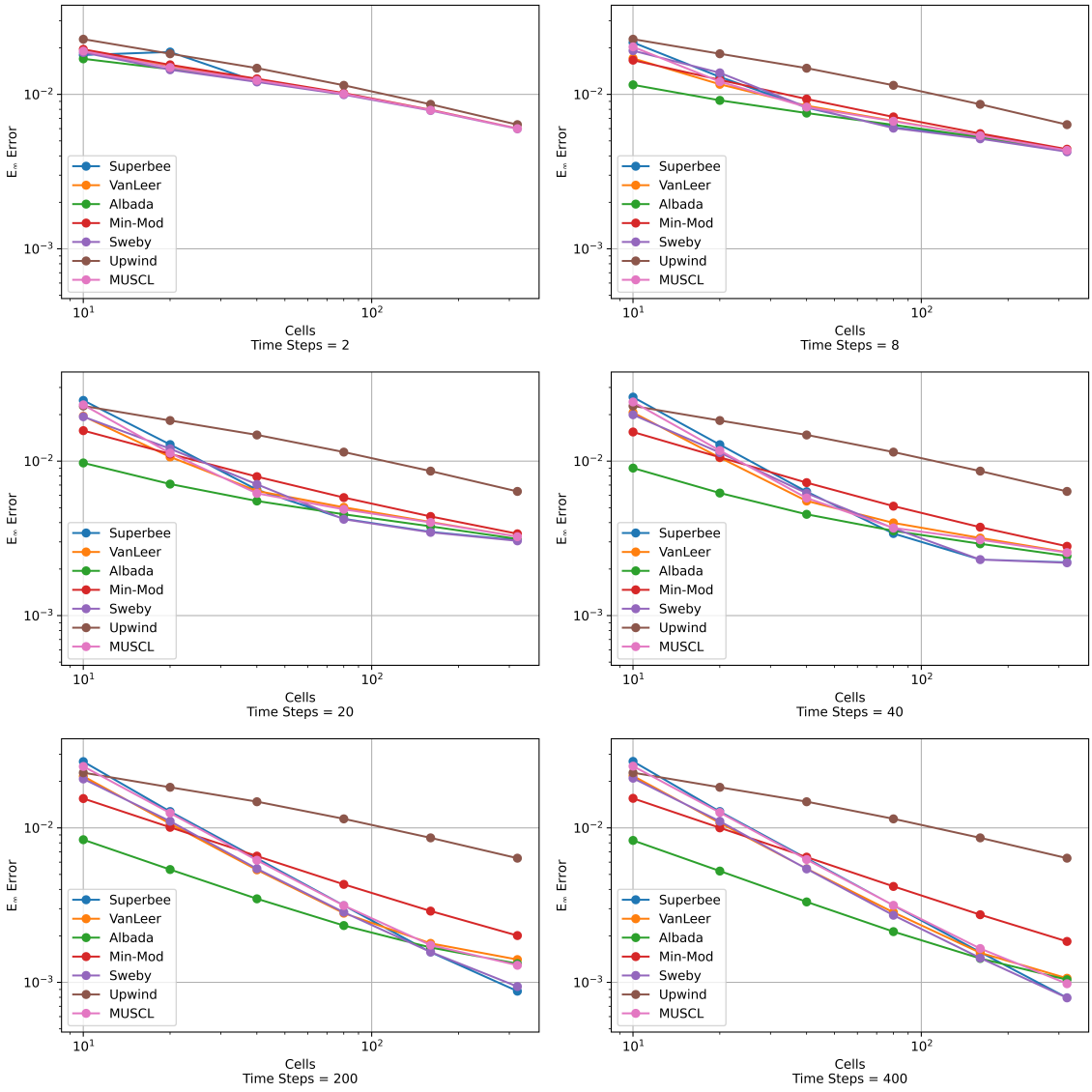

Figure 7: Problem 5 absolute E_∞ error with spatial discretization at various time steps

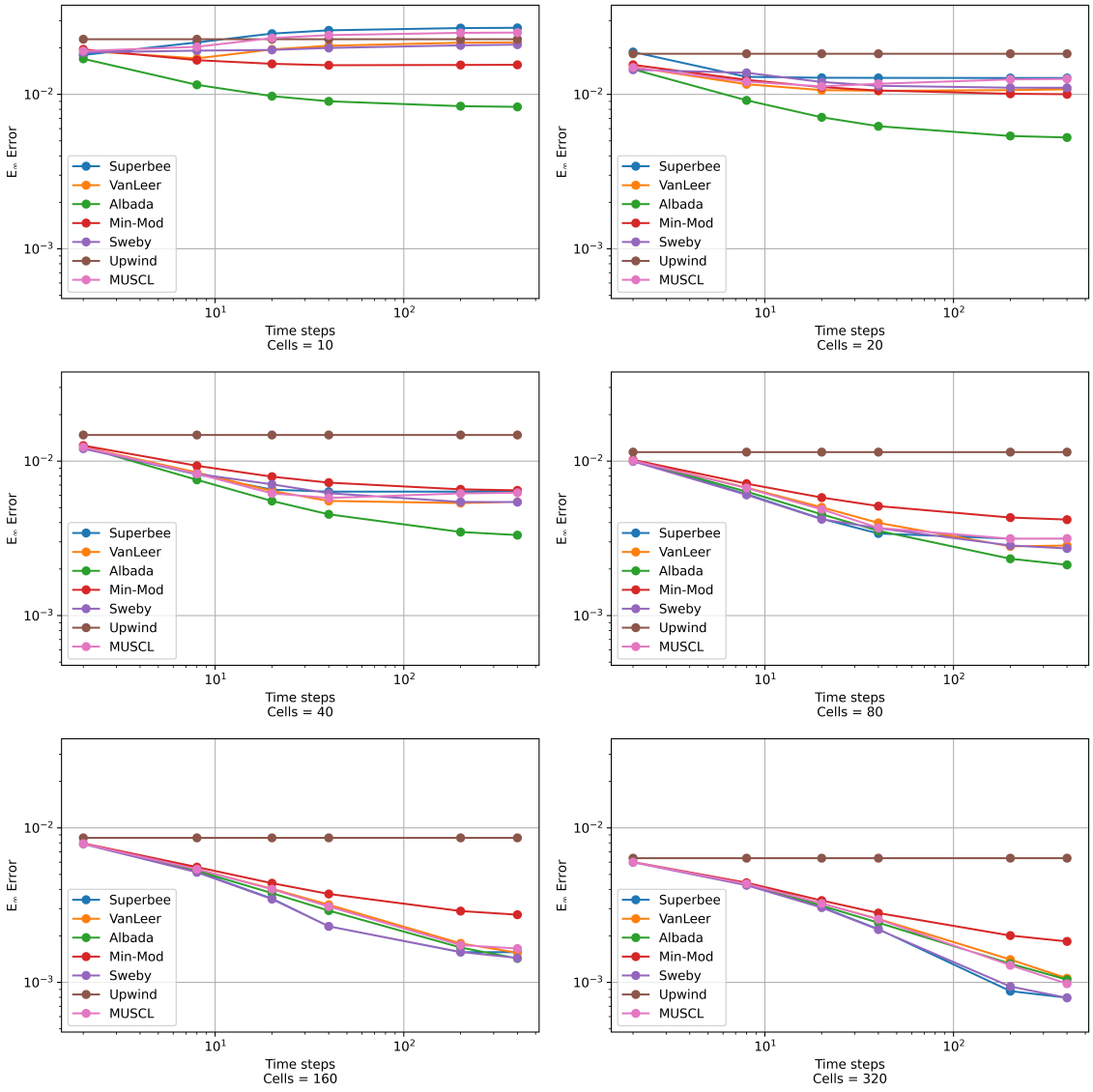

Figure 8: Problem 5 absolute E_∞ error with temporal discretization at various number of spatial cells

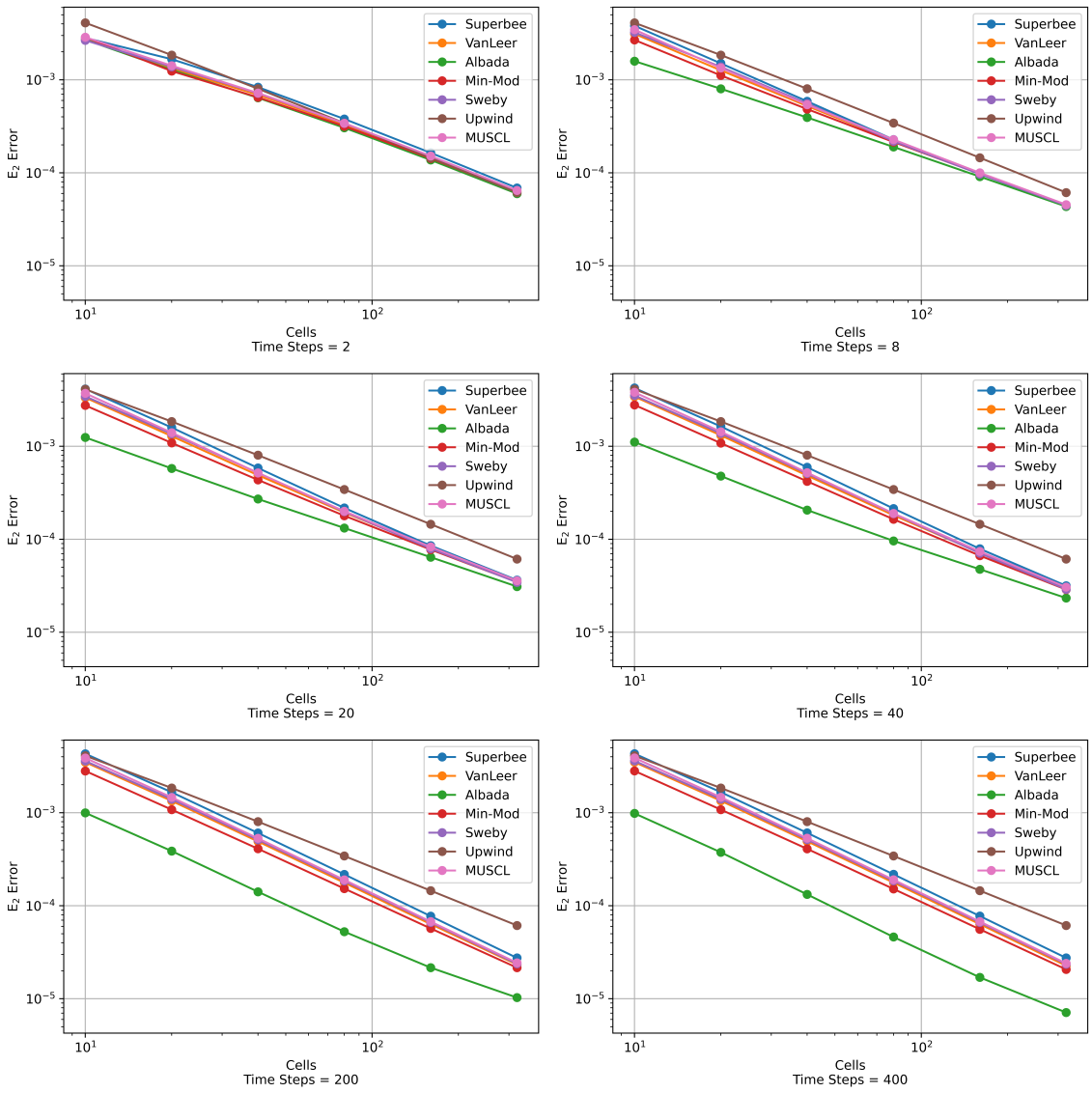

Figure 9: Problem 5 absolute E_2 error with spatial discretization at various time steps

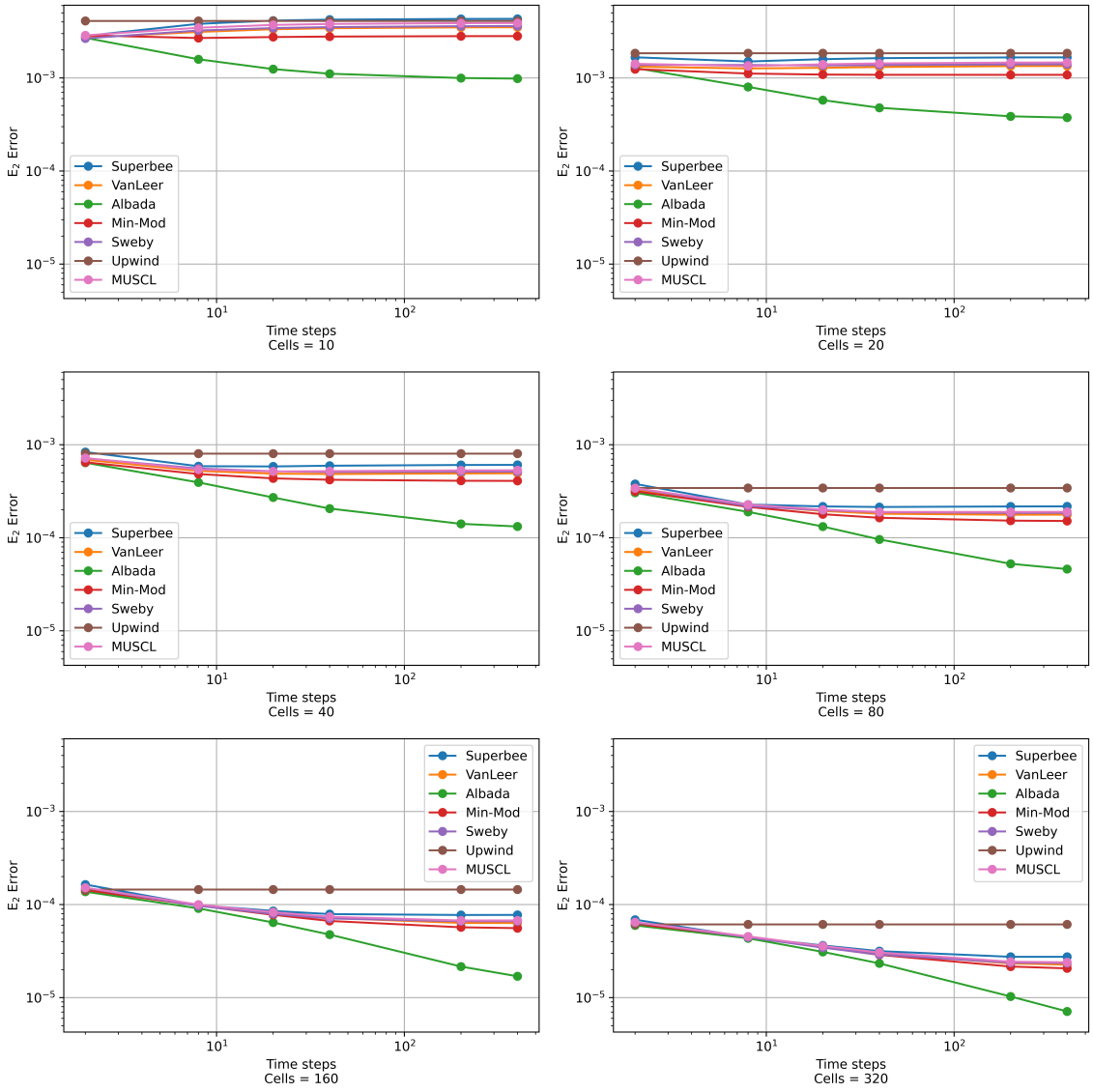

Figure 10: Problem 5 absolute E_2 error with temporal discretization at various number of spatial cells

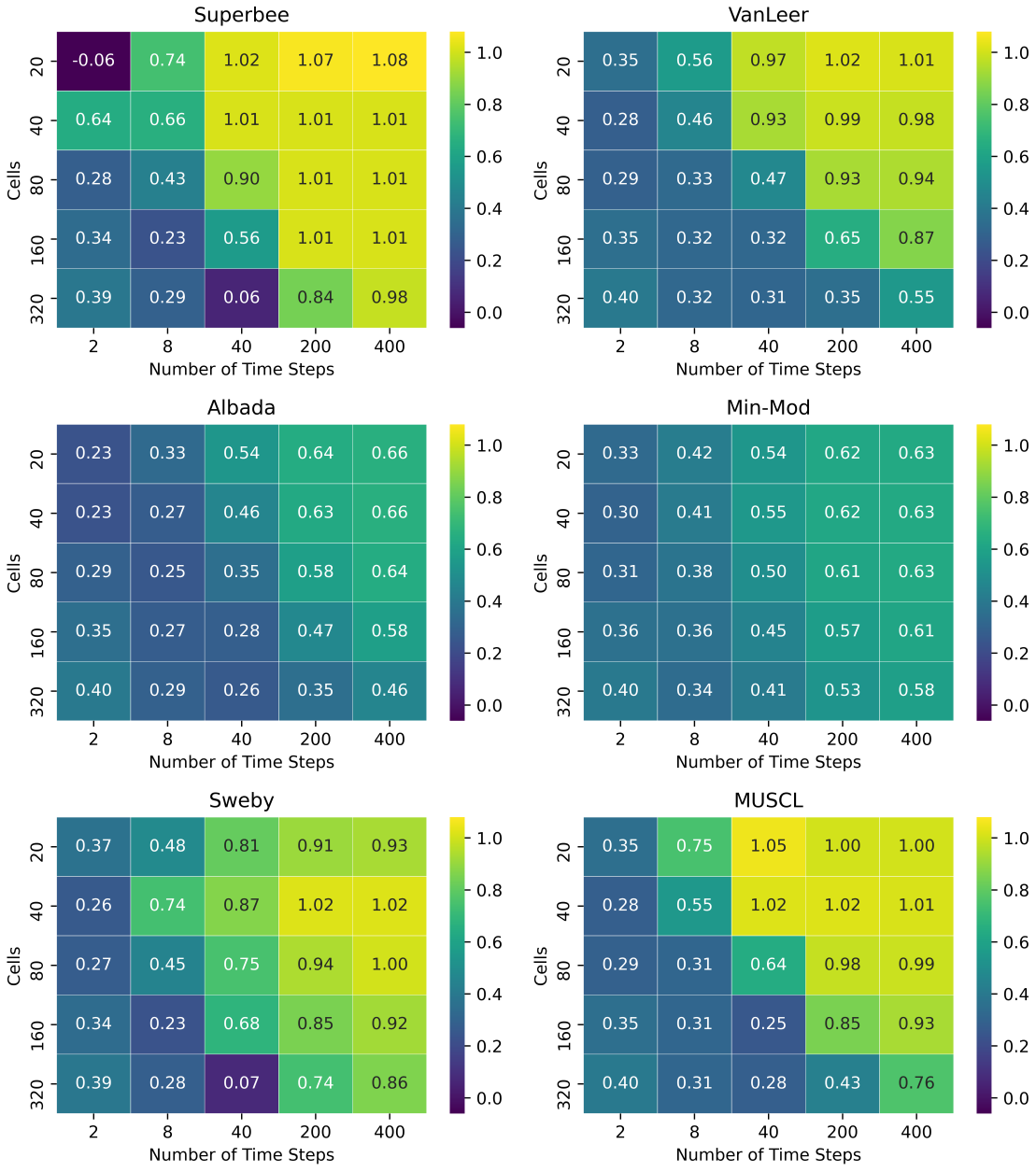

Figure 11: Problem 5 flux limiter E_∞ convergence rate using absolute error

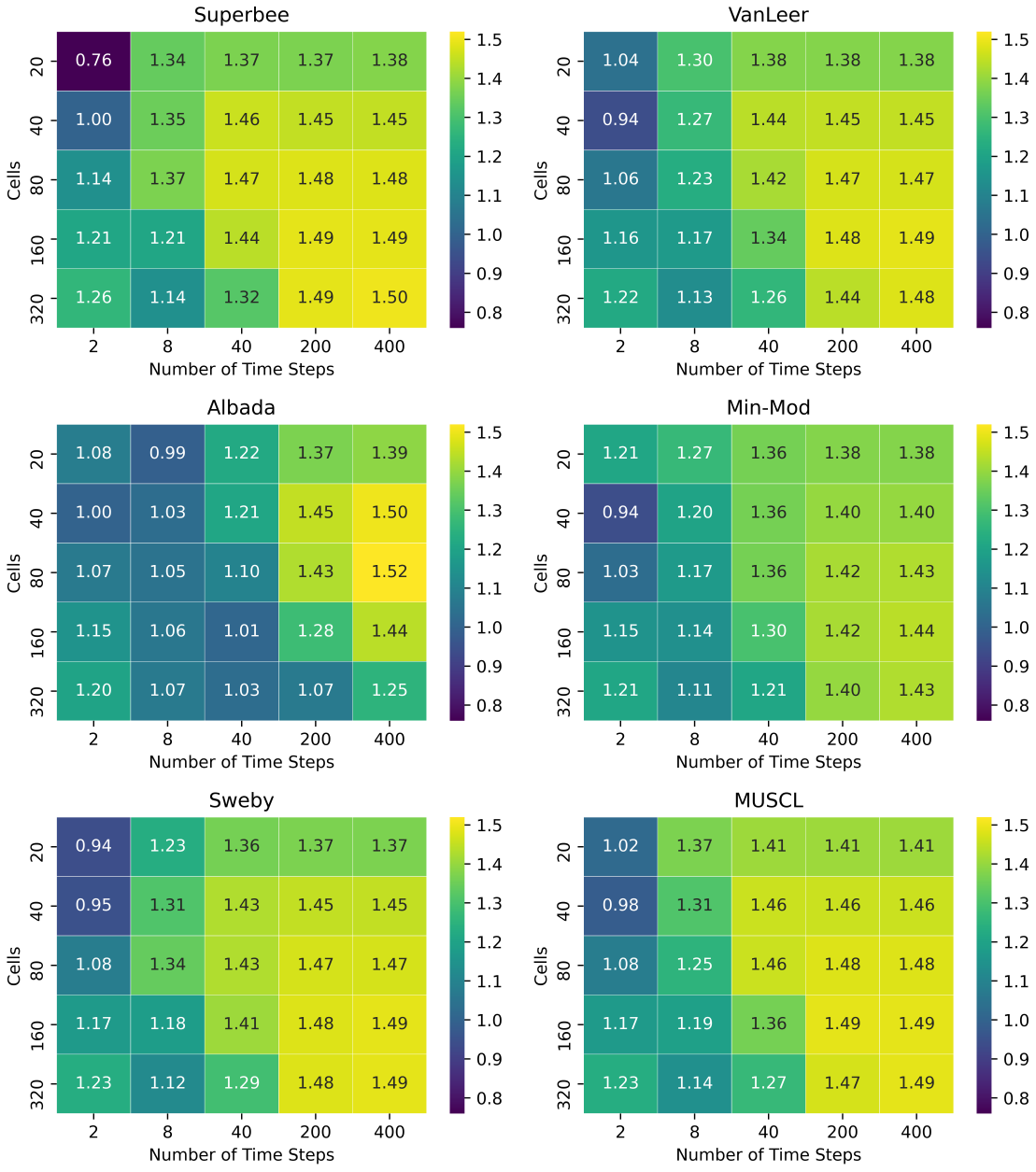

Figure 12: Problem 5 flux limiter E_2 convergence rate using absolute error

Vita

Zack Taylor was born 1992 in Memphis Tennessee to parents Bob and Francis Taylor. At a young age he and his family moved to the gulf coast of Mississippi where he grew up playing soccer and swimming in the bayou. In high school he was part of a foreign exchange program with a partner school in Germany. In 2016 Zack graduated from Mississippi State University with a Bachelors of Science degree in chemical engineering. After graduating he was accepted to The University of Tennessee, Knoxville, in the nuclear engineering program where he is pursuing his PhD. During his graduate research he received a Masters of Science degree in chemical engineering. Many of the research areas he is interested in includes; computational and non-equilibrium thermodynamics, mass transport and numerical analysis.