DISSERTATION

PARALLEL TIME INTEGRATION METHODS FOR HYPERBOLIC PDES IN APPLICATION

Submitted by

Jerett Charles Cherry

Department of Mathematics

In partial fulfillment of the requirements

For the Degree of Doctor of Philosophy

Colorado State University

Fort Collins, Colorado

Spring 2026

Doctoral Committee:

Advisor: Wolfgang Bangerth

Olivier Pinaud
Yongcheng Zhou
Bret Windom

ABSTRACT

PARALLEL TIME INTEGRATION METHODS FOR HYPERBOLIC PDES IN APPLICATION

Parallel in Time (PinT) integration methods are iterative methods which seek to increase computational parallelism in numerical solutions to initial value problems by solving future time states concurrently. PinT methods have been successfully applied to problems with sufficient dissipation, but struggle to converge quickly for problems without much dissipation or that are dominated by convection or hyperbolic in nature. In these cases, PinT is not expected to work well. Because of this, PinT methods have not been not favored by many computational scientists, though we may still desire faster time to solution in convective problems.

In this dissertation, we push the usefulness of PinT methods to include hyperbolic PDEs. To do so, we present a modification for PinT methods which computes a time averaged state rather than a pointwise in time quantity. Time averaged quantities like drag and lift of an airfoil are useful to engineers in application, so we ask how well PinT can be applied in computing them.

The calculation of time averages has not seen any research in the PinT community. We hope that this research expands the uses for PinT methods by exhibiting its usefulness in computing useful quantities for PDEs beyond those of parabolic behavior.

ACKNOWLEDGEMENTS

I would like to thank Dr. Wolfgang Bangerth for his support and detailed feedback during my research. Your insights have made me a better mathematician, software designer, and person.

Marc Fehling, Sam Scheuerman, Fernando Herrera, and Wasim Munshi, deserve recognition for their excellent editing and for reading this dissertation many times. Thank you for allowing me to discuss research ideas with you before I have figured out what the ideas are.

I have a deep sense of gratitude for my small cohort of graduate peers who were struggling in their own ways alongside me and pushed me not only to work hard but also to enjoy my time as a graduate student.

Without my family and partner I would be hopeless. Thank you all for putting up with my long years of study. Carly, thank you for keeping me on track and for your ceaseless support. J1, J2, J4, J5, J6 you all made me who I am and I appreciate you more than you can know. Obi, thank you for being excited to see me every day, no matter what.

DEDICATION

I would like to dedicate this thesis to my family and to my friends.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

In this preliminary examination paper, we seek to solve Initial Value Problems (IVPs) arising in scientific contexts of the form

$$\frac{dU(t)}{dt} = f(U(t), t) \quad U(0) = U_0, \tag{1.1}$$

where U is possibly a vector coming from discretizing a Partial Differential Equation (PDE). Such problems frequently arise when doing numerical science.

Given an initial condition, one might want to predict how a physical system evolves over time from $t = 0$ to $t = T$. Typically, any numerical method for the time evolution of U is given by selecting a *discrete* set of N time points at intervals of $\Delta t = \frac{T}{N-1}$:

$$t_0 = 0,\, t_1 = 1 \cdot \Delta t,\, \ldots,\, t_i = i \cdot \Delta t,\, \ldots,\, t_{N-1} = (N-1) \cdot \Delta t.$$

and constructing solutions

$$U_n := U(t_n)$$

at these time points via a sequence of operations

$$U_{n+1} = F(U(t_n), t_n, t_{n+1}) \;\; n = 0, 1, \ldots, N-2, \tag{1.2}$$

where $F(U(t_n), t_n, t_{n+1})$ integrates the solution $U(t_n)$ to the next solution $U(t_{n+1})$. And then an approximation for the continuous solution $U(t)$ is given by the vector of $[U_0, U_1, \ldots, U_{N-1}]$.

Definition 1 (Notation for Integrating Operator)**.** *We denote any method for taking an approximate solution U_n at a time-discretized point t_n and producing the next approximated solution at t_{n+1} as*

$$F(U_n, t_n, t_{n+1}).$$

A different method for the same could be denoted with another symbol like $G(U_n, t_n, t_{n+1})$.

Without going into excessive detail, this F operator can be thought of as a discretized version of $f(U, t)$ from (1.1). With this in mind, we can see that, for most numerical methods, constructing a discretized approximation to $U(t)$ is equivalent to solving a *sequence* of $N - 1$ problems (1.4)

$$U_0 = U(t_0) \text{ known} \tag{1.3}$$

$$U_1 = F(U_0, t_0, t_1), \quad \ldots, \quad U_{N-1} = F(U_{N-2}, t_{N-2}, t_{N-1}). \tag{1.4}$$

Once this is done, we can approximate the time evolution of U as the vector of U_i's. If we know the operation F, solving this sequence is no problem. One starts at $U(t_0)$ and constructs $U(t_1)$ from that. Any part of the evolution of U is known in this way. Almost any software used for scientific calculations solving problem (1.1) follows this paradigm and are too various to cite.

In some sense this matches well with our intuition about such time dependent systems–they are sequential. For example, what I do now effects the immediate future, which would then also affect the future's future. This is how we imagine the physical dimension of time: a long sequences of *nexts*, depending on all the *thens* and the *now*.

This kind of intuition has brought numerical science far, and people have spent much time writing code that uses the framework of (1.4). However, in an age of complex scientific questions, a sequential solution of IVPs could 1) require $1 \ll N$ for accuracy of our approximation $U(t_i)$ and/or 2) have sufficiently complex formulations of F which are computationally expensive. For example, both of these situations happen in simulations of plasma physics, where physical wavespeeds dictate that time steps are small, and complicated physics routines may need to be calculated to, for instance, model neutron transport.

Suppose each evaluation $F(U(t_n), t_n, t_{n+1})$ takes Q_s seconds on a supercomputer. Then, to integrate our initial condition to some final time T with $1 \ll N$ time points required for accuracy, one would need to wait $Q_s \cdot (N-1)$ seconds for the program to finish running. This is a potentially very long time to wait if the problem is complex enough for Q_s and N both large. One is out of luck if one desires a solution quickly.

In the past, people could count on Moore's law of transistor density doubling every 2 years to decrease how long their calculations took to run on the most up to date computers [1]. The joke was that if your code was too slow, waiting two years meant it could work. As Moore's law slows, we can no longer count on improved processing speeds, but rather must optimize existing code, develop new algorithms, and explore exotic numerical methods. Indeed, even during Moore's era, people argued that speedup in scientific computing was equally driven by algorithmic improvements as hardware advances [1], but today it may be all that is left for improved speeds.

To alleviate the cost in clock time, the idea this thesis aims to follow is to ***parallelize the time direction***. The class of generally iterative methods to do this are called **Parallel in Time (PinT)** methods, and an in depth review of these methods can be found in [2, 3, 4]. Their most basic idea, however, is simple: run computations on future time points while you run computations on current time points. To help us understand how this strange sounding idea would work ideally, let us introduce some terminology we will use throughout this document.

PinT methods break a time domain down into what we will call "time bricks" (or we could call "time slabs"), and perform parallel calculations on these bricks.

Definition 2. *A **Time Brick** is an interval of time $[t_n t_{n+1}]$, often a subinterval of a larger interval of time $[0, T]$. We denote the nth time brick of a split interval, $[t_n, t_{n+1}]$, as B_n. Each interval may contain M of the original N time points. Therefore we write $t_{n,i} \in B_n$ for $i = 0, \ldots, M-1$, the M points contained in B_n. With this notation, the start of B_n is written $t_{n,0}$ and the end $t_{n,M-1}$. Note that $t_n = t_{n,0}$ and $t_{n+1,0} = t_{n,M-1}$.*

In an ideal PinT method, we would like to break (1.4) into K bricks where each brick requires the solution of an IVP specific to that brick. Writing the mth brick like

$$B_m := [m\frac{T}{K}, (m+1)\frac{T}{K}],$$

we can write $[0, T] = \cup_{m=0}^{K-1} B_m$. Note that dividing N points into K bricks gives each brick $\frac{N+K-1}{K}$ time points since each brick shares a point with the surrounding bricks (Figure 1.1).

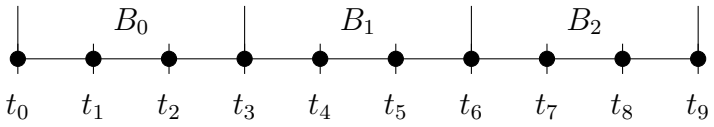


Figure 1.1: Dividing $N = 10$ time points between $K = 3$ bricks. Here, each brick has $\frac{10+3-1}{3} = 4$ time points, and needs to integrate over $\frac{N-1}{K} = 3$ steps.

Projecting the sequence from (1.4) onto our collection of bricks, we then get a set of sub-sequential IVPs, each with $\frac{N-1}{K}$ steps in the sequence:

$$U(t_{0,i+1}) = F(U(t_{0,i}), t_{0,i}, t_{0,i+1}) \qquad U(t_{0,0}) = U_0 \tag{1.5}$$

$$U(t_{1,i+1}) = F(U(t_{1,i}), t_{1,i}, t_{1,i+1}) \qquad U(t_{1,0}) = U(t_{0,\frac{N}{K}-1}) \tag{1.6}$$

$$\vdots \tag{1.7}$$

$$U(t_{K-1,i+1}) = F(U(t_{K-1,i}), t_{K-1,i}, t_{K-1,i+1}) \qquad U(t_{K-1,0}) = U(t_{K-2,\frac{N}{K}-1}) \tag{1.8}$$

where $i = 0, \ldots, \frac{N-1}{K}$ for each problem. Then we can now solve the K sub-problems (1.5), (1.6), (1.7), (1.8) concurrently on different computers. If this is possible, then one would only have to wait for the duration of one of the sub-problems to finish, which takes $Q_s \cdot (\frac{N-1}{K}) < Q_s \cdot (N-1)$ seconds, giving a perfect parallel speedup of K. This leads us to the **ideal PinT method** (Algorithm 1).

Algorithm 1 Ideal PinT Method

1: Split simulation interval $[0, T]$ into K time bricks so that the execution time $Q_s \cdot (\frac{N-1}{K})$ of each brick is small.
2: Pick a sequential solver $F(*, t_{m,i}, t_{m,i+1})$ for the set of problems (1.5)-(1.8).
3: In parallel, apply F to each time brick B_m for $m = 0, \ldots, K-1$.

Using Algorithm 1 would be great, were it not for the fact that the overall problem's sequential nature has not been broken at all. The IVP on B_m has an initial condition coming from the final condition on B_{m-1}. So Algorithm 1 will not do; we cannot solve B_m until B_{m-1} is done!

To break causality, we need to transform Algorithm 1 into an iterative one. Because we do not want to wait for B_0 before executing B_1, we must provide initial conditions for each of the K bricks. Since we cannot know the right conditions, we guess. Then, knowing our guesses are wrong, we produce corrections to those guesses in a mathematically sound way. Once we have done this on all bricks, we iterate and run on all bricks again, where we can produce still more corrections. Obviously, repeating computations on our bricks eats into the gains from the ideal Algorithm 1, but this method makes parallel time computations possible. We term this type of PinT method a **Predictor-Corrector** method. All PinT algorithms which fall into a predictor-corrector category follow the format of Algorithm 2 which we call the τ-*correction* algorithm of PinT, and only differ in how the correction is calculated.

Algorithm 2 τ-Correction

1: For each of K bricks B_m, provide $U^0(t_{m,0})$ initial guess.
2: **for** iteration $k = 0, \ldots, I$ **do**
3: Pick a sequential solver $F(*, t_{m,i}, t_{m,i+1})$ and solve (1.7) for each brick.
4: In Parallel, compute a correction $\tau_{m,(N-1)/K}$.
5: Correct the guesses at the end of each brick via

$$U^{k+1}_{m,(N-1)/K} \leftarrow U^k_{m,(N-1)/K} + \tau_{m,(N-1)/K}.$$

6: **end for**

In analogy, consider a team of people whose task it is to track the vector position of a ball $U(t)$ as it rolls down a hill. The sequential option to complete this task is to let the ball roll down the

hill and track its position. The PinT option would require the team to spread out along the hill, one at the top, the next some way down the hill, the next further still, and so on. Each person would carry their own ball, and be responsible for placing the ball in a spot on the hill where they think the ball at the top will be at their position. Once everyone is in place, they let their ball roll down to the next person's location. This piecewise trajectory, denoted $U^0(t)$, is probably not very accurate. However, this is done fast, since the ball did not have to roll very far.

Next iteration, each person looks at where the uphill ball had landed in comparison to where they predicted it would. From this information they can correct their initial position, and since they can look uphill to see how others have done their corrections, they can use this information to better inform their corrected starting point. Then everyone rolls their ball again and the process repeats for a piecewise trajectory $U^1(t)$.

It is not hard to imagine this might result in the the exact trajectory $U(t)$ after a while. For instance, after one iteration, the person who is first downhill will know exactly where the ball should have been placed, so the trajectory on iteration 2 will be exact down to the next person. Eventually, each person knows exactly where their ball should be placed. However, depending on the ease in predicting $U(t)$ is for each person's starting point, this process might be exact in only a few iterations–perhaps much earlier than simply waiting.

Some factors that affect the effectiveness of this process to speed up the task of tracking $U(t)$ might be found in the smoothness of the slope (a smoother slope could be easier to predict, while a rocky one harder), or the speed in which the ball rolls (a slower ball means that we have more time to gain during the correction process), or the number of people we have available to help (more people means each ball has to roll less distance, and predictions can come faster). In mathematical terms, a smoother slope might equate to a parabolic PDE, while a rougher one a hyperbolic PDE, and the speed at which the ball rolls is the cost of our sequential solution of (1.4). If the problem is nice enough, but costly enough. We can probably track the ball faster with this process.

This thesis is concerned with exploring the limitations of these methods to realistic problems, where the question is not always 'nice enough'.

1.1 Goals

In this dissertation, we present a research direction in PinT Methods for realistic applications– in the solution and analysis of hyperbolic PDEs. PinT integration methods have existed since the 1960s (see [3]) at least, and are still being investigated today. PinT integration works exceptionally well for some types of systems, but not universally out of the box. Overall, PinT methods have been successfully applied to integration of parabolic problems [5, 6, 7], but not very successfully to hyperbolic PDEs (see the discussion in Chapter 4). Because of this gap in cases, overwhelmingly PinT have not been applied to realistic questions. Though many realistic scientific problems are not hyperbolic in nature, at least an equal number are, which limits the scope of these methods' application.

With this in mind, this dissertation seeks to explore the possibilities for using PinT methods to the solution of IVPs resulting from hyperbolic PDEs with a focus on understanding the limitations one must grapple with in order to do so. To this end, we propose to leverage a combination of ideas like PinT with projection [8], multilevel PinT methods with spatial coarsening [9], selection of the right underlying time integration method, and the novel contribution of averaging across the time domain to help stabilize and improve the slow convergence these methods exhibit for hyperbolic PDEs.

In particular, we explore the calculation of quantities like

$$J(U,t) = \begin{cases} \text{drag, lift, friction} \\ \text{energy cascade, vorticity} \\ \text{etc.} \end{cases} \tag{1.9}$$

which depend on $U(t)$, the solution of a discretized PDE whose time evolution fits into the framework of (1.1). Such quantities may converge at the same rate as $U(t)$ does in a PinT method, but a statistical interpretation of $J(U,t)$, like

$$\int_0^T J(U,t)dt = \begin{cases} \text{average drag, lift, or friction} \\ \text{slope of energy cascade, average vorticity} \\ \text{etc.} \end{cases} \tag{1.10}$$

might converge at a higher rate.

This dissertation explores just how this type of averaging affects convergence with the hope that this opens avenues for hyperbolic PDEs to be solved with PinT methods more effectively. We have a goal of illustrating these capabilities on a collection of 'realistic' problems like calculating drag on an airfoil at low Reynolds numbers.

In Chapter 2, we outline the history of PinT methods, including a fixed point perspective. Then the most popular PinT methods are presented in Chapter 3, before finishing with a comparison of each method. Next, Chapter 4 emphasizes that PinT methods do not work well for hyperbolic problems (or those with finite speed of propagation) and presents a numerical example of this issue. Chapter 5 highlights that averaging in time for hyperbolic PinT is a feasible use for PinT methods, and present a loose justification with the same numerical example that this is not a crazy idea. Then, Chapter 6 discusses the algorithm and special modifications to PinT methods used to compute time averaged quantities. After this, Chapter 7 applies the algorithms herein to two test cases: pressure and drag on a cylinder in Mach 3 flow wake, and to an airfoil in landing configuration at Mach 0.23. Finally, Chapter 8 concludes the dissertation with a discussion of results, novel contributions, and future research directions.

Chapter 2

History and Conception of Parallel in Methods

PinT methods have been considered as additional avenues for computational parallelism, especially as processor clock speeds stagnate. In fact, the openings of many papers in the PinT field include just such a discussion (see [10, 11, 12, 7, 13] to name a few) as justification for PinT. As computers have advanced, researchers have found it difficult to utilize all the computational resources available to them. This is critical for recent trends in supercomputing, namely the push for Exascale computing, where experts have looked at various avenues to utilize computing resources to the fullest, including PinT methods [7, 14, 15].

However, the earliest PinT methods originated in the 1960s (before Exascale computing was remotely possible) as interesting methods themselves for solving IVPs. Today they may find usefulness in supercomputing where scientists desire faster time-to-solution than is currently accessible.

2.1 PinT Methods as Space-Time Domain Decompositions

Typical numerical methods for solving the vector form of (1.1), which typically results from spatially discretizing a PDE, involve splitting the vector $U(t_n)$ into smaller sub-vectors, upon which separate processors use the propagator $F(U,t)$ to update this smaller subset of data to create $U(t_{n+1})$.Consider a problem splitting U with g_x global elements (corresponding to all the spatial degrees of freedom), into $l_x \ll g_x$ local elements (corresponding to the spatial degrees of freedom that a given processor owns), as we can imagine from (2.1), where processor 0 is responsible for computing the time history of $U(x_i,t)$ for $i = 1,\ldots,l_x$, only a subset of the global vector, then other subsets of the vector are assigned to other processors. This is known as **Domain Decomposition (DD)**, since the spatial domain is split up, producing a speedup happens because each processor only has to loop over vectors of size $l_x < g_x$, reducing the amount of work each processor has to do to produce the next time point $U(t_{n+1})$.

$$U(\boldsymbol{x},t) \to U(t) = \left[\begin{array}{c} U(x_1,t) \\ \vdots \\ U(x_{l_x},t) \\ \hline \vdots \\ \hline U(x_{g_x-l_x+1},t) \\ \vdots \\ (U(x_{g_x},t) \end{array}\right] \begin{array}{l} \left.\begin{array}{l} \\ \\ \end{array}\right\} \text{Done by Process 1} \\ \vdots \\ \left.\begin{array}{l} \\ \\ \end{array}\right\} \text{Done by Process N} \end{array} \tag{2.1}$$

Often, for F to compute the local update, data from vectors not known to the local processor must be communicated. This type of thinking has been used everywhere in scientific computing, largely with the support of the **Message Passing Interface (MPI)** programming paradigm. This can be done, at the cost of needing to communicate non-local information. Communication (for example using MPI) is not free, however, so minimizing communication is a must. What results is that one cannot just throw extra processors at a spatially discretized PDE and expect a faster compute time. At some point, the problem saturates, and communication costs outweigh the gain from smaller problem size on each processor. On modern supercomputers, if the number of processors saturates in the DD sense, it is possible that a huge amount of computational power is left on the table.

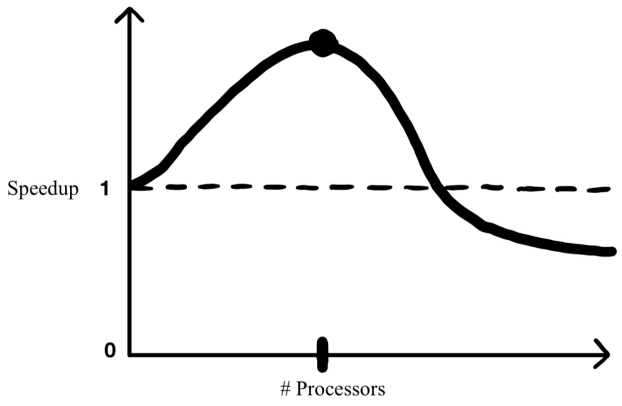


Figure 2.1: At a certain point, DD methods saturate, meaning adding more processors in space actually slows the computation down. If one has access to many more processors beyond this saturation which sit idle, one needs to explore options, like PinT, for increased parallelism.

Parallel in Time integrators solve the saturation issue by increasing parallelization beyond the spatial DD methods by reducing the number of time steps each process must handle. Think of this as a domain decomposition method in *space-time*. We split up the spatial dimension, and throw our extra processors at the time dimension. Thinking about splitting the time domain into multiple ways can help us design PinT methods.

2.1.1 Space-Time Parallel Decompositions

PinT methods differ fundamentally in how they decompose the space-time domain. The defining characteristic is in which direction one slices the space-time domain. The PinT literature has a few different methods: the brick-based predictor-corrector approaches which slice horizontally Figure 2.2, Waveform Relaxation methods which slice vertically Figure 2.4, and fully Space-Time Multigrid methods which do no slicing at all Figure 2.5. Splitting the space-time domain in these different ways forces one to define relaxations on the boundaries of those splits.

Recalling the brick-based, predictor-corrector type algorithm Algorithm 2, we can draw a space-time splitting as splitting the time dimension into horizontal bricks, the starts of which need predictions U^0 and corrections τ (depicted pictorially in Figure 2.2). This is the relaxation that one needs to define for this splitting: how do we start the sequential nature of each brick? This splitting has the added benefit of easily adding regular spatial DD in the space dimension, meaning one can utilize many more processors Figure 2.3. Predictor-Corrector methods all use this style of splitting, and the ease of using codes that already use DD with this type of splitting make them popular.

Contrast predictor corrector splittings with those of **Waveform Relaxation (WR)** methods which split the space-time domain vertically, like Figure 2.4 [16, 17, 18, 19]. This split one should think of as letting local areas in space run concurrently on different processors, at potentially different computational speeds. In WR methods, the speedup is now related to which spatial patch takes the longest to compute. This is a form of 'local time stepping'. With this splitting, one naturally splits the solution vector U into different parts like DD: $U = [U_1, U_2, \ldots, U_N]$, where

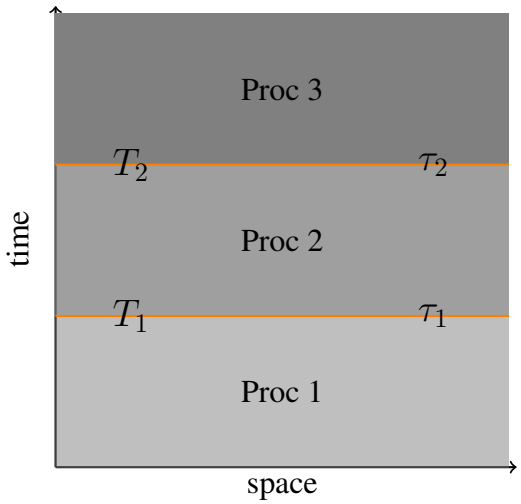


Figure 2.2: Predictor-Corrector PinT algorithms split the domain by breaking the time domain into bricks and doing prediction and correction at the starting point of each brick T_1, T_2 (in orange, τ_1, τ_2), iterating until a sufficient convergence criterion has been met. Here, time integration is done across the spatial domain for the duration of each brick, where this time integration could be done across many processors, like with Message Passing Interface and distributed data structures.

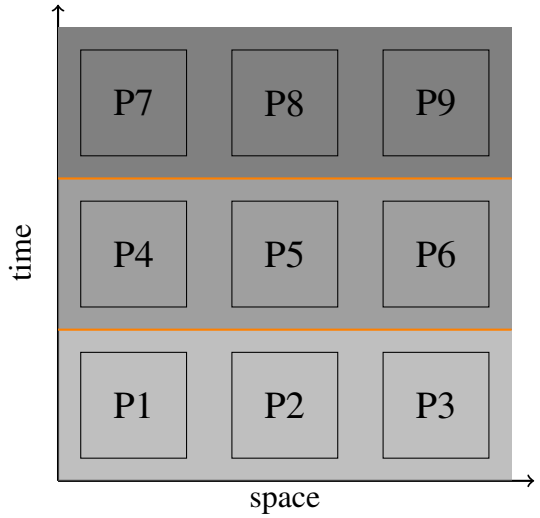


Figure 2.3: For predictor-corrector type spacetime-splitting, we can decompose each brick into many processes which allow for computational parallelization within each brick. This is the typical thing one sees drawn out in papers where MGRIT is defined.

U_i is a vector on process 1. To achieve local time stepping, we introduce artificial boundary conditions on each patch Ω_i. For example, we know that process two has two boundary conditions on U_2 which must match the data of Proc 1 and Proc 2 at x_1, x_2 respectively.:

$$U_2(x_1) = U_1(x_1) \text{ and } U_2(x_2) = U_3(x_1).$$

Clearly, we cannot run these patches independently of each other like we'd like to. Instead, we relax the boundary condition by making the process iterative. For iteration $k+1$, we solve for $U^{k+1} = [U_1^{k+1}, U_2^{k+1}, \ldots, U_N^{k+1}]$ with the boundary conditions given by previous iterations. For our example, this relates to a problem

$$\frac{\partial U_1^{k+1}}{\partial t} = F(U_1^{k+1}) \ , \ U_1^{k+1}(0,t) = U(0,t) \text{ and } U_1^{k+1}(x_1,t) = U^k(x_1,t) \tag{2.2}$$

$$\frac{\partial U_2^{k+1}}{\partial t} = F(U_2^{k+1}) \ , \ U_2^{k+1}(x_1,t) = U_1^k(x_1,t) \text{ and } U_2^{k+1}(x_2,t) = U_3^k(x_2,t) \tag{2.3}$$

$$\frac{\partial U_3^{k+1}}{\partial t} = F(U_3^{k+1}) \ , \ U_3^{k+1}(x_1,t) = U_2^k(x_2,t) \text{ and } U_3^{k+1}(L,t) = U(L,t). \tag{2.4}$$

Here, (2.2) only involves DOFs on Proc 1's patch Ω_1, (2.3) only involves DOFs on Proc 2's patch Ω_2, and (2.4) only involves DOFs on Proc 3's patch Ω_3. Since the k-information is known, these are fully decoupled problems that can run in parallel. We need to iterate to find the proper boundary conditions for each patch, and convergence is achieved when the patch solutions have boundary conditions that match along the interfaces for all t.

Contrasting both the previous methods is **space-time multigrid** methods which perform a multigrid algorithm on all of spacetime at once as one would do in a boundary value problem– but including the temporal direction as well. These are the least common, mainly because they require time to be an intrinsic variable in the data. Common scientific computation codes store only the variables as they vary in space, and time is more of a 'tag' that is attached to the data, rather than living on the datasets that are used for computation. A fully MG algorithm requires

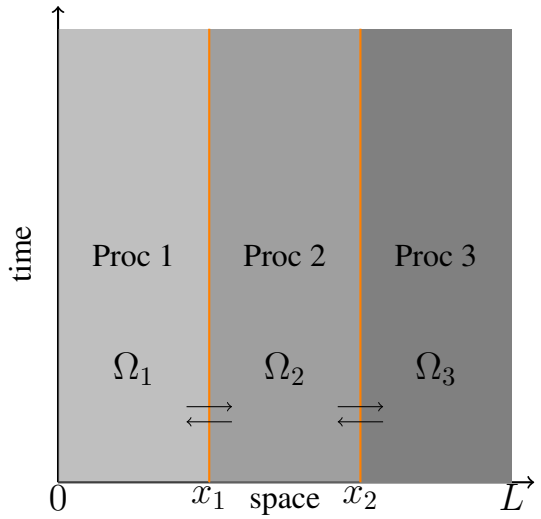


Figure 2.4: Waveform Relaxation methods are iterative PinT algorithms split the domain by introducing artificial spatial boundaries x_1 and x_2 (in orange) with artificial boundary conditions defined from information from previous iterations. The whole time interval is solved on each spatial domain, and is used to define the artificial boundary conditions for the next iteration.

time as a variable, not a tag. The parallelization of these comes entirely from the parallel solver one uses, and is not iterative in the same sense of the other two.

There is some debate about the future of PinT methods and what kinds are most suited for real scientific computing. Some people argue that space-time MG methods can offer the best parallel performance because of the overall performance of multigrid methods [20]. One drawback of this is that most codebases do not think of time as an extra dimension, meaning that to use these methods, one would have to re-write the data-structures to treat time as its own component of the solution state–on top of writing the multigrid framework. Others view the iterative, brick-based, methods as ideal because of their agnosticism to integrators, allowing practitioners to use their favorite integrator that they likely have spent much time developing for their specific problem. The drawback here is that the parallel performance gains can be smaller. Hence one may be forced to optimize their PinT code for one specific problem just to attain any speedup at all. Then, if one were to try another problem, this code may not work.

We take the point of view that a method non-intrusive to the codes that already exist is best: we can use optimized integrators in a plug and play manner, saving us time coding our methods. This

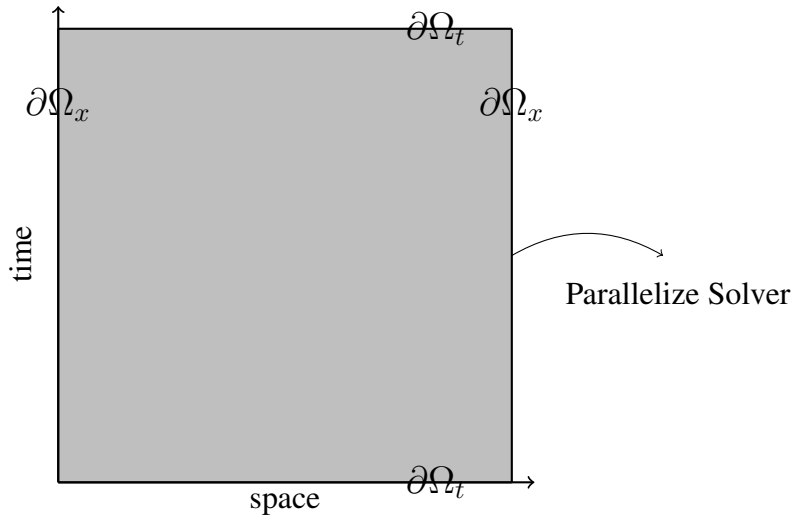


Figure 2.5: For a fully space-time multigrid method, we solve the whole domain akin to a MG method for a boundary value problem, where our boundary includes space and time. The space-time parallelization of this method comes from the parallelization we can get from the MG solver. Time is intrinsic to the data, not a 'tag'.

viewpoint considers the speedup of the algorithm as a function of developer time as well. These methods lend themselves to quick production.

2.1.2 PinT from a Root Finding (All at Once) Perspective

We have seen in the introduction Chapter 1 how we can conceive of certain PinT methods as brick-based iterative prediction-correction algorithms. However, another useful perspective for PinT methods can be found in viewing the sequence of problems (1.4) as a fixed point equation [21]. In this vein we can modify the sequence into a vector equation where we use an iterative root finding algorithm to solve for all the time points at once. Then we have

$$\overline{U} := \begin{bmatrix} U_0 \\ U_1 \\ \vdots \\ U_n \end{bmatrix} = F(\overline{U}) := \begin{bmatrix} U_0 \\ F(U_0, t_0, t_1) \\ \vdots \\ F(U_{N-1}, t_{N-1}, t_N) \end{bmatrix} \tag{2.5}$$

which we can write like a root finding problem

$$\widetilde{F}(\overline{U}) = \overline{U} - F(\overline{U}) = \mathbf{0}. \tag{2.6}$$

So, to solve our time integration, we desire to solve (2.6) for $\overline{U}$. One option is the sequential solve that is universal in integration methods. Another is to try to solve this nonlinear equation with something like a Newton-Krylov iteration. This has the benefit of solving for all the U_n at once, at the cost of an iterative procedure and solving for the inverse of the Jacobian of $\tilde{F}$. Denoting by $\widetilde{F}(\overline{U}) = \overline{U} - F(\overline{U})$ we write the newton iteration as

$$\overline{U}^{k+1} = \overline{U}^{k} - J^{-1}_{\widetilde{F}(\overline{U}^k)} \cdot \widetilde{F}(\overline{U}^k) \tag{2.7}$$

$$J_{\widetilde{F}(\overline{U}^k)} \cdot \left(\overline{U}^{k+1} - \overline{U}^{k}\right) = -\widetilde{F}(\overline{U}^k) \tag{2.8}$$

Carefully unraveling the effect of the Jacobian on the change $\overline{U}^{k+1} - \overline{U}^{k}$ provides us with a new set of equations for $n \in [1, \ldots, N-1]$ which are solved each newton iteration:

$$U_0^{k+1} = U_0^k \tag{2.9}$$

$$U_{n+1}^{k+1} = F(U_n^k) + \frac{\partial F(U_n^k)}{\partial U_n^k}(U_n^{k+1} - U_n^k). \tag{2.10}$$

Here, the superscript k is the newton iteration, and the subscript n indicates the time point. Now we have a way to compute all of the time points at once. This rearrangement seems unproductive since it is not parallel in time. We still need to compute a sequential set of equations–the value of U_{n+1}^{k+1} depends on knowing the value of U_n^{k+1}, the solution on the same iteration at the previous brick. However, it is possible to choose an approximation to $\frac{\partial F(U_n^k)}{\partial U_n^k}$ and rearrange so that we break this sequential dependence. For example, we could make the assumption that the differential term of (2.10) is well approximated by a difference of some "cheaper" method $G(*, t_n, t_{n+1}) := G(*)$

$$\frac{\partial F(U_n^k)}{\partial U_n^k}(U_n^{k+1} - U_n^k) \approx G(U_n^{k+1}) - G(U_n^k). \tag{2.11}$$

This is just what the Parareal algorithm, discussed in the methods section, does [22]. If we substitute (2.11) into (2.10) we can re-write the update as

$$U_0^{k+1} = U_0^k \tag{2.12}$$

$$U_{n+1}^{k+1} = F(U_n^k) + G(U_n^{k+1}) - G(U_n^k) \tag{2.13}$$

$$= G(U_n^{k+1}) + \underline{F(U_n^k) - G(U_n^k)} \tag{2.14}$$

$$= G(U_n^{k+1}) + \underline{\tau_n^k}. \tag{2.15}$$

What this does is sequester the "expensive" evaluation of $F(*)$ into a part that can be calculated from the previous iteration. Then a "cheap" sequential evaluation can be done for this iterations updates. Moving the expensive parts into a τ-term depending on iteration k allows for the parallelization of this τ on each brick, and a cheap global sequential part replaces this. One can see that an update like (2.15) fits perfectly with Algorithm 2 for PinT written in the introduction. In this way, the "all at once" perspective naturally leads us to one of the fundamental algorithmic themes of PinT.

PinT algorithms which take the "all at once" approach are collectively called **Multiple Shooting Algorithms** [3, 21, 23, 16]. They can all be written in the form of (2.10), and are all predictor-corrector methods. In the next chapter, we outline three of the most popular multiple shooting algorithms for PinT integration, as well as how these methods relate to each other and Eqs. (2.13) and (2.10). Near the end of the chapter, we outline weaknesses of these methods to handle IVPs resulting from hyperbolic PDEs, or PDEs with finite speed of propagation.

Chapter 3

Overview of Methods in Time Parallel PDE Solvers

The three most popular predictor-corrector PinT methods in the literature are the **Parareal** [22], **Multigrid Reduction in Time (MGRIT)** [24] (a generalization of Parareal, which takes inspiration from Multigrid Reduction [25]), and Parallel Full Approximation Scheme in Space Time (**PFASST**—pronounced 'fast') [26]. These methods are all iterative predictor-correctors in nature, producing correction terms on each time brick for the predictions. Like all predictor-correctors, their general algorithmic structure is that of Algorithm 2, however, the manner in which each method produces corrections differs greatly.

In this section, we will outline the way in which each method produces corrections τ. First, we explore the sterotypical PinT method Parareal, one of the first PinT methods. We try our hand at parareal with a small toy problem, and write down some theory about speedup and convergence for parareal. Next, we look at MGRIT, and discuss how this is a generalization of parareal, indeed, we can show that under certain choices, MGRIT is equal to Parareal. We write a correction term for MGRIT in a general way. Then we give an overview of PFASST noting especially how PFASST is more restrictive than Parareal or MGRIT, but affords better theoretical speedup. Finally, we compare and contrast these methods before justifying the choice of method we will use in this dissertation.

3.1 Parareal Algorithm

The prototypical PinT algorithm is the parareal algorithm introduced in 2001 by Lions, Maday, and Turinci [22]. Parareal is arguably one of the most popular PinT methods. It has been tested in fluid-structure interactions [6], fluid flows [27], turbulence [9, 28], optimal control [29], Molecular Dynamics [30, 31], and even pricing American stock options [32], among others. In this section, we outline Parareal, state some convergence bounds for the literature, and speculate on it's performance for PDE's with finite speeds of propagation.

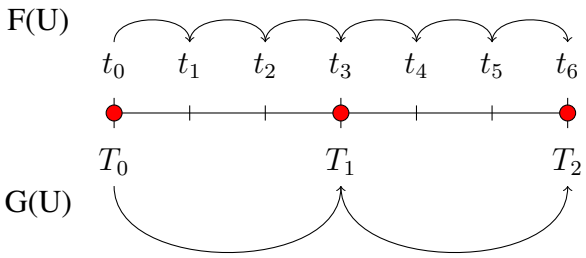


Figure 3.1: Parareal is a two level PinT method where we have a fine integrator F which takes steps at fine time points t_i, and a coarse integrator G which takes larger steps at a subset of the fine points T_j, called C-points. Parareal compares the difference at C-points to produce a correction.

Parareal is a two level algorithm where one utilizes a 'coarse' (hence cheap and computationally fast) propagator, together with a 'fine' (hence expensive and computationally slow) propagator, to iteratively produce improved guesses for solution state at various time points parallel in time. Therefore, this algorithm falls into the category of predictor-corrector PinT methods.

The Parareal algorithm proposes to use coarse guesses as the initial conditions of each time brick. Parareal then produces in parallel an informed correction on each time brick to be applied to the starting point of the subsequent initial condition. The goal is, upon iterating, that the initial conditions on each time brick are corrected to the true state. One can think of the prediction as a cheap sequential solution on a coarse time mesh to the IVP in question, and the correction—giving the accuracy of the method—as an expensive but time parallel solution to many sub-IVPs, one on each time brick, with which one can compute a correction to the prediction.

For Parareal we therefore assume that we have two time stepping operators, one fine operator F, and a coarse operator G as in 1, which can solve (1.1). For example, one could choose F to be the fourth order accurate Runge-Kutta 4 (RK4), and G to be implicit Euler. Typically, F is many magnitudes more expensive computationally than G.

Since Parareal is a predictor corrector algorithm, it has a general form is that of Algorithm 2, but this time with a specific correction procedure. Like before, we let U_i^n represent the solution at t_i on iteration n of our method.

Some key points about this algorithm:

Algorithm 3 Parareal

Follow Appendix 2, with corrections below.
Calculate $\tau_i = \mathcal{F}(U_{i-1}^{n-1}) - \mathcal{G}(U_{i-1}^{n-1})$ in Parallel
Sequentially propagate corrections

$$U_i^n = \mathcal{G}(U_{i-1}^n) + \tau_i$$

1. The correction τ_i, known in the literature as the **tau** correction, for all time points depends on the difference from the *previous iteration* of the coarse and fine propagators at that time point. This can be calculated in parallel since the coarse approximation is known at the start of the iteration, and the fine approximation is done on disjoint bricks and hence in parallel.

2. The corrections are *propagated*: a serial pass is made on the coarse level, where at each coarse time point the solution U_i is updated by correcting the coarse guess $G(U_{i-1})$, and this updated solution is propagated to the next coarse step. In other words, on iteration 1 we do $U_2^1 = G(U_1^1 + \tau_1) + \tau_2$, not $U_2^1 = U_2^0 + \tau_2$.

3. Since this algorithm is iterative, many time bricks run the fine computation many times. Therefore, this algorithm can be expected to use much more computational resources than a sequential solve which traverses each brick only once during a run.

4. It may not be obvious, but this algorithm is exact in N iterations, where N is the number of time bricks. If convergence happens in K iterations, we hope that $K \ll N$, otherwise the added computational effort required is for naught.

5. We can define any convergence criterion we want. Often we check if $U(t_{N-1})$ is converged.

Parareal, like other predictor corrector methods, needs a startup procedure to tell us the initial condition on each brick. Typically, one simply uses the coarse propagator as a kind of preconditioner, meaning to get the initial conditions, we do one initial solve with G. In practice, this is the best for speedy convergence, though in principle the initial condition can be anything.

Then in subsequent iterations, while not converged, we calculate a more accurate approximation to all of the coarse points (meaning the ends of each brick) with F on each time brick. This can be done in parallel since we have a guess to start each fine level approximation on each brick. Then a correction term, τ_i is calculated, which is simply the difference of the calculated fine approximation to the previous coarse guess. At this point, we do the coarse approximation in serial, and add the correction term. Iterating this process drives the solution state to match the fine propagator's accuracy for all $t_i \in T$.

The reason Parareal is guaranteed to be exact after we iterate the same number of times as there are bricks is because after the first iteration, B_1, which has an initial condition that is exact since all simulations start with an initial condition at time 0, has run the full—which is to say, sequential—simulation on the fine level. This means U_0 and U_1 should be equivalent to the fine solution which we would expect from a sequential solve. The other bricks have only stepped the coarse approximation in a fine way, and therefore are not expected to be exact. One clearly sees that on the next iteration B_2 will be exact in the same sense since the initial condition on B_2 is exact from the exactness of B_1. This continues for each time brick and each iteration.

From this basic analysis of exactness, we can tell that to get a computational speedup over sequential time stepping, we must have $K \ll N$. Indeed, Parareal is shown in [33] to have a speedup similar to

$$\text{Speedup} \sim \frac{N}{K-1},$$

where K is the number of iterations. This implies that if we can get convergence in two iterations we can achieve nearly perfect parallel speedup. For problems that are well behaved with parareal, we indeed see convergence in two iterations [33], while poorly behaved but still convergent problems may converge only after N iterations. This suggests two things: that for best use, one should pick N large; and that minimizing the number of iterations K is imperative. In the following example, we illustrate the type of speedup one can achieve in a simple problem, and the computational cost required to do so. This will illustrate the computational costs one needs to appreciate before using such a time parallel algorithm.

3.1.1 Numerical Example

Consider applying the Parareal algorithm to the IVP

$$X'' = -X, \quad X'(0) = 1, \quad X(0) = 0 \tag{3.1}$$

on the interval $[0, 1]$, which has the exact solution $X = \sin(t)$. To fit (1.1) better, we make this a vector ODE by defining

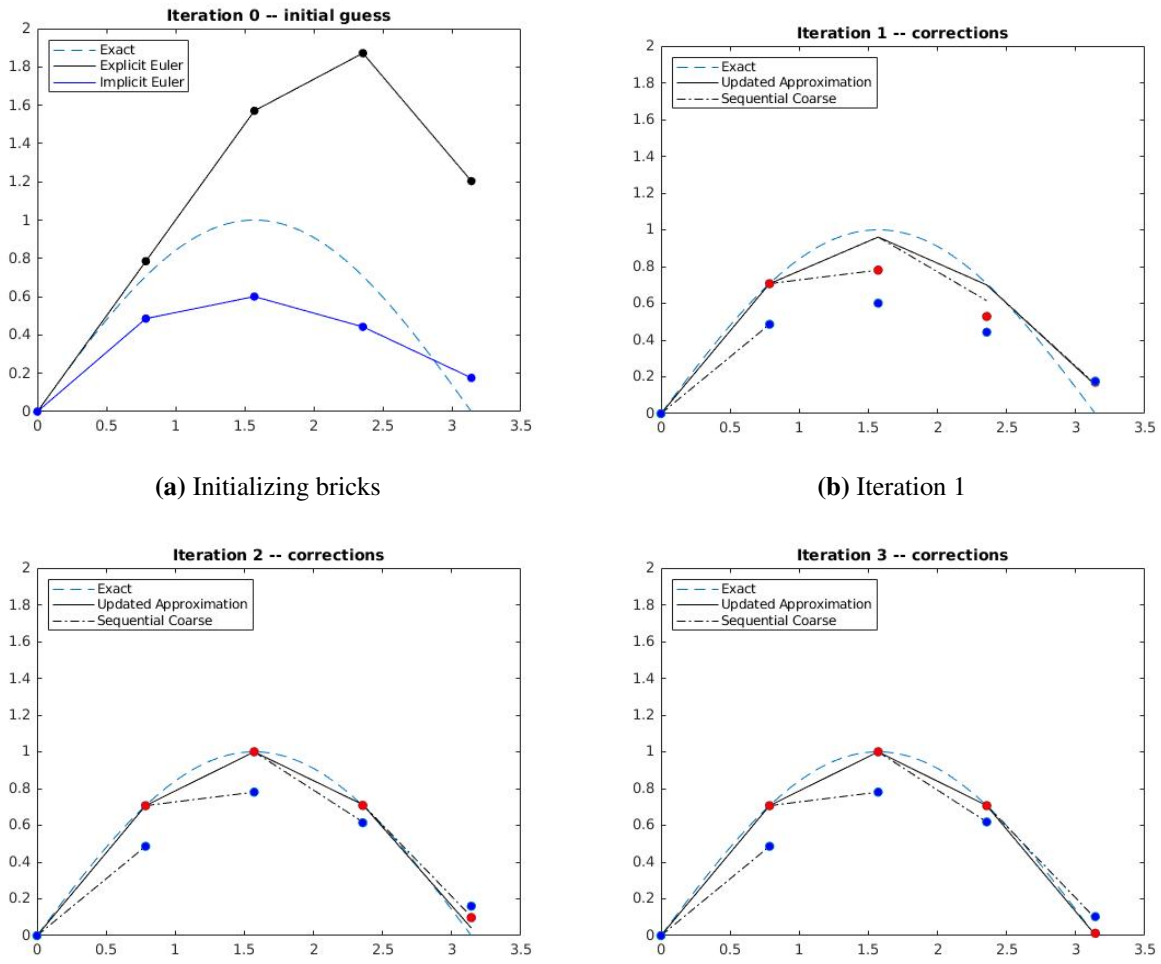
$$\frac{d\overline{X}}{dt} := \begin{bmatrix} X'(t) \\ V'(t) \end{bmatrix} = F(\overline{X}(t)) := \begin{bmatrix} V(t) \\ -X(t) \end{bmatrix} \text{ with } \overline{X}(0) = \begin{bmatrix} 0 \\ 1 \end{bmatrix}. \tag{3.2}$$

With Parareal, we can store the solution at

$$T_0 = 0 < T_1 = 0.25 < T_2 = 0.5 < T_3 = 0.75 < T_4 = 1,$$

hence $N = 4$ corresponding to time bricks $B_i = [T_i, T_{i+1}]$, $i \in [0, 1, 2, 3]$. Parareal then asks for an approximation to the solution at the start of each of these time bricks. This can be random, informed by a coarse approximation like the forward Euler method or implicit Euler method, or any other guess (two options for this specific problem are in Figure 3.2a).

Then, in parallel, since we have an initial state for each time brick, an accurate time integration is made on each brick and a difference is computed between the accurate integration and the coarse guess. This difference is the correction, τ_i. Once τ_i is known, the sequential coarse approximation is done again but this time, at each step of the sequence the proper correction is added to the solution. The final solution state of this process is then used as the initial coarse approximation to the next Parareal iteration. A full picture of the algorithm applied to (3.1) is given in figure Figure 3.2. Here, we see that convergence is satisfied after three iterations. Obviously, three is not much less than four, so the speedup is far from ideal.



(a) Initializing bricks

(b) Iteration 1

(c) Iteration 2

(d) Iteration 3

Figure 3.2: Top Left: We may have many choices for the initial conditions on each brick. Top Right: After selecting our initial coarse solution (blue) we compute the fine solution on each brick (red). Then, using the τ correction (red minus blue) we do a sequential coarse solve applying the τ in sequence. Bottom Left: On iteration 2, all points seem converged except the last. Bottom right: Convergence.

Based on the above, we see that our speedup should be $\frac{4}{3-1} = 2$. This problem was implemented in C++ with F being RK4, and G being implicit Euler. Four threads were used to handle the time parallel section of the Algorithm 0 The results are shown in the following table.

We can see from Table 3.1 that even though parareal converged for $K = 3$ with $N = 4$ bricks, we did get an appreciable speedup in nanoseconds. This gain is small, but nonzero. We get a solution faster than with a serial solution using RK4.

Table 3.1: Wall-time for three iterations of Parareal with $N = 4$ vs. sequential RK4 Solving (3.2).

Speedup of Parareal Applied to Equation (3.1)			
Method	Execution Time	CPU Time	Speedup Compared to RK4
Parareal	69154 [μs]	~272000 [μs]	1.21
RK4	83816 [μs]	~85000 [μs]	1

We note, however, that the amount of CPU time is much higher for parareal than the sequential RK4. This is due to the iterative nature of parareal. We are doing many passes of RK4 on each of the B_i each iteration. This is an important lesson to take: PinT methods can offer real speedup at large cost in computation time.

3.1.2 Parareal as a Multiple Shooting Algorithm

As alluded to in Section 2.1.2, Parareal corresponds to a particular choice for the term action of the Jacobian in (2.10), which we restate in its vector form here

$$\begin{bmatrix} U_0^{k+1} = U_0^k \\ U_{n+1}^{k+1} = F(U_n^k) + \frac{\partial F(U_n^k)}{\partial U_n^k}(U_n^{k+1} - U_n^k) \end{bmatrix} \implies \begin{bmatrix} U_0^{k+1} = U_0^k \\ U_{n+1}^{k+1} = F(U_n^k) + G(U_n^{k+1}) - G(U_n^k) \end{bmatrix} \tag{3.3}$$

where we can identify that

$$\frac{\partial F(U_n^k)}{\partial U_n^k}(U_n^{k+1} - U_n^k) \approx G(U_n^{k+1}) - G(U_n^k). \tag{3.4}$$

In other words, the action of the Jacobian has been approximated by a difference in the coarse operator's action on U between iterations. If we rearrange this equation, we get the form of the τ-correction for Parareal as was previously introduced:

$$U_{n+1}^{k+1} = F(U_n^k) + G(U_n^{k+1}) - G(U_n^k) \implies U_{n+1}^{k+1} = G(U_n^{k+1}) + F(U_n^k) - G(U_n^k). \tag{3.5}$$

Therefore, Parareal really is a multiple shooting algorithm, and fits into the overarching 'all at once' perspective introduced in Ch.1.

3.1.3 Convergence Properties and Maximum Speedup

Parareal has a couple convergence bounds that are worth restating here. The following theorem is due to [34], and illustrates the exactness guarantee of Parareal in N iterations, and a superlinear convergence bound based in K iterations. The notation has been adapted to match the notation in this dissertation.

Theorem 1. *Let $F(U_i^k, t_i, t_{i+1}) = u(t_{k+1})$ denote the exact solution at time t_{i+1} starting with U_i^k at t_i on iteration k, and $G(U_i^k, t_i, t_{i+1})$ denote the coarse solution at t_{i+1}, with an error bounded by $C_1 \Delta T^{p+1}$ where p is the order of the error. Assume U_n^k is the result of Parareal's kth iteration at the nth time point. If the coarse operator satisfies a Lipschitz condition in U (with an appropriate norm):*

$$||G(X, t, t + \Delta T) - G(Y, t, t + \Delta T)|| \leq (1 + C_2 \Delta T)||X - Y||,$$

then

$$\max_{1 \leq n \leq N} ||u(t_n) - U_n^k|| \leq \frac{C_1 \Delta T^{k(p+1)}}{k!} (1 + C_2 \Delta T)^{N-1-k} \prod_{j=1}^{k} (N - j) \max_{1 \leq n \leq N} ||u(t_n) - U_n^0|| \quad (3.6)$$

$$\leq \frac{(C_1)^k}{k!} e^{C_2 (T - (k+1)\Delta T)} \Delta T^{pk} \max_{1 \leq n \leq N} ||u(t_n) - U_n^0||. \quad (3.7)$$

The bound in (3.7) tells us that if the coarse operator is behaved well enough, we can show that Parareal has a superlinear convergence to the exact solution, this is the content of Equation 3.7. We also get a peek at the guarantee of exact convergence if we iterate N times, which is to say iterate the same as the number of bricks that we have, we see that the product will contain $N - N$, which means that the error will be zero. Overall this theorem should indicate to us that the convergence of Parareal depends mostly on the coarse propagator G. All terms on the right hand side of (3.6) and (3.7) all come from G's behavior.

Of course, it is possible that not all problems with such a coarse G operator satisfy this theorem with small constants C_1 or C_2, but the principle remains. Additionally, we note that Theorem 1 is true in principle if F is not the exact integrator. This means that we expect that kind of error bound

to hold for inexact fine operators. This is to say that for PinT often the 'exact' solution we wish to compute is the action of an expensive, 'inaccurate' fine integrator F.

3.2 Multigrid In Time Algorithm

Now that we understand the Parareal algorithm, we discuss a multilevel extension of it. **Multigrid Reduction in Time** (**MGRIT**) is an algorithm that was developed by researchers at Lawrence Livermore National Lab [24]. MGRIT is a time dimension analog to Spatial **Multigrid (MG)** Methods where we now use multiple time grids in a multigrid prediction-correction scheme.

Multigrid methods vary in implementation, splitting between Geometric Multigrid (GMG) and Algebraic Multigrid (AMG), both of which seek to solve a problem of the form

$$Au = g, \tag{3.8}$$

where, for example, A is a matrix resulting from the discretization of some system. Broadly speaking, MG methods construct a hierarchy of levels, typically a fine level and many recursively coarsened levels, where a so called 'restricted' version of the previous level's problem is defined and solved. This restriction is motivated by the fact that on coarser levels, different Fourier modes of the numerical error are more naturally described and solved for. Once the coarse error modes are known, we can produce corrections on the fine level by interpolating the error modes onto the fine level. As such, we call moving from a fine to a coarser level **restriction**, and moving from a coarse to a finer level **prolongation**.

AMG methods produce coarser levels in relation to the matrix A, while GMG methods create coarser levels by producing a hierarchy of discretization *meshes*, which each define their own version of (3.8) written as

$$A_l u_l = g_l \quad \text{for} \quad l \in 0, 1, 2, \ldots, L-1. \tag{3.9}$$

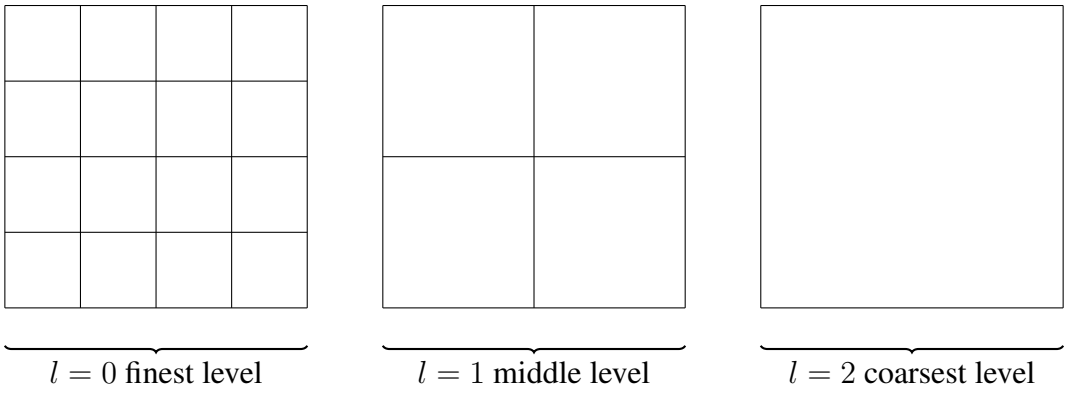


Figure 3.3: An example of a hierarchy of grids which can be used to form a set of problems to be used in GMG. Each grid's associated problem naturally captures certain modes of the error, those with wavelength proportional to the mesh size Δx, while also lessening the computational load of the problem.

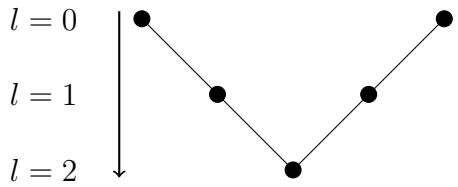


Figure 3.4: A MG **V-cycle**, which traverses the levels in the shape of a V.

Typically, the coarsest problem is at level $l = L - 1$, while the original fine problem is $l = 0$. L represents the total number of levels.

Multigrid methods once we select the basic type then traverse the levels in different orders. The first order one may think to try is a **V-cycle**, which traverses the levels from fine to coarse, then interpolates coarse to fine as in Figure 3.4. Another way we can traverse this is to start at the finest level, and work our way to the coarse level, but on interpolation, we bounce between coarser levels, known as an **F-cycle**, as in Figure 3.5. We can in theory define cycling in any way that we want, but these two cycle types are the most common in the PinT literature. Some key things to note are that different cycling types can have faster convergence properties, distribute the amount of processing done on each grid level, and can require more or less interpolation/restriction. All of these require careful consideration when using MG in any given problem. Overall, V- and F-cycles work well.

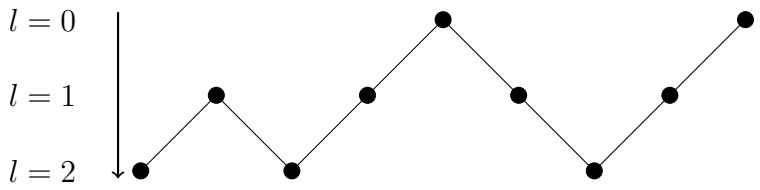


Figure 3.5: A MG **F-cycle**, which traverses the levels in the shape of a F on its side, roughly. One can see that this type of cycling requires more work than a V-cycle Figure 3.4.

Looking back at Parareal, we see that we have two *temporal* grids that we define our problems on. Naturally, Parareal as a two level scheme fits into the hierarchical nature of MG methods. Indeed, [24] writes Parareal as a two-level Temporal MG method. But it begs the question: Can we use more than two levels and get better PinT performance? MGRIT is an algorithm that extends Parareal to multiple levels while using the language of spatial MG methods (such as relaxation, restriction, interpolation, cycle-type).

Thinking of time integration in the same perspective as spatial MG methods expands the possibility for tuning our PinT algorithm to better suit whatever IVP we are interested in solving by allowing us to define appropriate restriction/interpolation operators, by tuning the cycling (V-cycles or F-Cycles), and deciding on different coarse operators on each multigrid level. Each of these choices can improve the performance of PinT integration if chosen appropriately. For example, using weighted relaxation methods [12], or varying the amount of relaxations performed per level [35], can change the convergence properties of MGRIT.

This section first outlines the MGRIT multigrid method, including the Full Approximation Scheme used to restrict the residual to coarser time levels, then presents the generalized MGRIT algorithm.

3.2.1 MGRIT's Origins as a Spatial Multigrid Analog

The MGRIT algorithm originates from viewing a sequential time integration as a matrix solution solved by applying well-known multigrid techniques. If done properly, these techniques can break the sequential nature of the matrix and replace it with an iterative multigrid procedure involving parallel computations. To illustrate how we can convert the full sequential time integration

problem (2.6), restated here:

$$\widetilde{F}(\overline{U}) = \overline{U} - F(\overline{U}) = \mathbf{0}.$$

Dropping the dependencies of $F(U_i, t_i, t_{i+1})$ and viewing it as an operator, the solution of equations (2.6) can be concisely written as a forward solve of the (possibly block) matrix system

$$\underbrace{\left\{ \underbrace{\begin{bmatrix} I & & & \\ -F & I & & \\ & \ddots & & \\ & & -F & I \end{bmatrix}}_{M\times(N+1)} \right.}_{M\times(N+1)} \begin{bmatrix} U_0 \\ U_1 \\ \vdots \\ U_{N-1} \\ U_N \end{bmatrix} = \mathbf{0}. \tag{3.10}$$

The literature typically calls this the fine system, and writes it concisely as

$$A\mathbf{U} = \mathbf{0}. \tag{3.11}$$

Note that since each U_n could represent a solution at M spatial degrees of freedom, the system (3.10), if we assume the operator F is linear, is solved as a block system of size $(M[N+1]) \times (M[N+1])$.

A two level MGRIT proceeds to define a coarser grid of time points, analogous to the C-points from Parareal, by coarsening the time domain by some factor m. For example, consider coarsening by $m = 2$ the set of time points $[t_0, \ldots, t_N]$ where the number of time points N is divisible by two as we can see in Figure 3.6. Note that in the count of number of time points we only count future time points, not T_0.

This procedure produces N/m coarse time points $[T_0, \ldots, T_{N/m}]$, which are a subset of the original time points. The coarse time points correspond to a coarse problem for $U_\Delta(T_i)$ of the

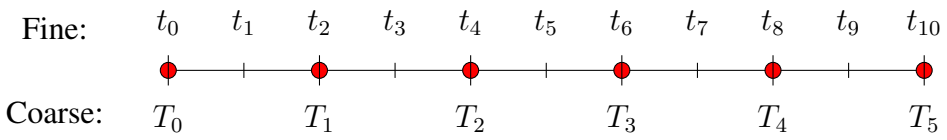


Figure 3.6: Coarsening the time dimension with $N = 10$ fine time points with $m = 2$ produces $N/m = 5$ coarse time nodes (red dots, excluding the start point) where we can do prediction-correction using MGRIT on bricks defined by the coarse points.

form

$$U_\Delta(T_0) = U_0, \quad U_\Delta(T_1) = F^m(U_\Delta(T_0)), \quad \ldots, \quad U_\Delta(T_{\frac{N}{m}}) = F^m(U_\Delta(T_{\frac{N}{m}-1})). \tag{3.12}$$

In the above, the notation F^m simply means m applications of F.

The problem (3.12) states that the coarse problem is the same as solving m steps of the fine problem of the coarse points, i.e. it is exactly the same as (3.10). In this light, we now write the coarse system matrix as

$$\underbrace{\begin{bmatrix} I & & & \\ -F^m & I & & \\ & \ddots & & \\ & & -F^m & I \end{bmatrix}}_{\frac{N}{m}+1 \times \frac{N}{m}+1 \text{ matrix, excluding spatial dofs}} \begin{bmatrix} U_{\Delta,0} \\ U_{\Delta,1} \\ \vdots \\ U_{\Delta,\frac{N}{m}-1} \\ U_{\Delta,\frac{N}{m}} \end{bmatrix} = \mathbf{0}_\Delta \tag{3.13}$$

where $U_{\Delta,i}$ is the ith coarse point. The coarse system solves for all the coarse points with

$$U_{\Delta,i+1} = F^m(U_{\Delta,i}) := \underbrace{F(F(F \ldots F(U_{\Delta,i})))}_{m\text{-applications of } F}. \tag{3.14}$$

By denoting

$$A_\Delta = \begin{bmatrix} I & & & \\ -F^m & I & & \\ & \ddots & & \\ & & -F^m & I \end{bmatrix} \tag{3.15}$$

we can write the resulting coarse system we will write as

$$A_\Delta \mathbf{U}_\Delta = \mathbf{0}_\Delta. \tag{3.16}$$

Solving this system (3.13) is no less computationally expensive to solving the fine system (3.10) due to the fact that we still need to perform a forward solve of A_Δ, which means passing our state through the time integrator F m times ber brick. Doing so is identical to the full sequential problem.

MGRIT's approach, therefore, is to approximate the matrix in (3.16) with a cheaper matrix which closely approximates that of A_Δ, i.e we find a $B_\Delta \approx A_\Delta$.

We do this by choosing an operator $G \approx F^m$, which approximates the action of m applications of F. This results in a sequence like

$$U_\Delta(T_0) = U_0, \quad U_\Delta(T_1) = G(U_\Delta(T_0)), \quad \ldots, \quad U_\Delta(T_{\frac{N}{m}}) = G(U_\Delta(T_{\frac{N}{m}-1})), \tag{3.17}$$

which is represented by

$$B_\Delta \mathbf{U}_{\mathbf{\Delta}} = \begin{bmatrix} I & & & \\ -G & I & & \\ & \ldots & & \\ & & -G & I \end{bmatrix} \cdot \begin{bmatrix} U_{\Delta,0} \\ U_{\Delta,1} \\ \vdots \\ U_{\Delta,\frac{N}{m}-1} \\ U_{\Delta,\frac{N}{m}} \end{bmatrix} = \mathbf{g}_\Delta. \tag{3.18}$$

In (3.18), $\mathbf{g}_\Delta$ is a possible forcing term needed to make the coarse level consistent with the fine level. The equation (3.18) looks close to what we had to solve before, but the operator G is chosen to be cheaper to evaluate than m evaluations of F. The solution of this system can be thought of as a solution to a coarser system which closely approximates the fine system. Note that this coarser system is smaller by a factor of m than the fine system (3.10). Viewing the problem this way, we are entering the language of Multigrid Methods, so we will need to define restriction operations, relaxation operations, and prolongation operations.

Restriction is the way we choose to communicate the fine level to the coarse level. There are various ways in which restriction from the F-points could be defined in principle. The simplest way to do this is to simply set $U_{\Delta,i} = U_{m*i}$ which is possible since the set of C-points is a subset of the F-points (for a different version of this, see [12]). Therefore the restriction operator is

$$R : U_{m*i} \to U_{\Delta,i} \tag{3.19}$$

Prolongation is done in various ways, but often, simply injecting the common coarse points into the fine mesh (the inverse operation of the restriction) is done. So we define the prolongation operator

$$P : U_{\Delta,i} \to U_{m*i}. \tag{3.20}$$

Using P, we can smooth the error at all the C-points (the $m * i$ subscript indicates we only get corrections for error at C-points) on the fine level, but how about the F-points on the fine level?

To smooth the error on the fine level at all the F points we should have a *relaxation* that relaxes the corrected solutions at the C-points to all the F-points. Since our corrections on each level happen at the initial conditions on each brick, relaxation is simply a choice of how many times to perform our time integration from the beginning of each brick. For example, [24] defines two relaxation techniques: FC and FCF relaxation. FC relaxation is demonstrated in Figure 3.7, and involves integrating from the initial condition on each brick through all of the fine points on each brick **(F)**, followed by integrating to the final time on each brick (C). Therefore one sweep of FC

relaxation results in the first brick being exact. On the other hand, we can do FCF relaxation, which is FC plus another F relaxation, resulting in two bricks being exact before moving to the next coarsest level.

3.2.2 Full Approximation Scheme

To enable us to state the full algorithm for MGRIT, we need to define the forcing term mentioned in (3.18) that forces consistency between the fine and coarse level.

For MG schemes, one can think of the coarser problems as a way to solve for the error on the finer level, which once known can be prolonged to the fine level and subtracted on the fine level. We need a way to solve for the fine error on the coarser level. Then, once knowing the error, we can correct the fine level followed by a relaxation of the error.

The way that MGRIT connects the coarser level to the finer level is through the **Full Approximation Scheme (FAS)** for nonlinear problems [36]. The idea of this is to restrict the approximate solution Au, in other words, we restrict the residual $r = g - Au$ to a coarser (cheaper) level as the forcing term. Once this is done the cheaper problem is solved for the error on the coarser level, but hopefully with the accuracy of the finer level. Knowing the error allows for corrections.

The way in which we also produce a correction term on the coarse level seems different than how we produce corrections in the Parareal or MGRIT methods, and it is in the sense that those methods do not seem to use any corrections on the coarse level–however, there is no difference because the Parareal correction is computed only at C-points, the ends of the time bricks.

We now outline the way FAS constructs an estimation of the error on the fine level. Consider a problem of the form

$$A(U) = g \tag{3.21}$$

is defined on a fine grid. This could, in a time dependent problem, come from a similar equation to the “all at once" idea presented in (2.6). However, the development of the FAS ideas were for spatial multigrid; we make no effort in this section to make a distinction between spatial and temporal problems in this subsection. Given that we can approximate the exact solution U with $\tilde{U}$,

we can write the error $e = U - \tilde{U}$, and the residual as

$$r = g - A(\tilde{U}). \tag{3.22}$$

The residual is a measure of how closely our approximation satisfies the exact equation. If we subtract our exact equation from the residual, we can get another form for the residual

$$A(U) - A(\tilde{U}) = A(\tilde{U} + \epsilon) - A(\tilde{U}) = r \tag{3.23}$$

If A is linear, then we get an equation for the error $A(\epsilon) = r$, but generally we make no suppositions about A, so (3.23) is the best we can do. Equation (3.23) is the error equation on the fine level, and if we could solve it for ϵ, based on some approximation $\tilde{U}$, we can solve for our exact solution U by just adding the error since $U = \tilde{U} + \epsilon$. Practically, solving the error equation for ϵ is as difficult as solving the original (see discussion in Section 3.2.1. In FAS, the solution is to solve an error equation approximately on a coarser grid, and then prolongate this error to the fine level to add to our approximate solution.

To help define the coarse grid error problem, we introduce inter-grid restriction R (which moves solutions from the fine level to the coarse level) and prolongation P (which moves solutions from the coarse level to the fine level), again, see Section 3.2.1, which translate approximations and errors between grids. We then define a coarser problem,

$$A_\Delta(U_\Delta) = g_\Delta, \tag{3.24}$$

by defining a coarsened operator A_Δ, usually some approximation of A, on a coarser grid, and by restricting g to get $g_\Delta = R[g]$. We can write a coarse equivalent to (3.23) by using the restriction operator R on the approximate fine solution and on the residual, resulting in

$$A_\Delta(R[\tilde{U}] + \epsilon_\Delta) - A(R[\tilde{U}]) = r_\Delta = R[r] = R[g - A(\tilde{U})] \tag{3.25}$$

if we assume that the restriction is linear (for MGRIT, the restriction R really is linear [24]) and writing $U_\Delta = R[\tilde{U}] + \epsilon_\Delta$, we can rearrange the problem into a *corrected* coarse problem:

$$A_\Delta(U_\Delta) = R[g] + A_\Delta(R[\tilde{U}]) - R[A(\tilde{U})] \tag{3.26}$$

$$A_\Delta(U_\Delta) = g_\Delta + \underbrace{A_\Delta(R[\tilde{U}]) - R[A(\tilde{U})]}_{\tau_\Delta} = g_\Delta + \tau_\Delta \tag{3.27}$$

where we could compute U_Δ as the exact solution to the corrected equation (3.27) and the correction term $\tau_\Delta = A_\Delta(R[\tilde{U}]) - R[A(\tilde{U})]$. We form the correction term from a difference between the action of A_Δ on the restricted fine approximation $\tilde{U}$, and the restriction of the action of A from the fine problem on $\tilde{U}$. Using the corrected problem allows the coarse problem to achieve the accuracy of the fine problem, but with the resolution of the coarse problem [36, 37]. Once an approximation $\tilde{U}_\Delta$ is solved on this grid for the exact U_Δ, we calculate the error as $\epsilon_\Delta = \tilde{U}_\Delta - R[\tilde{U}]$, and can prolongate ϵ_Δ to the fine level to get a correction to the fine approximation as

$$\tilde{U}_{\text{new}} = \tilde{U} + P[\epsilon_\Delta]. \tag{3.28}$$

This process can be equivalently thought of as solving directly for the error on the coarse level then subtracting the error from the approximate solution $\tilde{U}$ on the finer level, with the hope that this makes our approximation closer to the exact solution. FAS is used as a convergence accelerator for multigrid methods in space, but the theories of solving for the error on the coarse grid of a more accurate *corrected* problem can apply equally well for time integration schemes, and actually play a role in the way in which MGRIT is defined in [24], which is also in Algorithm 4.

Restriction of Residual vs. Interpolation of Residual

Be warned that the FAS restriction typically does not mean that we are *interpolating* our solution from the fine time grid to the coarse time grid. Indeed, since these grids share common points, the interpolation is unnecessary. Normally, the restriction operation of MGRIT FAS stage is simply an operation where the result from the end of a brick (otherwise called a C-point) is

taken up to the coarser level by copying the information from the end of the brick. This does not involve any interpolation because the time point on each level is the same. The authors of **XBraid**, a software implementation of the MGRIT algorithm, note that "Interpolation is ideal or exact, in that an application of interpolation leaves a zero residual at all F-points" [38, p. 9], if the residual on the coarse level is zero. Here the word 'Interpolation' means the restriction defined in the FAS.

In other words (using the notation from Definition 2 and Section 3.2.2) if $U_\Delta(t_{n+1})$ is the solution on the coarse level of the time brick B_n and $U(t_{n,M-1})$ is the solution at the end of the time brick B_n on the fine level, the restriction operation is

$$R : U_\Delta(t_{n+1}) \to U(t_{n,M-1}), \tag{3.29}$$

since by definition $t_{n+1} = t_{n,M-1}$. We do not need interpolation if the sample points are the same.

The main point to understand while using FAS is that the solutions $\tilde{U}$ and $\tilde{U}_\Delta$ need to represent solutions to the same problem, just with different levels of accuracy—here, the problem is (3.23). Importantly, this problem represents (at least in the MGRIT framework) the integration in the dimension of time. So the fine and coarse problems represent different accuracies in time. This is not to say that the spatial discretization is unimportant, only that this aspect of the problem does not change from the fine time mesh to the coarse. After all, we are solving PDEs, which have more dimensions than just time.

However, a given spatial mesh has an essential speed-limit for hyperbolic PDEs which strongly influences the time grids we are allowed to use in MGRIT: called the **CFL** condition

$$\Delta t = \mathbf{C} \max_{x \in \Omega} \cdot \frac{\Delta x}{||\mathbf{u}(x)|| + a(U(x))}, \tag{3.30}$$

where $\mathbf{u}(x)$ is the velocity vector and $a(U(x))$ is speed of sound (a function of the whole fluid state $U(x)$), each of which may vary spatially, Ω is the spatial mesh, Δx is the size of the cells in the triangulation for the spatial mesh, and C is a parameter saying what portion of the maximum we should allow for Δt. This equation says that for a given spatial mesh Ω there is a largest Δt that

results in stable time integration. For MGRIT, this can be problematic since we want aggressively coarser time grids requiring large Δt.

Generally we want much coarser time meshes in time for performance, so these CFL restrictions are unfortunate since they effectively put an upper bound on the allowable Δt for any time grid. Instead, we could coarsen in space as well, so that a larger time step is allowed (for CFL-like conditions, larger Δx implies larger Δt). However, this would mean that the fine time grid and the coarse time grid differ in which spatial grid they solve on. In principle, there is no reason why this is not something one could do, but it probably comes at a cost. The reason we might expect to pay a price for both spatial and temporal coarsening is that our FAS restriction would now also require an interpolation stage.

To better understand the cost we pay, let's rewrite the FAS equations when we need to interpolate between spatial meshes. Let $\mathbb{T}$ be the spatial mesh for the fine problem $A(U) = g$ and $\mathbb{T}^c$ be the spatial mesh for the coarse problem $A_\Delta(U_\Delta) = g_\Delta$. Following the logic of FAS, the coarse problem is defined by restricting the b from the fine problem. Previously, this was simply a usage of the values from b at the end of our time bricks. Now the underlying spaces are different so we need to inject interpolation functions $I_f^c : \mathbb{T} \to \mathbb{T}^c$ and $I_c^f : \mathbb{T}^c \to \mathbb{T}$ into our restriction. Then, the coarse problem (3.27) would now read

$$A_\Delta(U_\Delta) = \underbrace{R[I_f^c(b)]}_{g_\Delta} + \underbrace{A_\Delta(R[I_f^c(\tilde{U})]) - R[I_f^c(A(\tilde{U}))]}_{\tau_\Delta}. \tag{3.31}$$

We can see that doing this injects an interpolation error into the coarse problem. Therefore, we should expect that at a certain point, this error might dominate the problem.

Further, even if the interpolation error on the coarse mesh is small, we still need to prolongate the error from the coarse mesh to the fine, resulting in

$$\tilde{U}_{\text{new}} = \tilde{U} + \underbrace{P[I_c^f(\epsilon_\Delta)]}_{\epsilon}. \tag{3.32}$$

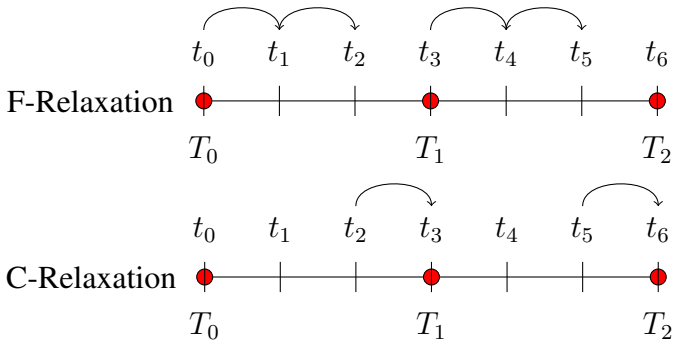


Figure 3.7: For FC relaxation, each brick integrates from its initial condition through all of the fine level time points it contains (F relaxation), which is followed by integrating to the end point of the brick (C relaxation). Relaxing the problem like this results in the so called **FC relaxation**. FCF relaxation would therefore be followed by another F relaxation.

There is a potentially different interpolation error here in the presence of I_c^f before prolongation. Of course if these errors are small, then we would expect the same behavior as the FAS on the same spatial mesh. In general, though, this source of error is not zero, so eventually it should dominate the MGRIT convergence. Still, if one can live with this error, then one is allowed to use much coarser time meshes. One may even want to use coarser meshes regardless of this added error, see Section 4.2.2

In [24], the authors note that Algorithm 4 is equivalent to Parareal under the condition that we use two levels, $L = 2$, and if FC relaxation is used. Since Parareal is a special case of MGRIT, we can see that MGRIT is a generalization of Parareal. Indeed, many authors [39, 24, 35, 10, 12, 40, 7] look to MGRIT for avenues of speedup that Parareal does not contain—different numbers of levels, different relaxation options—in the hope that these provide more optimal speedup. Parareal in comparison only offers a choice in how we define the operators F, G.

3.2.3 Statement of the MGRIT Algorithm

With the language from Section 3.2.1 and Section 3.2.2, we now state the full MGRIT algorithm in Algorithm 4. The basic MGRIT algorithm is the basis for the modified version used in this dissertation, see Algorithm 7.

Algorithm 4 highlights the interplay between parallel and sequential computations. There always is some coarse sequential solve that informs the fine parallel solves. Ultimately, the speedup will be limited by how accurate the coarse level solves the same problem as the fine, and also how fast. If the accuracy lacks on the coarse level, we should expect either no convergence or slow convergence, which will mean that the number of cycles K will be high. If the sequential solution is slow, then we should expect slow time to solution since we have a seqeuential bottleneck.

Algorithm 4 L-Level MGRIT with $FC-*$ relaxation and K iterations of V-cycles (L-MGRIT-FC*-V)

Produce an initial guess U_0^0 at C-points.
for Iteration $k \in \{0, \ldots, K\}$ **do**
  Down Cycle:
  for Level $l = 0;\ l \leq L-1; l = l+1$ **do**
    if $l \neq L-1$ **then**
      In parallel, relax $A_l U_l^k = g_l$ using a choice of relaxation $FC-*$ (integrate all bricks).
      Compute and restrict the residual using FAS (Section 3.2.2) at C-points

$$g_{l+1}^k = R\big[g_l^k - A_l U_l^k\big].$$

    end if
    if $l = L-1$ **then**
      Solve $A_l U_l^k = g_l$ exactly (sequentially) for the error $\epsilon_l = U_l^k - R[U_{l-1}^k]$.
    end if
  end for
  Up Cycle:
  for Level $l = L-2;\ l \geq 0;\ l = l-1$ **do**
    Correct using coarse error from FAS:

$$U_l^{k+1} \leftarrow U_l^k + P(\epsilon_{l+1}^k).$$

  end for
end for

3.3 Other Methods

Parareal and MGRIT are only two methods for PinT. Already alluded to in Section 2.1.1, PinT methods can be defined differently based on how we do space-time decompositions for our processors. Parareal and MGRIT are two of the brick based predictor-correctors, but another popular PinT method in the literature is the **Parallel Full Approximation Scheme in Space and**

Time (PFASST) introduced in [26]. A Google scholar search can show that (at the time of writing) Parareal has about 9000 results, PFASST has about 3000, while MGRIT has about 600. So PFASST is popular more on the scale of Parareal.

Other methods like Waveform Relaxation [16, 19] come from other spacetime decompositions. In this section, we only overview PFASST–without much detail–because it is brick-based like Parareal and MGRIT. All other methods we do not go into any detail, and instead point the reader to citations [16, 2] for general reviews of all PinT variations.

3.3.1 PFASST

The PFASST method, introduced in [26], produces the τ correction with a Full Approximation Scheme [36, 37] on multiple levels, like MGRIT and Parareal both do, but is based on modifying the ODE to be solved on a set of quadrature nodes in time, contrasted to Parareal and MGRIT which make fewer requirements on the actual time points we solve for during solution. To wit, PFASST writes a so-called Picard form of (1.1):

$$U(t) = U(0) + \int_0^T f(s, U(s))ds. \tag{3.33}$$

The solution of (3.33) can be computed by using quadrature for the integral term, assuming we break the integral into its respective bricks. Using quadrature on the points $\{t_i\}_{i=0,\dots,N_q} \subset [0, T]$, we define

$$\boldsymbol{U} = \begin{bmatrix} U(t_0) \\ \vdots \\ U(t_{N_q}) \end{bmatrix} \text{ and } F(\boldsymbol{U}) = \begin{bmatrix} f(t_0, U(t_0)) \\ \vdots \\ f(t_{N_q}, U(t_{N_q})) \end{bmatrix}, \tag{3.34}$$

allowing us to write an approximate solution as

$$\boldsymbol{U}(T) = U(0) + \boldsymbol{S}F(\boldsymbol{U}) \tag{3.35}$$

with $\boldsymbol{S}$ the matrix of quadrature weights.

Doing such an approximation allows PFASST to coarsens the number quadrature nodes [26] on each brick, creating a cheaper quadrature approximation for (3.34). PFASST forms a FAS scheme on the coarser levels, using the cheaper quadrature to solve the more expensive quadrature error. To ease solution transfer between levels, often the coarser quadrature nodes are taken to be a subset of the finer (higher accuracy) ones (of which there are more).

To compute the necessary values for $F(\boldsymbol{U})$, PFASST does a prediction stage, and then produces multilevel corrections by a coarser problem defined with restriction of the residual and error (like FAS), but only on time nodes that correspond to coarser quadrature points on the interval $[0, T]$ for the whole time interval. This procedure is also iterative like most PinT methods, but unlike Parareal and MGRIT, requires the use of **Spectral Deferred Correction (SDC)**[41] as the time integrator F or G. For some applications, this lack of choice may be a downside, especially if a code has been designed without SDC in mind, making PFASST highly invasive to the codebase. The benefits of this PinT method are that it has a higher theoretical speedup than that in Parareal, and its accuracy can be made high order as SDC offers [41]. PFASST has been used on many of the same problems as Parareal and MGRIT [26, 42, 43].

Because PFASST is so invasive for the underlying brick time integrators F, G, we do not use this method in this thesis.

3.4 Summary of Key Ingredients to any PinT Method

From all of these methods and their connections. We observe a few key ingredients that any PinT method needs to consider. First, there are always **multiple levels of time grids that we need to solve** on, and it is up to the practicioner to decide how to define this coarser problem. Then, all methods, being iterative in nature, need to **produce a correction term**, and it is up to the practitioner again to define how this is done. From within these two considerations that must be addressed lie most of the research directions.

For the coarser time grid, each of Parareal, MGRIT, and PFASST have found it useful to couple restriction in time with restriction in space. For this, restriction operators which take the solution

from a finer spatial mesh to a coarser spatial mesh, while ideally retaining important qualities of the fine-mesh solution, must be defined. PFASST is most explicit about this, with the FAS coarse correction most clearly forcing the solution on the coarse grid to better approximate the fine level solution. However, the way one defines the coarse operator in Parareal/MGRIT also benefits from matching the fine solution more accurately. Then, we also need interpolation functions which closely retain the important qualities. We will see that these mesh transfer functions have use in extensions of MGRIT like seen in [44] which uses MGRIT to solve Taylor-Green vortex with a PinT method.

Then, we must define how we produce the correction term. In Parareal and MGRIT, the τ correction is defined by a difference between the fine and coarse propagators, and can be easily extended to higher order corrections [10], with the hopes that these corrections speed up convergence. In PFASST, the correction term comes from solving a corrected equation on a coarser level for the error on the fine level. The way these problems are solved on each grid involve different operators/processes.

Parareal and MGRIT are agnostic to the way one produces a result at a time point t_i. This offers certain ease in implementation, because F and G the fine and coarse operators are black-boxes, where it does not matter to the functioning of these two algorithms how a solution is produced. The flexibility of this is an attractive aspect for the algorithms, though the speedup is limited. With a special choice of F and G–that being SDC on certain fine and coarse quadrature nodes in time– PFASST can achieve a greater parallel speedup, but is algorithmically invasive as it requires the adoption of SDC for time integration. This means that existing codes would need a more intensive rewrite to use PFASST.

We list all the considerations one might use to select a PinT method for their own purposes:

1. Do we want to achieve the best possible speedup?

2. Do we have a favorite time integrator F that is already tailored to my specific problem?

3. Does my problem have specific properties to which my choice of PinT method is amenable?

3.5 Review of Methods

Parareal is known as a two level algorithm meaning that it produces correction terms between a fine and a coarse level where the method happens to be agnostic to the underlying integrator. Variations of this method exist, but the typical correction term that is applied in this is a simple difference between the fine and the coarse levels at some specified coarse time points:

$$\tau_i = F(U_{i-1}) - G(U_{i-1}). \tag{3.36}$$

MGRIT can be thought of as an extension of Parareal to multiple levels. Indeed, under special choices for the number of levels and the relaxation technique, MGRIT is equivalent to Parareal. However, MGRIT uses the language of multigrid methods, affording more optimal strategies for iterating like varying the cycle (V or F cycles for example) and different relaxation techniques (weighted relaxation, multiple relaxation sweeps on each brick, and so on). Being in the language of multigrid also allows the application of some well known theory allowing the derivation of error estimates for choices of relaxation and cycle. MGRIT is agnostic to the underlying integrators as well. MGRIT produces corrections on each level by difference to a coarser level just like Parareal does. Therefore the correction term that shows up here is a function of correction terms on multiple levels L and cycle type C_t:

$$\tau_i = \mathcal{T}(\tau_i^1, \ldots, \tau_i^L, C_t). \tag{3.37}$$

PFASST was born from the Spectral Deferred Correction (SDC) method and the Full Approximation Scheme (FAS). In its two level form, PFASST uses a high order quadrature (in time, what SDC does) on the fine level and a lower order quadrature (in time) on the coarse level. Then an error estimate is solved on the coarser level using the full approximation scheme and is interpolated down to the finer level. This differs from the other two algorithms in that it requires a quadrature based time integration – so is not agnostic to the type of integrator used like the other two. It requires SDC as the integrator. However, it turns out that the optimum parallel performance of

this method is theoretically higher than the other two algorithms because SDC is used as the time integrator.

Requisite for this method are proper transfer functions T_c^f, T_f^c, which transfer corrections τ^* and states U^* from the coarse to the fine grid, and the fine to the coarse grid, respectively. Then the correction term is given by an interpolation from the coarser level correction term:

$$\tau_i = T_c^f(\tau_i^c), \tag{3.38}$$

where τ_i^c is calculated with SDC and FAS on a cheaper coarse level. If a correction at a certain node i does not exist on the coarser level, we need to interpolate it from corrections at surrounding points. (PFASST makes a distinction between interpolating in space only–where the quadrature nodes are shared, and interpolating in space and time–where the finer quadrature nodes are not present in the coarse level and hence an interpolated value is required).

In all, these **multiple shooting (MS)** algorithms differ only in how they produce corrections on the bricks. Parareal does two level coarse differencing, MGRIT does a multilevel version of coarse differencing, and PFASST uses many levels of SDC for corrections. But all share the τ-correction form if one carefully writes their algorithms out. In light of this, PinT will mean methods for time integration which use **Prediction then Correction** on many bricks in parallel.

Chapter 4

Slow Convergence of PinT in Hyperbolic Setting

The PinT literature has established that vanilla PinT methods (Parareal or MGRIT) do not behave well with problems that are second order ODEs [8, 45], or problems that are hyperbolic in nature [46, 40]. Full pointwise convergence of the solution state $U(t)$ in these types of problems often takes as many iterations as there are bricks, so all of the parallel speedup is lost. This amounts to waiting for the equivalent of a sequential solution to propagate to the end time, while doing lots of extra computational work.

We can think of this as being because of the presence of waves at the current time whose effects on some distant time brick cannot be easily known (i.e. predicted) well enough for Parareal to converge unless those waves propagate through the time direction (which is a sequential process). This suggests, of course, that tailoring the coarse propagator to capture as accurately as possible the waves that contribute most to the accuracy of the problem is ideal. For example, one could design higher order corrections [10] with the hope that this correction captures better the propagation of solutions; one could use a Fourier decomposition to select the wavenumbers we care about and design a coarse propagator for this [28]; or, as is done most frequently, one ensures that the coarser time problem also corresponds to a coarser spatial problem where larger wavelengths are more naturally described, meaning the essence of the solution will be well resolved on this level [9, 13, 11, 47].

Because of this property, the PinT algorithms outlined in Chapter 3 struggle with systems of equations which have finite finite speeds of wave propagation without damping/dissipation. This limitation also occurs with the other types of PinT (MGRIT and PFASST) we introduce here. We now include a numerical example to illustrate what this looks like.

4.1 Numerical Example: Euler Equations at Mach 3 Around Two Cylinders

For an illustration of PinT issues in hyperbolic systems, consider a two dimensional (dim=2) problem which models the flow of air around two cylinders without gravity. In the absence of viscosity, this system can be described by the Euler equations which represent the laws of conservation of mass (via the scalar density ρ), momentum (via the product of the density ρ and velocity vector $\mathbf{u}$), and energy (via the scalar e) for a fluid, together with a description of the pressure p (known as an equation of state $p(\rho, \mathbf{u}, e)$):

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{u}) = 0 \qquad \text{Conservation of Mass} \tag{4.1}$$

$$\frac{\partial (\rho \mathbf{u})}{\partial t} + \nabla \cdot (\rho \mathbf{u} \otimes \mathbf{u}) + \nabla p = 0 \qquad \text{Conservation of Momentum Equation(s)} \tag{4.2}$$

$$\frac{\partial e}{\partial t} + \nabla \cdot (e \mathbf{u}) = 0 \qquad \text{Conservation of Energy} \tag{4.3}$$

Typically, this system can be thought in a vector form where we want to solve the time evolution of

$$U = \begin{bmatrix} \rho \\ \rho \mathbf{u} \\ E \end{bmatrix} \in \mathbb{R}^{\text{dim}+2}, \tag{4.4}$$

where $E = \rho e + \frac{||p\boldsymbol{u}||^2}{2\rho}$ the total energy, with which we create the vector valued PDE

$$\frac{\partial U}{\partial t} = -\nabla \cdot F(U), \tag{4.5}$$

where

$$F(U) = \begin{bmatrix} \rho \mathbf{u}^T \\ \rho \mathbf{u} \otimes \mathbf{u} + p\mathbf{I} \\ \mathbf{u}(E + p) \end{bmatrix}, \tag{4.6}$$

where p is the pressure given for example by a polytropic gas equation of state (EOS):

$$p = (\gamma - 1)(E - \frac{||\rho \boldsymbol{u}||^2}{2\rho}), \tag{4.7}$$

where γ is the ratio of specific heats (in this paper we set $\gamma = 7/5$). We also choose an expression for the entropy ψ in this system:

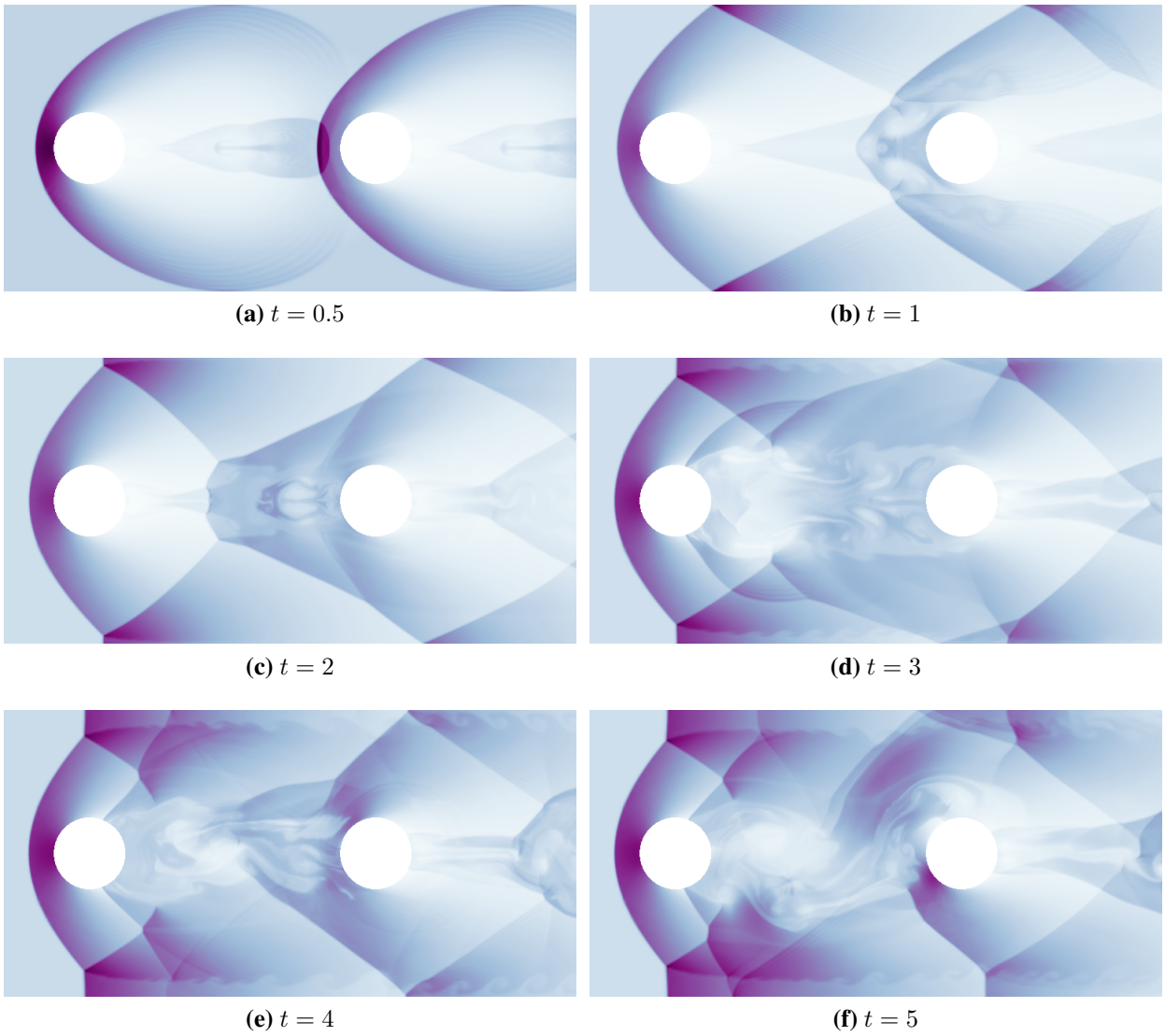
$$\psi = \frac{\rho e}{\rho^\gamma}. \tag{4.8}$$

From (4.5), we can see that we have a problem that fits our expectations from (1.1). Assuming we know how to discretize the right hand side of (4.5), we can choose operators as in MGRIT and check the solution state of, for example, the evolution of the density. The exact fluid might evolve as in Figure 4.1, where we can see complex wave phenomena in the wake of the first cylinder which interact on the surface of the second cylinder. This solution is given by ryujin, a library implementing a second order, graph-viscosity based upwind scheme for fluid flow problems [48, 49].

A PinT implementation with $N = 80$ bricks and $K = 4$ iterations struggles to recreate the complex wave phenomena that we see in the exact solution, for example see $t = 4$ seconds in Figure 4.2. We compare the wake of the first cylinder and see that the waveforms are nowhere near close to the truth. For another, the shock locations on the bow-shock in front of the leading cylinder has strange, potentially nonphysical artifacts.

Still, on a larger scale, we can see that the PinT algorithm more or less captures the behavior of the fluid flow at a large scale. Nevertheless we should expect that the PinT method will *not* converge in a pointwise sense until all the bricks have run essentially in sequence–not the goal of our PinT algorithm. In other words, we will have convergence in $K = 80$ iterations. For more clarity in this, see a direct comparison between $t = 4$ from the exact and from the PinT implementation Figure 4.3, where clearly the density is not quite right.

This issue is difficult to overcome. Essentially, the reason for this problem is that it is difficult if not impossible to predict the advection of high frequency wave-forms far into the future (while



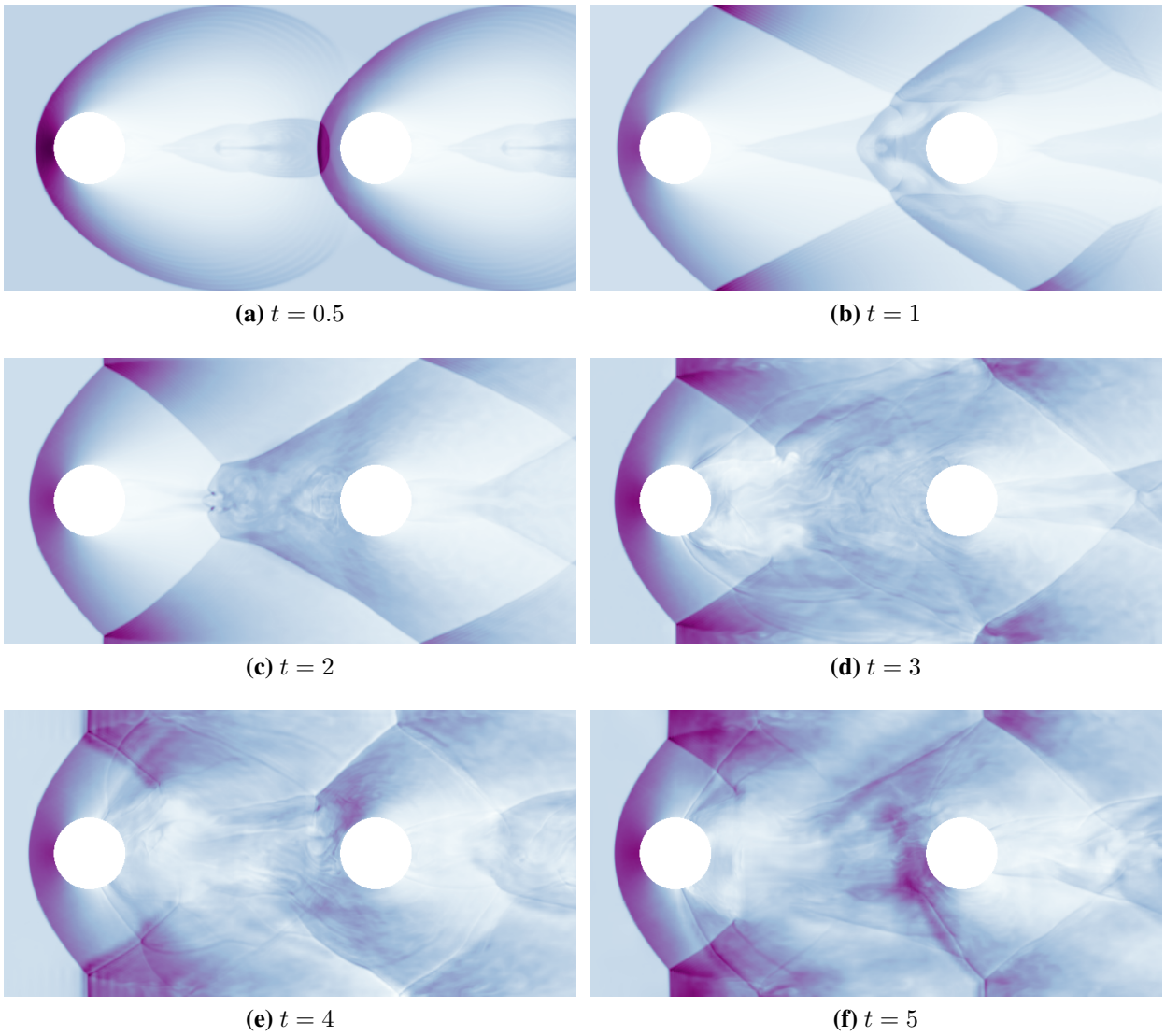
(a) $t = 0.5$ **(b)** $t = 1$

(c) $t = 2$ **(d)** $t = 3$

(e) $t = 4$ **(f)** $t = 5$

Figure 4.1: The time evolution of density in Mach 3 flow Euler Equations. Regions of darker color have higher density.

earlier times like $t = 0.5$ seconds look pretty good comparing between the exact and the PinT). For further evidence of this, see [50], where the authors design coarse operators that track characteristic curves of certain spatial modes hoping to achieve better convergence for MGRIT applied to a linear advection problem. Since brick-based PinT methods are essentially predictor-corrector, this problem in prediction is prohibitive. Many other papers have noted this issue [27, 8, 51, 46]. These issues have been known since the early 2000s, and are a main reason that PinT methods have not found widespread use.



(a) $t = 0.5$ **(b)** $t = 1$

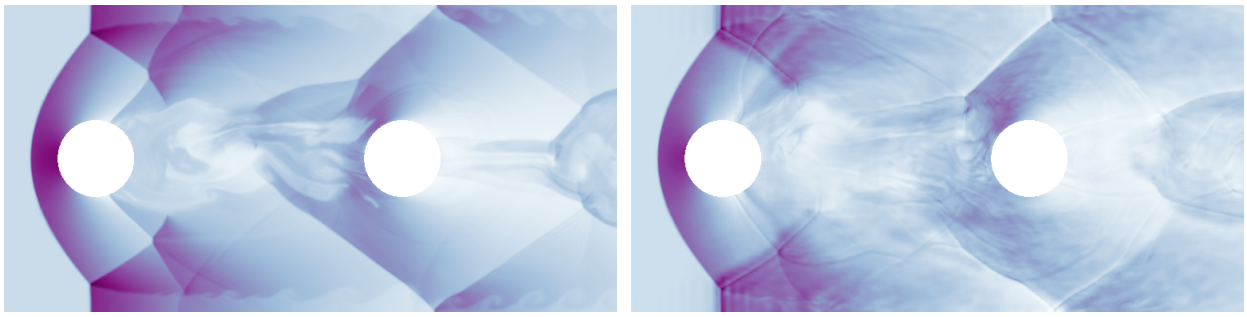
(c) $t = 2$ **(d)** $t = 3$

(e) $t = 4$ **(f)** $t = 5$

Figure 4.2: The time evolution of density in Mach 3 flow Euler Equations given by an implementation of PinT. Regions of darker color have higher density.

Conversely, highly dissipative problems (like the heat equation [6, 45, 23], or Navier-Stokes dominated by dissipation [52, 47, 27, 51]) naturally remove these higher frequency modes, and therefore we expect that PinT methods converge much better.

Multigrid PinT methods can help, especially if temporal and spatial coarsening (i.e. the coarse problems are defined on a coarse spatial mesh), because they naturally resolve modes at the scale of the mesh (see for example [13, 9, 11]). Therefore the coarser modes are likely to be well resolved in a few PinT iterations. This we also see in Figure 4.3.



(a) Exact $t = 4$ **(b)** PinT $t = 4$

Figure 4.3: Comparing PinT and Exact in an advection driven flow, we can see that certain fluid characteristics are not well resolved in the PinT implementation.

4.2 Special Modifications for PinT used in Hyperbolic Setting

It turns out that various additional considerations are needed to modify the classical PinT algorithms we have outlined so far to ensure that they have a chance at working for hyperbolic type problems. These include projections, coarsening in space and in time, and careful design of coarse propagators. In the following sections, we outline why each of these might be necessary before concluding with a "best practices" outline which we propose to follow in the novel contributions section of this thesis. We feel obliged to note that the results in Section 4.1 result from a code which implements many of these so-called fixes, but the results are still poor meaning that more needs to be done for hyperbolic settings.

4.2.1 Stabilization with Projection

Some authors note that the basic PinT algorithms may actually be *unstable* [53, 46, 52, 47], in addition to slowly converging, for hyperbolic problems. The reasons for this are many. For one, we have previously made no constraints on how the correction term behaves. To see why this may be necessary let us consider the following.

It is common for explicit time integration methods for hyperbolic PDEs to follow a constraint on the time step known as the CFL condition, restated here from (3.30).

$$dt = \mathrm{C} \max_{x \in \Omega} \cdot \frac{\Delta x}{||\mathbf{u}(x)|| + a(U(x))}. \tag{4.9}$$

Therefore, the time step is limited by the *maximum wavespeed* in the problem (the sum of $||\mathbf{u}||+a$). Often, we need models for the actual form of a, as well as make the assumption that $a \geq 0$ everywhere in our domain Ω. Commonly in fluid dynamics we model the speed of sound with a gamma law.

$$a(U) = \sqrt{\frac{\gamma\rho(U)}{p(U)}}. \qquad \text{Gamma Law} \tag{4.10}$$

For this we see that both ρ and p the pressure from (4.7) must be non-negative for this formula to give a real valued speed of sound.

What, then, do we do if our correction in Parareal given by (3.36)

$$\tau_i = F(U_{i-1}) - G(U_{i-1}),$$

overcorrects the density component of our solution state U (coming from the Euler equations, for example), resulting in a negative pressure or density? This is not unreasonable to expect might happen, since

$$U = \begin{bmatrix} \rho \\ \rho\boldsymbol{u} \\ E \end{bmatrix} \text{ and } \tau = \begin{bmatrix} \tau_1 \\ \boldsymbol{\tau}_2 \\ \tau_3 \end{bmatrix}$$

and so $\rho + \tau_1 < 0$ could happen if $\tau = F - G$ is sufficiently negative in the first component. We probably want to disallow this if our explicit method relies upon density being positive. Projection in this case is equivalent to truncation, and we can define a projection P as

$$P(U_i + \tau_i) := \begin{cases} \langle \rho + \tau_1, \rho\boldsymbol{u} + \boldsymbol{\tau}_2, E + \tau_3 \rangle_i & \text{if } \rho + \tau_1 \geq 0 \\ \langle \rho_{\text{min}}, \rho\boldsymbol{u} + \boldsymbol{\tau}_2, E + \tau_3 \rangle_i & \text{if } \rho + \tau_1 < \rho_{\text{min}}. \end{cases} \tag{4.11}$$

In other words, if the density would be negative after correction, we simply set the density to a predetermined admissible minimum density ρ_{min}. We should, in general, expect a degraded Parareal convergence, since we are limiting the size of a component of our correction.

For another, projection PinT stabilization might be relevant in the incompressible Euler equations, which model incompressible fluid flow and have add to the Euler equations (4.5) a constraint on the velocity $\mathbf{u}(x)$:

$$\nabla \cdot \boldsymbol{u}(x) = 0 \qquad \text{Incompressibility.} \tag{4.12}$$

This forces solutions to lie on a surface defined by (4.12). Therefore, we hope that the corrections from Parareal keep us on this surface, but just like for the density, this is not guaranteed, so we likely need to limit our corrections in the direction of the surface, for example by defining an orthogonal projection onto the incompressibility constraint:

$$P(U_i + \tau_i) := Proj_{\nabla \cdot \boldsymbol{u}=0}(U_i + \tau_i). \tag{4.13}$$

This sort of thing is commonly done even in serial time integration methods [54]. We may also expect degraded performance here, since we are only allowing our correction terms to land on a predetermined surface, so are necessarily truncating the corrections in many directions at once. More generally, this kind of projection might find use if the system being integrated is a **Differential Algebraic Equation (DAE)**, as these equations must satisfy an algebraic constraint for all time t.

Much literature notes that projection-based modifications to Algorithm 4 are often necessary for stability for hyperbolic regimes—[30, 47, 55, 45, 56] to name a few.

4.2.2 Coarsening in Space and Time

MGRIT itself is a multigrid in *time*, with coarser problems defined on coarser time meshes. However, when solving PDEs spatial meshes can also be coarsened. We can choose to define the coarse problem

$$A_c U_c = b_c$$

on a coarser spatial mesh along with the coarser time mesh. One obvious case where this may make sense is when using explicit time integrators on all levels. These integrators will invariably have

a stability condition like the CFL (3.30) which depends on the spatial mesh size. Consequently, the coarser time mesh would only be allowed to be so coarse. The PinT literature typically gets around this by using very expensive and high order explicit time integrators on the finest time grid, and expensive implicit time integrators on the coarser levels which can take larger time steps [24]. One can also circumvent the CFL constraint on a coarser spatial mesh.

As stated in 3.2.2, coarser spatial meshes introduce interpolation errors into the MGRIT algorithm. We expect that MGRIT is exact from the exactness property that it shares with Parareal, but with interpolation error we have no such guarantee.

Another downside of coarsening in space and time concurrently is a mismatch on the characteristic wavelengths on each level. Many authors cite the difficulty of coarse propagators in accurately tracking certain fine level waves as they propagate over coarse time step lengths [46, 45, 50]. Coarsening in space naturally disregards waves whose wavelength is smaller than the mesh size, so the coarser problem cannot hope to track the evolution of the smaller wavelength information.

Still, all indications are that resolving problems on coarser spatial and temporal meshes within the MGRIT framework is overall worth doing [9, 11]. One can conceive that since the coarser meshes resolve longer wavelength waves, that for problems where the solution U can be decomposed as a superposition of wavelengths above length L and below this length denoted l

$$U = U_L + U_l, \tag{4.14}$$

that if U_l is small, then MGRIT will have decent convergence because the coarse modes will be described fairly well by the coarser levels. Finally, the whole point of PinT is to achieve speedup. If we want to wring out all the computational performance we can, we might need to employ aggressive coarsening in time, necessitating a coarsening in space.

4.2.3 Designing Better Coarse Propagators or Corrections

Much of the convergence of PinT methods relies on how well one can design coarse propagators. This is especially true for hyperbolic PDEs where coarse propagation error dominates the error [50, 57]. Theory indicates that when coarse propagators are not designed to track certain characteristic waves then MGRIT exhibits too slow convergence.

In fact, some convergence bounds for MGRIT exist, one of which is worth restating here.

Theorem 2. *For MGRIT with F the fine grid propagator with eigenvalues $(\lambda_k)_{k=1}^n < 1$ and G the coarse grid propagator with eigenvalues $(\mu_k)_{k=1}^n < 1$, which are both diagonalizable by a unitary transformation. Assume we use FCF relaxation with a coarsening factor m and n_t time points in time, and n points in space. Then one can bound the error on the k-th mode E_k of the solution as follows [58]:*

$$||E_k|| \leq \sqrt{m}|\lambda_k|^m \frac{|\lambda_k^m - \mu_k|}{1 - |\mu_k|}\left(1 - |\mu_k|^{n_t/m-1}\right). \tag{4.15}$$

This suggests that 1) if we can design a coarse propagator so that $\mu_k \approx \lambda_k^m$ for all modes we will have better convergence, and 2) for modes $|\mu_k| \approx 1$ we have slow damping due to the term on the bottom of the middle term. The solution here would be to ensure that for those slow to decay coarse modes, we compensate by ensuring that $\mu_k \approx \lambda_k^m$ is more accurate for those modes. Therefore, [50] notes that this implies that G more accurately approximate the largest in magnitude eigenvalues of F^m. Therefore our coarse propagator needs to satisfy $\mu_k \approx \lambda_k^m$ for all modes, but more accurately so for modes $|\lambda_k|, |\mu_k| \approx 0$.

Much work has been done to try and improve coarse propagators including producing higher order corrections [10], characteristic tracking/phase corrections [57, 46], Richardson extrapolation [7], reinterpreting corrections as linear operators which retain physical quantities [33], truncation of Fourier modes [28], filtering spurious oscillations to target certain modes [55], defining asymptotic coarse problems to filter modes [59], or perhaps using artificial intelligence (AI). None of these ideas have won out as the best idea, and all have shown limited scope.

4.3 Summary

Overall, hyperbolic PDEs remain extremely difficult to solve with PinT methods. At best, convergence is slow, and at worst the problems diverge. Even with special considerations which we outlined above, it is our opinion that PinT has not been applied successfully to hyperbolic PDEs as of yet. PinT has not found a home, nor has a convincing speedup been presented in the regime, at least one that is widely applicable.

Most results that **do** show speedup in hyperbolic regimes seem to depend tightly on the problem, meaning that small variations in problem parameters can vastly change the convergence properties of PinT methods applied to hyperbolic problems [51, 19, 52].

Still, it is worth summarizing what one ought to do to have any hope of applying PinT to such PDEs more generally:

- *Use MGRIT with many levels.* Multilevel algorithms offer more parameters to tune so we have more options to get speedup.

- *Project corrections onto stable sets.* Projection is needed for stability, and though it can hamper convergence speed, at least it is stable.

- *Coarsen in space and time at the same time.* Coarsening in space and in time simultaneously allows for longer wavelength modes to be better resolved on coarser meshes, and for explicit methods on many times coarser time meshes. Be warned, however, that you may have to pay an interpolation cost for using a coarser spatial mesh to correct a finer spatial mesh.

- *Carefully design coarse propagators.* We should design coarse propagators that target the worst modes, in an appropriate sense.

Doing some combination of all of these seems the best place to start applying PinT to hyperbolic problems. But we note that none of these has sufficiently proven to speed up the convergence of hyperbolic PDEs. In the next section, we pose a related question for PinT methods which might actually converge fast enough for appreciable speedup in hyperbolic settings. We will explore

the foundational observation that enables us to derive likely quantities that converge quickly in hyperbolic PinT applications.

In the next section, We propose some derived quantities that may work, and perform some simple tests to demonstrate probable reason for study of these. We also suggest realistic statistical quantities that people may wish to compute with PinT methods.

Chapter 5

Modeling Time Averaged Quantities using Parallel in Time Integration

We have seen a large gap in the capabilities of PinT methods in their ability to accurately solve problems that are hyperbolic in nature (see Figure 4.3) at specific time points at large t. It is our opinion that these kinds of pointwise in time issues are likely to remain barriers for PinT methods. The main reason we suspect that this is the case is fundamentally due to the difficulty of designing coarse propagators which respect the long term interactions of the waveforms in these problems when sufficient dissipation is missing. Even if we could design such propagators for a specific problem with specific parameters or geometry, small changes in problem geometry or initial/boundary conditions could sufficiently change the behavior of these waves, undoing the work we did to specialize our coarse propagator for the previous form of the problem.

However, not every engineering application will be interested in averaging in time the flow field $U(x,t)$. Engineering applications might instead be interested in the time average on some part of the domain Ω. Consider an engineer who wants to know the time averaged pressure around a cylinder (like in papers [60, 61] which provide numerical and experimental averaged pressure curves on a cylinder in an airflow), or an engineer computing drag on a cylinder in Euler flow [62, Figure 2.1.3], or an engineer who cares about other time averaged quantities like the average flow structure of the wake of a cylinder in airflow [63, 64, 65, 66, 67], or an engineer wishing to understand the far field noise generated by airfoils at various angles of attack [68].

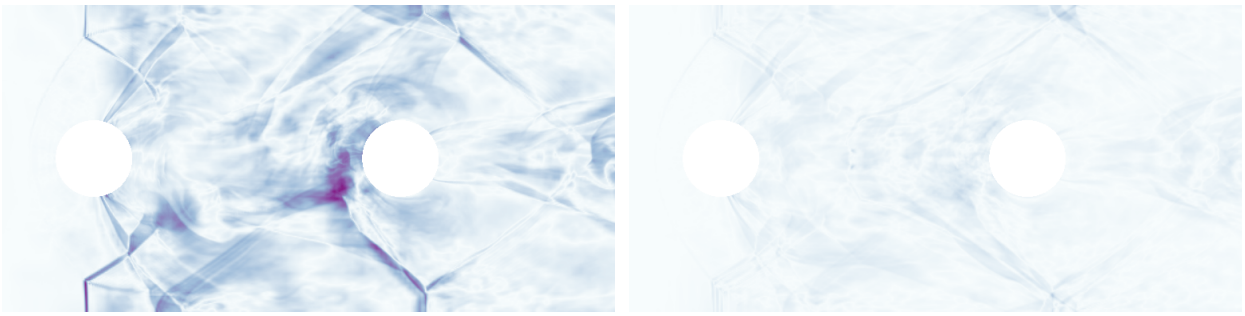
It is possible some time average information incident in U ***is*** accurately computed in a slowly converging PinT method. For example, the plots in Figure 4.3 definitely look off, but if you squint your eyes, you would be forgiven if you believed them to be converged. Squinting your eye is less than mathematical, so let us show some evidence that a time averaged quantity might converge faster than a pointwise on in a PinT method.

5.1 Convergence of Average Flow with PinT

If we compute an average state for the density across all time bricks in both the exact and PinT cases we already have shown in 4.1—see for example Figure 4.3—we notice that the differences between the two images at a specific time (Figure 5.1a) are more noticeable than the differences of the time average (Figure 5.1b). This fact, coupled with the examples of real time averaged quantities of interest by engineering applications, motivates us to study the convergence properties of quantities like

$$\overline{U} = \frac{\int_0^T U(x,t)dt}{T}, \tag{5.1}$$

as they can be computed with PinT methods.



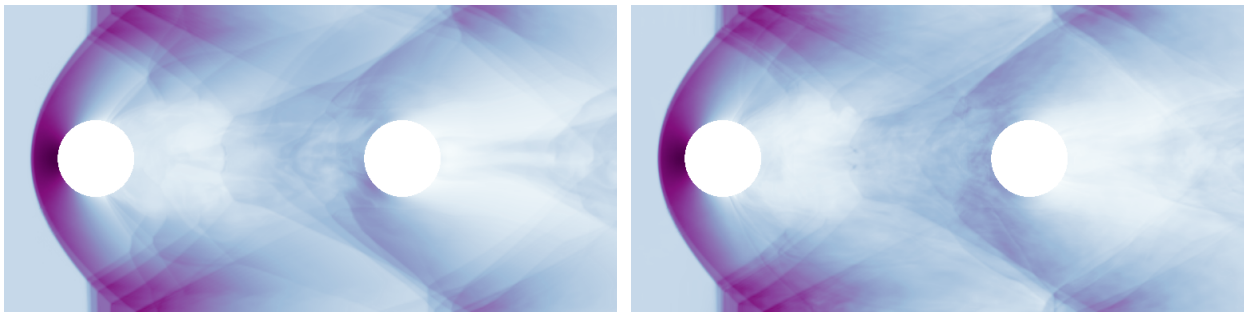
(a) Pointwise absolute difference at $t = 5$.

(b) Difference of averages.

Figure 5.1: Plotted on the same scale, we see a large magnitude decrease in the error from the pointwise in time difference in density (right) compared to when time averaging is used for the density (left). This indicates a lower error calculating $\overline{U}(x)$ versus the time dependent version $U(x, t = 5)$ in an L^∞ sense with PinT methods. Darker regions indicate places with higher error in the density.

We can phrase this mathematically by denoting the exact time average $\overline{u(x)} = \int_0^T u(x,t)dt$ and the one computed from a PinT method at iteration k

$$\overline{U^k(x)} = \int_0^T U^k(x,t)dt,$$



(a) Average in time of exact density.

(b) Average in time of PinT.

Figure 5.2: When averaging in time, the exact and PinT look much closer to the eye. This motivates calculating fluid quantities which are time averaged.

and asking how does $\overline{U}$ converge to $\overline{u}$ as $k \to \infty$. With a norm in space, we can then measure the error

$$e^k = ||\overline{u}(x) - \overline{U^k}(x)||_{L^*(\Omega)}. \tag{5.2}$$

Here, we can make a choice for the spatial norm $L^*(\Omega)$, commonly $* = 1, 2, \infty$. In the L^∞ norm, we see that the error e^k of the time average is smaller than the L^∞ norm of the solution at the end time (Figure 5.1). We need to make a quantitative study of the error. However, we can use Figure 5.1b as justification that time averaged quantities are more accurate than pointwise for PinT methods in hyperbolic problems.

5.2 Average Drag and Pressure Might Converge

While an average over the whole spatial domain Ω is demonstrated to have possibility as a PinT computed quantity in the previous section (specifically in Figure 5.1), the question of average forces over time naturally leads to an average along a boundary of our fluid flow simulation. It is not a guaranteed that because the time average state on the whole domain Ω is better than the pointwise in time on Ω that the same will be true on some sub-domain.

For example, if we care about the average drag on an body ω, where the boundary of the body is some surface $\partial\Omega|_\omega$ (thought of as part of the boundary of Ω) contained in the domain Ω of our

problem. With this, we seek to measure some quantity

$$J(U(x,t),t) := D_{\partial\Omega|_\omega}, \tag{5.3}$$

where D is the drag, which depends on the fluid state and time.

The force of drag on a body ω in a direction $\hat{\boldsymbol{e}}$ is computed by finding the overall forces $\boldsymbol{F}$ on ω and computing $\boldsymbol{F} \cdot \hat{\boldsymbol{e}}$. In the case of the Euler equations (4.1), (4.2), (4.3), the forces are found by integrating the pressure $p(x,t)$ along the boundary in question:

$$\boldsymbol{F}_\omega(t) = -\int_\omega p(x,t)\mathbb{I}_d \cdot \boldsymbol{n} dx \tag{5.4}$$

where $\mathbb{I}_d$ is the identity matrix of dimension d (the tensor $-p\mathbb{I}_d$ is the fluid stress). Assuming that $\hat{\boldsymbol{e}}_x$ is the direction of motion, the drag would be

$$D_\omega = \boldsymbol{F}_\omega \cdot \hat{\boldsymbol{e}}_x. \tag{5.5}$$

With this notation, we can modify the quantities we care about by asking how well we can compute the average of such a $J(U,t)$,

$$\overline{J} = \frac{\int_0^T D_\omega dt}{T}, \tag{5.6}$$

given by a PinT method and whether that matches the exact value from the sequential solution.

Just because we have seen in Figure 5.1 that an average state on Ω might converge in a certain way, does not mean that an average state on the object ω. In the next section, we present some qualitative evidence similar to Figure 5.2 which shows that quantities like drag and pressure computed on some object ω (think airfoil) might also converge with a time averaged PinT method.

5.2.1 Convergence of Drag and Pressure in Euler Numerical Example

For a numerical example, consider again looking at the time variable drag curve in time calculated at various cycles of basic MGRIT Algorithm 4, as computed from the Euler flow example

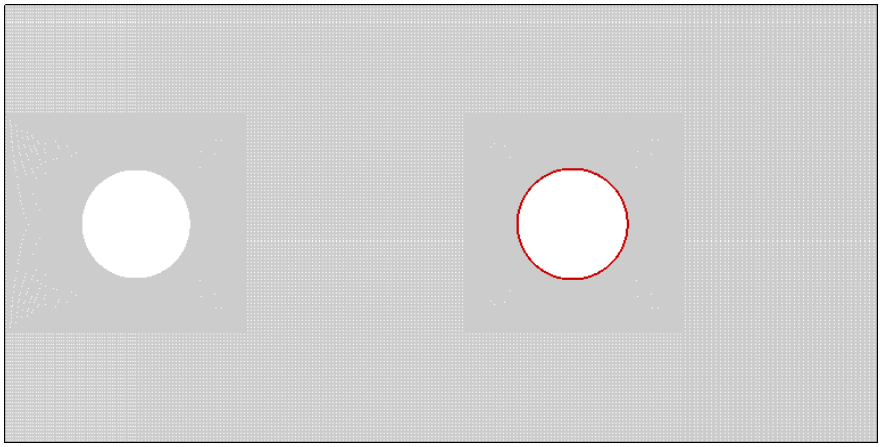


Figure 5.3: We calculate the drag on the downwind cylinder (mesh in light grey, and the downwind cylinder highlighted in red), which is in the wake of the upwind one. Perhaps a time averaged drag can be inferred from a PinT computation.

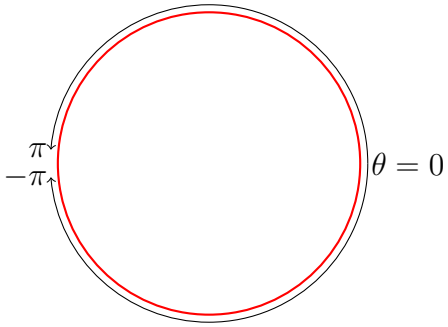


Figure 5.4: When printing solutions, we reference angles theta defined on the second cylinder (red in Figure 5.3) using the interval $\theta \in [-\pi, \pi]$. These angles are show in in this figure.

with two cylinders, see Figure 5.5. We see that after a few cycles the drag on the downwind cylinder (this cylinder is highlighted in red in Figure 5.3) in the direction $\hat{e}_x = \langle 1, 0 \rangle$, shown in Figure 5.5, is not perfect. Again, this should not be surprising since PinT cannot hope to accurately resolve the effects in the wake of the leading cylinder, which directly affect the variation of pressure along the downwind cylinder, see Figure 5.7. Since the drag is computed by integrating the pressure, we can see if the average drag is closer by looking at how the average pressure on the cylinder behaves in a PinT method.

To explore this a bit, we can examine first the time dependent resolution of the pressure $p(x, t)$ along the second cylinder. As noted in (5.4), the forces on the cylinder are given only by integrating the pressure, so we may use visualizations of pressure to estimate the evaluation of drag.

Unsurprisingly, the PinT predictions at individual far off time points are off, as shown in Figure 5.7 (note also that the differences at earlier times are zero, recall that PinT methods have an exactness property where each iteration we are guaranteed that a new brick is exactly converged). This fits with our previous study of PinT convergence in hyperbolic regimes.

Yet, when we average the pressure on the second cylinder in time, we see an improved 'convergence' to the exact time averaged pressure (Figure 5.6), just like the behavior that we saw in Figure 5.1. Note, however, that there is some numerical issue with the code, where the pressure is nonphysical at certain times making the average too high around $\theta = -0.9$ radians.

This is the sort of instability which we expect from the discussion in Chapter 4, and which we will need to deal with before we analyze using time averaging with PinT. This is done in Section 6.3. Despite this obvious issue, the time averaging looks to be somewhat promising as most of the average pressure curve is close to the exact average curve.

In Section 5.1 and Section 5.2, we outlined the plausibility that time averaging coupled with PinT might produce PinT results for hyperbolic regimes which historically do not converge fast enough for true parallel gains. In all the literature we have seen for PinT methods, to the best of our knowledge, nobody has explored time averaging for a hyperbolic problem. In the remaining chapters of this thesis, we study the performance of time averaged quantities for hyperbolic PDEs coupled with PinT. We feel that this is a question which adds to the literature.

People have asked how well PinT works for hyperbolic problems where finite speed wave interactions interact with the hope that the time dependent solution can be well known after only a few PinT iterations. It is our opinion that the literature has shown that we ought to not expect this is possible in many situations. Time averaging, however, may afford a range of situations for which we can make PinT for hyperbolic regimes work if only we ask the right question, or want to measure the right quantity.

It is not clear that time averaging affords faster convergence at all. However, we argue time averaging is a reasonable question to ask since time averaged quantities related to hyperbolic PDEs make sense to compute with PinT methods since without PinT, for large T, a sequential method

would need to integrate over the whole interval $[0, T]$ to even begin to compute $\overline{U}(x)$, while the brick based PinT algorithms can get predictions for this value immediately. PinT can be used in such a way to truncate the averaging process early.

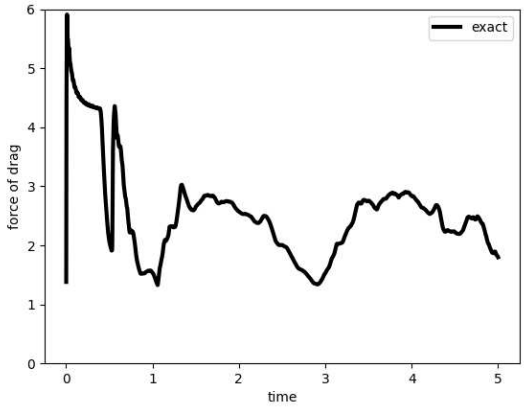
We hope that actual speedup may be achieved since convergence in a time dependent sense is sufficiently slow for hyperbolic PDEs that we might be able to beat the time to convergence with our time averaged quantities by virtue that there is a lot of time to play with to get faster time to solution. However, assessing time averaging with PinT will help to expand the usefulness of PinT methods beyond their current scope. It is my opinion that PinT methods have not found a place where they are truly useful, and I hope that this thesis will help them find this place.

In using time averagint and PinT together, we also get a benefit over other methods that produce time averaged estimations like solving steady state equations [69] or making model assumptions to help predict drag [70], since we need to make no model assumptions and we do not restrict ourselves to problems that have a steady state. On the contrary, with PinT and time averaging, we get the time average (hopefully faster) plus early time views of how our solution varies with time — after all, for short intervals of time we can notice that PinT predicts solution states well (Figure 4.2, Figure 5.7) — of systems as complicated as we want without making concessions in our models.

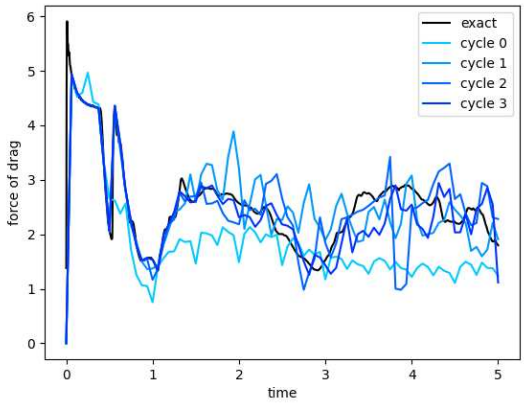
In the remaining chapters, we will to explore PinT plus time averaging in some scientific settings and over large T intervals of time to which PinT alone is ill suited, evaluate the convergence properties of such time averaging, and understand the gains which are to be found here. Two questions are the primary focus of the rest of this document.

The questions we seek to answer are:

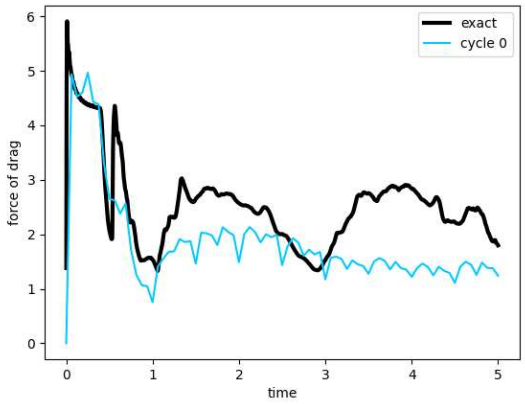
- Do PinT methods compute time averages faster than a sequential method?
- Do these time averages converge in a predictable way, or at all?



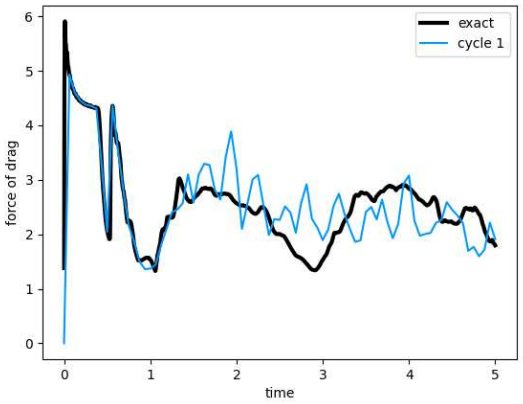
(a) Exact drag curve on the red cylinder in Figure 5.3 in a hypersonic wake.



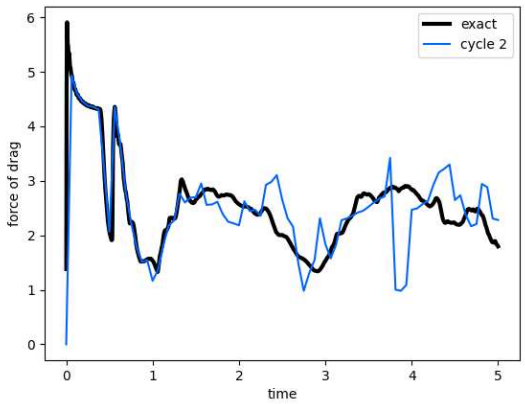
(b) Comparing exact drag to various cycles of PinT.



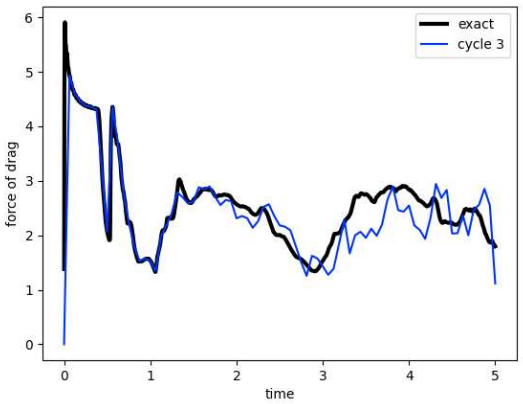
(c) Comparing exact drag to cycle 0.



(d) Comparing exact drag to cycle 1.

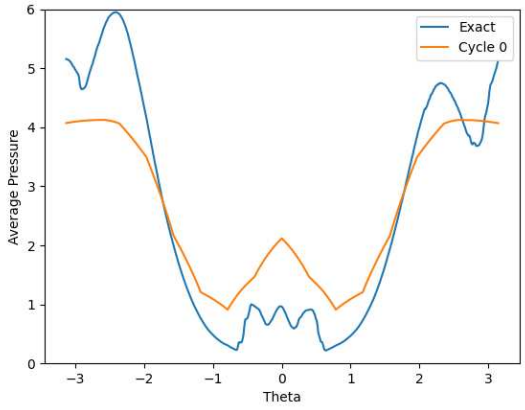


(e) Comparing exact drag to cycle 2.

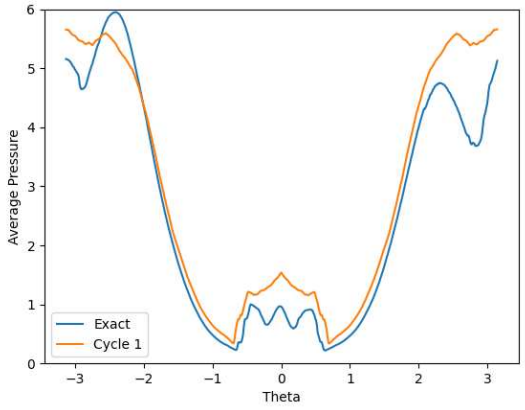


(f) Comparing exact drag to cycle 3.

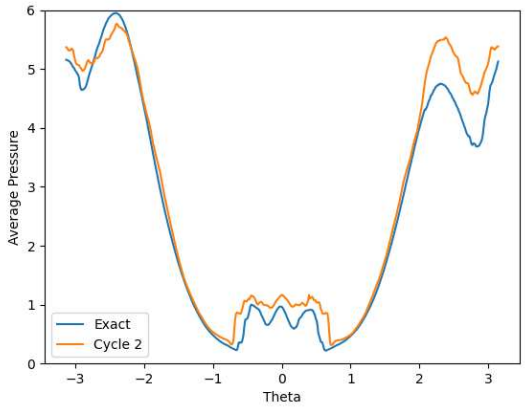
Figure 5.5: Drag curves coming from a cycle of the PinT method. Here the PinT drag curve matches some of the large scale movements of the exact curve after three cycles, but misses the smaller scale fluctuations. Overall, we see that PinT misses some of the time dependent nature of the drag after a few PinT cycles.



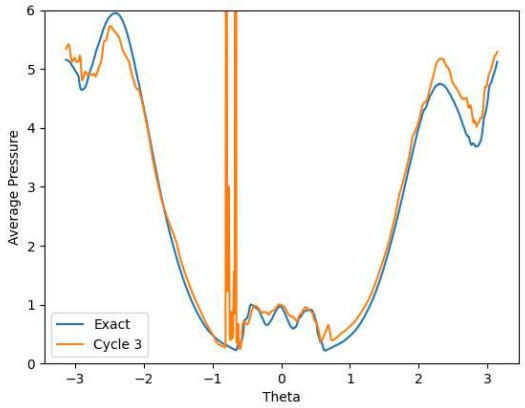
(a) Time Averaged Pressure Cycle 0.



(b) Time Averaged Pressure Cycle 1.

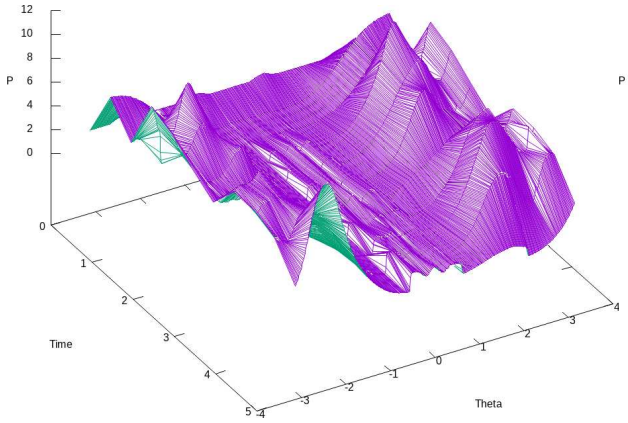


(c) Time Averaged Pressure Cycle 2.

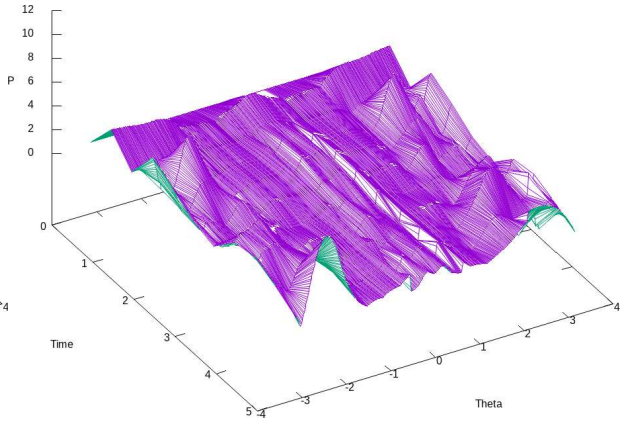


(d) Time Averaged Pressure Cycle 3.

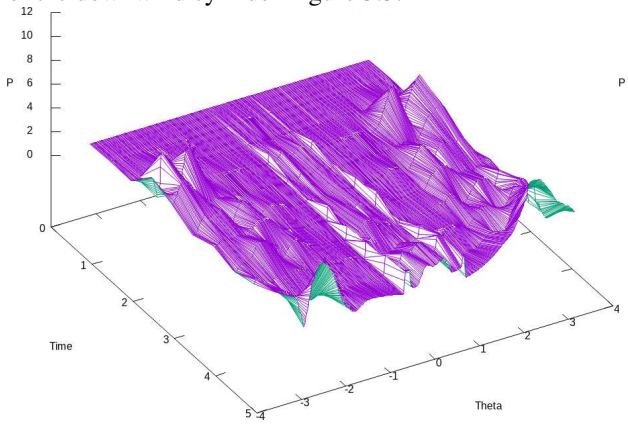
Figure 5.6: Averaging across the time dimension of Figure 5.7 seems to predict pretty well the exact time average pressure curve. Clearly there is some numerical issue in this that still needs worked out since cycle 3 shows nonphysical behavior near $\theta = -0.9$, this might be well resolved by improving the stability of our PinT method with projection (Section 4.2.1). Other θ look pretty good only after 3 iterations. Notice a large spike in the average on the last cycle 3. We will need to understand where this spike comes from to be able to apply PinT for time averaging.



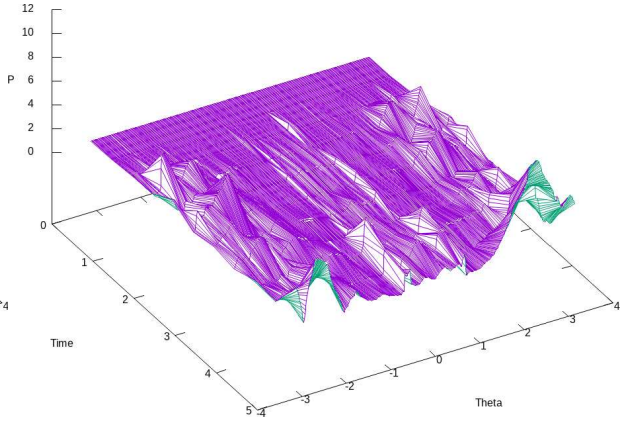
(a) The variation of pressure in time around the boundary of the downwind cylinder Figure 5.3.



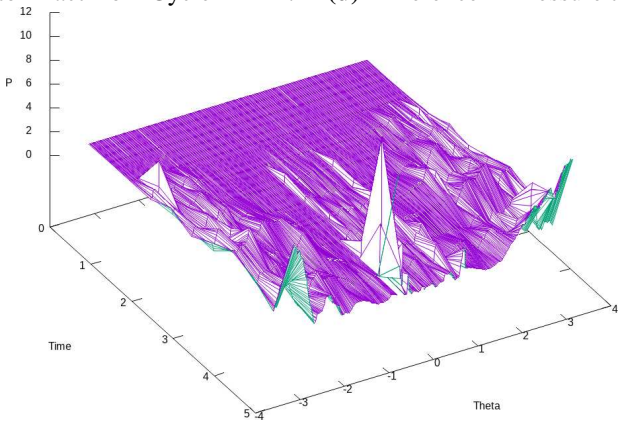
(b) Difference in Pressure to Exact from Cycle 0 PinT.



(c) Difference in Pressure to Exact from Cycle 1 PinT.



(d) Difference in Pressure to Exact from Cycle 2 PinT.



(e) Difference in Pressure to Exact from Cycle 3 PinT.

Figure 5.7: Plotting the pressure on the downwind cylinder as a surface, we see the exactness process of PinT by plotting the difference from PinT to exact, but that errors further in future are not well resolved by PinT. Further, we can see that there is some issue during cycle 3 where the pressure becomes nonphysically large.

Chapter 6

Time Averaged MGRIT Algorithm

In this chapter we introduce the MGRIT algorithm used for our study as well as the specific integration scheme used in it. Before we do this, we recall that it turns out the basic MGRIT correction can cause instabilities for hyperbolic codes, see Chapter 4. To have any hope of using PinT methods to compute time averaged quantities, we must fix this with a stabilization.

This chapter starts with defining a stable set for MGRIT in Section 6.1 a useful set onto which we can project a physically unrealistic PinT corrected state. Next, we define a specific one for the time integrator, *ryujin* [48, 49], used in this dissertation in Section 6.2. We design a projection onto the stable set defined by *ryujin* in Section 6.3 and Section 6.4. After, Section 6.5 collects the basic modifications needed for MGRIT to ensure stabilization, providing a pipeline that others can follow to do time averaging with PinT under a different integrator choice. Finally, Section 6.6 states the full stabilized MGRIT algorithm that we use in our numerical results.

6.1 Stable Sets for MGRIT

This section outlines the mathematics behind the MGRIT setup used for the hyperbolic PDE results in this dissertation. The key point to pay attention to is to use projection onto a stable set (to see why projection might be necessary, refer to 4.2.1). Projection can mean many things because a stable set can be many things. Ultimately, we want projection to ensure that our solution state after correction, $U + \tau$, can be stably integrated by the chosen fine level integrator F. Therefore, we should define some sort of stable set $\mathcal{M}_F$ that represents the collection of all states U such that $F(U)$ is defined and respects some general physical constraints (for eg. see (6.1), (6.3), (6.2)), and does not blow up or crash as computer code. As we can see, the set depends on the operator F, so each choice of operator defines a potentially new stable set.

Definition 3 (Stable Set)**.** *Let F be an integrator for a PDE. We define the* ***stable set $\mathcal{M}_F$*** *to be the collection of states U where we are guaranteed that $F(U) \in \mathcal{M}_F$ and is also stable in a numerical sense. In particular, this means that N integration steps of F, $F^N(U)$, should also be in $\mathcal{M}_F$ and that nowhere in these N repeated integration steps does $F^i(U)$ blow up for $0 \leq i \leq N$ (blow up in an appropriate numerical sense).*

The set $\mathcal{M}_F$ is our safe space. We can use any of the PinT type corrections without worry as long as we know that our corrected states lie on $\mathcal{M}_F$ we are guaranteed to produce a future state to use for future corrections. For MGRIT, with it's multiple levels and their (potentially) separate integrators, we will have many such stable sets. Utilizing the stable sets will allow us to circumvent the issues outlined in Section 4.2. Therefore, to use MGRIT for our problem of computing time-averaged quantities of interest in hyperbolic problems, we need to understand the stable set for our chosen integrators.

Please note that it's not always obvious that given any F integrator, that $\mathcal{M}_F$ should exist. Generally we should expect that this could be hard to do. However, in this thesis we use *ryujin*, which is designed to have just such a stable set. In the next section, we define this set.

6.2 A Specific Stable Set $\mathcal{M}_r$ for Ryujin

Often, integrators F designed for the sequential solution for hyperbolic PDEs come with physical constraints on parameters. For example, a common discretization technique involves solving a riemann problem to evaluate fluxes (in the finite volume method) the estimation of a maximum wavespeed to evaluate numerical fluxes which often require a sound speed calculation, which has a square root of density like in (4.10). For a review of methods that do this see [71] and for variations of these specific to the *ryujin* integrator F, see [48, 49]. FUrther, high order methods for hyperbolic PDEs often need to adapt their approximations to lower order near shocks (discontinuities) to preserve physically realistic qualities like a maximum principle (near a maximum in the solution U, we should preserve that maximum and not create another). Fundamentally, high order methods are approximations by high order polynomial interpolation, which does not generally respect such

physical characteristics. Godunov's Barrier Theorem [72] states that one indeed does have to drop the polynomial order in interpolation near shocks. All of this is to say that our methods, if not designed carefully, can over or undershoot physical limits that we expect to hold.

First, due to the square root in (4.10) numerical stability constrains density to be positive.

$$\rho(U) = \rho > 0 \tag{6.1}$$

Then, most methods, like those in *ryujin*'s implementation, also require positivity of the internal energy of the system.

$$\rho e(U) > 0 \tag{6.2}$$

Lastly, in certain PDEs, we also need that the proper notion of entropy ψ is non-negative also:

$$\psi(U) > 0. \tag{6.3}$$

The physical constraints from equations (6.1), (6.2), (6.3), define the physically correct domain $\mathcal{M}_r$ (r for *ryujin*) that the integrator F is designed to handle. We formally state this in the next section.

Definition 4 (Stable Set for *ryujin*'s Integrator [48])**.** *The stable set for ryujin, denoted $\mathcal{M}_r$, is the collection of all states U such that equations* (6.1)*,* (6.2)*,* (6.3) *hold:*

$$\mathcal{M}_r := \left\{ U \;\middle|\; \rho(U) > 0,\ \rho e(U) > 0,\ \psi(U) > 0 \right\}.$$

The authors of the method that *ryujin* uses prove that the integrator F is **invariant domain preserving (IDP)** [48, 49]. In other words if U is on the stable set, then so will $F(U)$,

$$U \in \mathcal{M}_r \implies F(U) \in \mathcal{M}_r. \tag{6.4}$$

We use this invariant domain preservation when designing our PinT method. Our corrections, just like the numerical methods of hyperbolic PDEs, need to respect physical constraints of the problem. If we do not add some method to check and fix the state $U + \tau$ we see that the constraints can be violated, again refer to Section 4.2.

Basically, we need to ensure that $U + \tau \in \mathcal{M}_r$. Then our method is stable in the sense that we know $F(U + \tau)$ exists. This assurance comes from designing a projection-like operator that can take a potentially invariant domain violating $U + \tau$ to a $P(U + \tau)$ that lies in $\mathcal{M}_r$. We outline one possible projection-like operator in 6.3. One needs to be careful in general, since the projection-like operation may not be well defined, and if it is well defined, could be hard to design in an appropriate way. Since we take the perspective that the stable set depends on the choice of integrator, one projection operator for F may not work for another integrator F', meaning that each time you change your PinT algorithm to use some different software for time integration, you will also need to re-design an appropriate projection.

6.2.1 Leave Integrator Design to Someone Else

As we can see, hyperbolic solvers need to balance physical realities with numerical desires. For example, high order reconstructions on discretized spatial domains can form oscillations near discontinuities which could force density to be negative violating physical realities that we know to be true. Striking the right balance is the basis for a huge field of research, and is hard. People have been working on this since at least the 1950's, see Godunov [72], and been a topic of continued work, see Roe's (more recent) work [73]. *Ryujin*'s authors also seek answers for this numerics and physics balance, but also add computational and implementation complexity on top because we also want our integrators F to be computed as fast as possible.

Layering a PinT scheme on top of this can only add to the difficulty. We choose to stand on the shoulders of those investing years striking the right balance in designing their integrator F. Thus, we avoid any look into the integrator F, and ask instead: What sorts of guarantees on U does F

need to produce results $F(U)$? Once we know this, we can plug F into any PinT method and be sure that we can produce some result.

As a PinT user, we only need to understand the design of the integrators F and G at a high level. We will most often plug them into out PinT scheme, so rather than digging deep into the specific integrator, we need to see the big picture of our methods so that we can weave them efficiently into a PinT method.

6.3 Designing a Stabilization for Hyperbolic MGRIT

This section deals with three questions: 1) What is the guarantee on $U + \tau$ needed to ensure $F(U+\tau)$ exists in some appropriate sense? 2) Does a projection onto the space required exist? and 3) Can we find such a projection or a suitable approximation? We will answer them in sequence, since each answer builds on the previous. Specifically, we answer these for *ryujin*'s integrators.

6.3.1 Using Ryujin as an Integrator Means that the Stable Set should be $\mathcal{M}_r$

In this dissertation, the choice of *ryujin* as a time integrating software enables an easy choice for a stable set onto which we should do a projection. Therefore, the answer to 1) is that we ought to choose $\mathcal{M}_r$ as our stable set. If we can ensure that out corrections $U + \tau \in \mathcal{M}_r$, then fact that *ryujin* is IDP (6.4) ensures that our method won't fail due to some square root of a negative number, or any other violation of assumptions used by F. Further, since we will use multilevel PinT, and define the coarse operator G only as either a mesh restriction of F, or a cheaper version of F (like swapping a third order explicit runge kutta method for an first order explicit runge kutta) we get for free that $\mathcal{M}_r$ is the correct set on each level since in either case we know that our integration method comes from the same fundamental method that *ryujin* uses.

We reiterate that to choose another integrator than *ryujin*'s is to choose another stable set. It is reasonable to assume for any such time integrator we may use, that we will be able to find an appropriate stable set since at a physical level we often have a sense of what is a physical state for our solution and what is not, while at a mathematical level we often have stability constraints

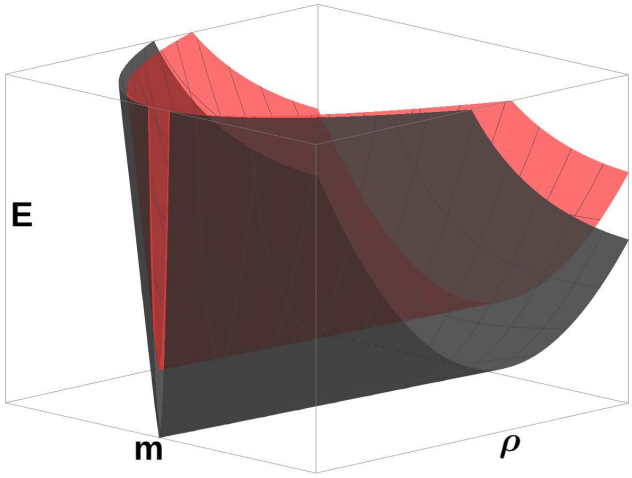


Figure 6.1: Level Surfaces of the Internal Energy $\rho e(U)$: In black is the surface $\rho e(U) := E - \frac{|\mathbf{m}|^2}{2\rho} = 0$, and in red is the surface $\rho e(U) = 1$. Since we are disallowed from projecting onto the black curve, we need to strengthen the IDP constraint for internal energy to some positive value (red). These level surfaces are like the bow of a ship. The red illustrates that strengthening the invariant domain $\mathcal{M}_r$ a bit by enforcing $\rho e(U) \geq \epsilon_e > 0$ (what the red surface indicates) rather than a $\rho e(U) > 0$ (what the black surface indicates) allows for a projection that also respects the original IDP constraints.

which define what solutions will blow up and so should be avoided. This is not to say that a stable set is always easy to nail down given a complex time integrator, only that one likely exists.

In summary, for *ryujin*, we have a stable set that drove the development of the method, so it is a natural choice for the stable set in this dissertation. The next section argues that $\mathcal{M}_r$ admits a projection that we can use in our MGRIT scheme.

6.3.2 Existence of a Projection onto $\mathcal{M}_r$

We need to be careful when examining any choice of stable set. Since we need some projection-like operator, we must also evaluate whether or not the set of choice would even admit a projection. Projections generally exist if the space is closed and convex. Since we want to sit on $\mathcal{M}_r$, we present views of each of the physical constraints that define it. The views of $\mathcal{M}_r$'s constraints in Figure 6.1, and Figure 6.2 show that if we relax the strict (never equal) inequalities (6.1), (6.2), and (6.3) which define $\mathcal{M}_r$ with a non-strict (sometimes equal) version then we get a set of states that are closed and convex. For the density constraint (6.1), it is not necessary to present an image since $\rho \geq \epsilon_\rho$ is closed and convex without any doubt.

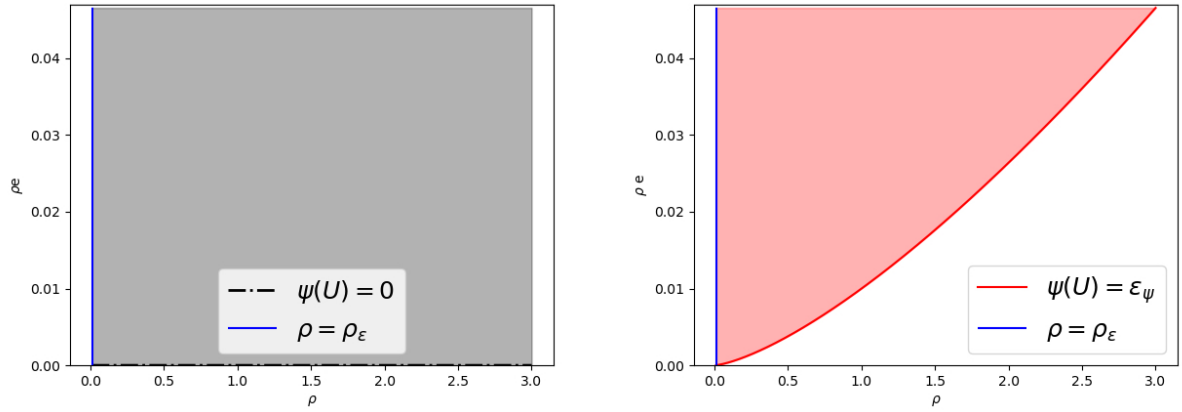


Figure 6.2: Similarly to the strengthened physicality condition of internal energy, we see that a condition on the entropy like $\psi(U) := \frac{\rho e}{\rho^\gamma} \geq \epsilon_\psi > 0$ (in red) will result in a closed and convex shape. In black we have shaded the region above curve $\psi(U) = 0$, which is disallowed for use in projection. We see that the red shaded region is a subset of the black shaded region, meaning that our strengthened IDP condition for entropy $\psi(U)$ respects the weaker condition. In blue is the strengthened ρ condition, which needs to be respected concurrently with ψ.

Since we must respect the conditions (6.1), (6.2), and (6.3), the way we strengthen the conditions is to define some minimal positive ϵ_* for each condition such that each is greater than or equal to this minimum.

$$\rho(U) = \rho \geq \epsilon_{rho} \tag{6.5}$$

$$\rho e(U) \geq \epsilon_{\rho e} \tag{6.6}$$

$$\psi(U) \geq \epsilon_\psi \tag{6.7}$$

Doing so ensures a well-defined projection operator, which is guaranteed to land within $\mathcal{M}_r$. Therefore, we will choose the strengthened constraints (6.5), (6.6), (6.7) and use them to define a stable set to use in the MGRIT scheme utilized herein.

We therefore use the following modified invariant domain in this dissertation:

$$\mathcal{M}_{r,\epsilon} := \left\{ U \;\middle|\; \rho(U) \geq \epsilon_\rho,\ \rho e(U) \geq \epsilon_{\rho e},\ \psi(U) \geq \epsilon_\psi \right\}. \tag{6.8}$$

Note that $\mathcal{M}_{r,\epsilon} \subset \mathcal{M}_r$ which ensures that, if we project onto $\mathcal{M}_{r,\epsilon}$ before integrating, *ryujin*'s integrators will work on our projected state.

6.4 A Possible Projection Operation

Since we now know that if we relax *ryujin*'s invariant domain like in Section 6.3.2, we need to design a projection operation to use in this thesis to avoid invariant domain violations during the MGRIT correction phase. Much work can go into designing a projection operation that respects important physics, for example like some sort of conservative local averaging [74].

We take the perspective that the corrected state $U + \tau$ is only minimally away from $\mathcal{M}_r$, so rather than designing a projection that minimizes the distance between $U + \tau$ and $\Pi(U + \tau)$ in an L^2 sense, we will truncate the correction with locally defined minimums for the density, internal energy, and entropy.

The motivation for a simpler type of projection comes from some numerical evidence that we observed during the code development where the main issue would be a small negative density or small violating internal energy. This small internal energy is just barely off of $\mathcal{M}_r$, so walking along grid directions instead of on a minimal length projection introduces a small amount of error. Figure 6.3 presents a helpful representation of this idea.

With this image in mind, and knowing that a projection exists we now outline the exact projection operation used in this thesis. In short, any degree of freedom which violates $\mathcal{M}_r$ is set the values to some pre-determined values. The averaging, as alluded to earlier, is not only reasonable computationally (we only need to look at a few DOFs during projection), but is also guaranteed to be invariant domain preserving (IDP) when the nodes in the stencil are all within $\mathcal{M}_r$. The exact algorithm used is in Algorithm 5.

In Algorithm 5, both density and internal energy bounds are enforced explicitly, so we know that (6.5) and (6.6) are satisfied. The entropy ψ is set implicitly in Algorithm 5, because in certain cases we have guarantees that if $e(U) > 0$ and $\rho > 0$ then $\psi(U) > 0$. For example, in the form of

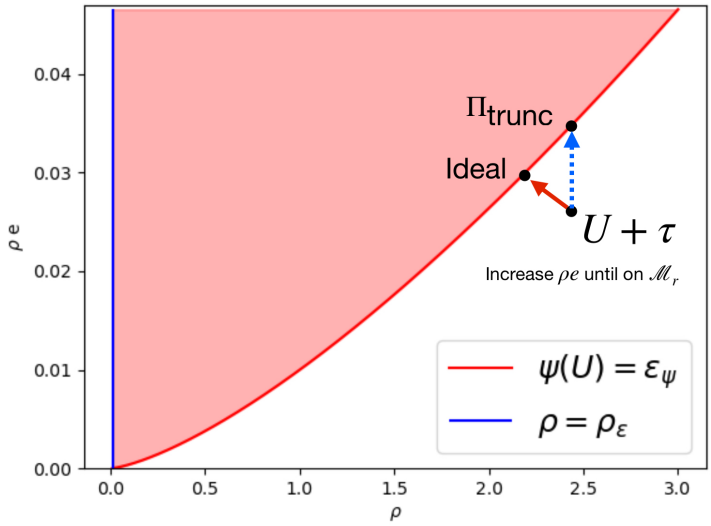


Figure 6.3: A view of the invariant domain for internal energy $\rho e(U) \geq \epsilon_e$ shows that for internal energies which are minimally IDP violating, the difference in a L^2-type ideal projection (red arrow) and increasing a component in a coordinate direction (here, we increase e) until we land on the constraint (blue arrow) might not be large. This justifies the use of Algorithm 5 in a modified MGRIT algorithm for hyperbolic PDEs.

Algorithm 5 Truncated IDP Projection Π_{trunc}

Given user defined reference density ρ_{ref}, internal energy ρe_{ref}, and scaling factor α, we compute minimum allowable density $\epsilon_\rho = \alpha \cdot \rho_{\text{ref}}$ and minimum allowable internal energy $\epsilon_{\rho e} = \alpha \cdot \rho e_{\text{ref}}$. Then we limit densities and energies deemed too small.

for all nodes $i \in \mathbb{T}$ **do**
  if $\rho(U_i + \tau_i) < \epsilon_\rho$ **then**
    Set the density $\rho(U_i + \tau_i) = \rho_{\text{ref}}$.
  end if
  if $e(U_i + \tau_i) < \epsilon_e$ **then**
    Set the internal energy of the node $\rho e(U_i + \tau_i) = \rho e_{\text{ref}}$.
  end if
end for

Overall, this operator maps potentially violating states to our stable set:

$$\Pi_{\text{trunc}} : U + \tau \to \Pi_{\text{trunc}}[U + \tau] \in \mathcal{M}_{r,\epsilon}.$$

---

the specific entropy (4.8) in this dissertation

$$\psi(U) = \frac{\rho e}{\rho^\gamma},$$

just such a guarantee is met. Probably a minimum value of $\epsilon_\psi = \frac{\epsilon_{\rho e}}{\epsilon_\rho^\gamma}$ would accomplish the same as our implicit minimization.

We note that the reference values and the scaling factor are chosen y. A reasonable first choice might be to select be a density ρ and internal energy e based on the initial condition on the finest level, and $\alpha = \frac{1}{100}$. Future work should look at choosing these coefficients smarter. Other options for projection exist, see [74] which defines an IDP and conservative projection for use in adaptive mesh refinement. We use Algorithm 5 for it's simplicity and ease in programming. Then, all we need to do to have a stabilized MGRIT to use with hyperbolic integrators is to add in a projection operation after every correction step. In the next section, we outline how to inject a projection operation into PinT methods to ensure

6.5 Pipeline for Ensuring that PinT τ-Corrections lie on $\mathcal{M}_r$

Under the assumption that F A possible PinT pipeline to ensure physicality of our corrected state could be as follows:

1. Allow the PinT corrections to be computed in whatever way the normal MGRIT method would do so.
2. Apply the corrections normally. i.e. $U + \tau$, then project them $\Pi(U + \tau)$.
3. Replace the potentially unstable step $F(U + \tau)$ with the projection stabilized step $F(\Pi(U + \tau))$ anywhere we take a step. This includes steps on any level of MGRIT.

If one looks at Algorithm 7, we simply replace anywhere where an integrator F or G takes a step, and pre-compose with Π. This is enough, in the context of stable sets, to ensure that MGRIT does not break.

6.6 The Full Algorithm for Stabilized MGRIT+Π

In this section, we present the full computational algorithm used for the results in this paper. We call this method **MGGRIT+Π**. The method is basically MGRIT, but we 1) select a specific startup procedure and 2) add a projection operation. First, we introduce the startup procedure used

in this dissertation. Then we write out the modified MGRIT algorithm which adds projection. Finally, we discuss different ways to our problem.

6.6.1 Startup Procedure

The first step in a PinT method is to specify the initial conditions on each brick. In this section, we outline the options that we can choose, and we write a startup procedure that we use in the Stabilized MGRIT algorithm, see Algorithm 6.

Many PinT papers test their method by initializing bricks with random initial conditions [12, 24, 7]. We do not take this perspective since we need our solution to be physical. We are not sure of a way to randomly initialize a physically relevant flow in any meaningful way. Other papers use the coarse G to initialize each brick [27, 22]. We take this perspective since this is basically how we introduced PinT methods in Section 3.1. The specific way we choose a coarsest problem varies. Section 6.6.3 outlines two ways that we can define a coarsest problem.

On a technical level, the startup procedure in this dissertation is sequential and happens as a pre-processing step. The MGRIT+Π code simply reads output information from another code. What this means is that Algorithm 6 exists as a separate code from the code that implements Algorithm 7.

Algorithm 6 Startup Algorithm

---

1. Given a choice of coarse integrator G, an initial condition $U_{\Delta,0}$ and a set of N bricks

   $$\{B_i : i \in \{0, \ldots, N-1\}\}$$

   perform the sequential time integration

   $$U_{\Delta,i} = G(u_{\Delta,i-1}) \text{ for } i \in \{1, \ldots, N-1\}$$

   to set the initial condition for every brick B_i.

2. Save the initial conditions to a file to be read in later by the MGRIT code.

6.6.2 MGRIT+Π Algorithm

We write down the full MGRIT+Π algorithm (Algorithm 7) in this section. As a first step, we read in the data generated by our choice of Algorithm 6. Then, we begin our MG V-cycle with the specified number of levels and coarse problems. Overall, there is only a small change made between MGRIT+Π, and regular MGRIT. Mostly, the algorithms are identical. The only change follows the outline in 6.5, we replace every $F(U+\tau)$ in MGRIT with $F(\Pi(U+\tau))$ in MGRIT+Π to ensure stability. This replacement is highlighted in red in Algorithm 7.

In Algorithm 7, the notation $A \circ \Pi$ is meant to indicate the replacement of F in the matrix A defined in Section 3.2.1 with $F \circ \Pi$. In other words, MGRIT+Π makes the replacement on each level l

$$A_l = \begin{bmatrix} I & & & \\ -F_l & I & & \\ & \ddots & & \\ & & -F_l & I \end{bmatrix} \rightarrow A_l \circ \Pi = \begin{bmatrix} I & & & \\ -F_l \circ \Pi & I & & \\ & \ddots & & \\ & & -F_l \circ \Pi & I \end{bmatrix} \tag{6.9}$$

for the time integration matrices on the down cycle. We also replace the up cycle steps by modifying the correction stage with a projected correction

$$U_l^{k+1} = U_l^k + P(\epsilon_{l+1}^k) \rightarrow U_l^{k+1} = \Pi[U_l^k + P(\epsilon_{l+1}^k)]. \tag{6.10}$$

MGRIT+Π is the algorithm used to generate time averaged data in this thesis. Note that by its structure, MGRIT+Π is a V-cycle. We do not consider other cycling schemes in this (recall the F-cycle Figure 3.5 is another option). However, the replacements (6.9) and (6.10) can be made in any MGRIT cycling scheme. See Section 8.3 where we comment that other cycling structures are a future research direction.

Algorithm 7 Stabilized MGRIT for Hyperbolic Integrators (MGRIT+Π)

Iteration 0: Read in initial guesses on all bricks B_i using data generated from Algorithm 6.
Using Π defined in Algorithm 5, perform the loops below.
for Iteration $k \in \{1, \ldots, K\}$ **do**
  Down Cycle:
    for Level $l = 0;\ l \leq L-1; l = l+1$ **do**
      if $l \neq L-1$ **then**
        Solve $A_l \circ \Pi U_l^k = g_l$ using $FC-$ relaxation (integrate bricks)
        Compute and restrict the residual at C-points

$$g_{l+1}^k = R\big[g_l^k - A_l \circ \Pi U_l^k\big]$$

      end if
      if $l = L-1$ **then**
        Solve $A_l \circ \Pi U_l^k = g_l$ exactly (sequentially) for the error

$$\epsilon_l = U_l^k - R[U_{l-1}^k]$$

.
      end if
    end for
  Up Cycle:
    for Level $l = L-2;\ l \geq 0;\ l = l-1$ **do**
      Correct using prolongation of coarse error:

$$U_l^{k+1} = \Pi[U_l^k + P(\epsilon_{l+1}^k)]$$

    end for
end for

6.6.3 Coarsening in MGRIT+Π

Now that we have defined the MGRIT+Π structure, we need to get specific about the way in which we construct the coarser problems. Note that we have already selected the fine integrator F to be that from *ryujin*. We have already alluded to the fact that the coarse problems F_l or G (depending on which section of this dissertation you read) can be nearly anything we want Section 3.4. One should use more discretion when selecting a coarse problem than that. It seems to me that there are probably two main ways that we'd want to coarsen a problem for MGRIT+Π.

Spatial Coarsening

The first is to define a coarser problem by using a related but coarser spatial mesh. If $\mathbb{T}$ is our mesh on the finest level, and we assume that it is the result of many global spatial refinements, then we already have a natural hierarchy of meshes $\{\mathbb{T}_l\}$ where each $\mathbb{T}_l \subset \mathbb{T}$. Then, our coarse problem has an integrator F_l which operates on the coarse triangulation $\mathbb{T}_l$. Coarsening this way has the added benefits of how coarsening globally in the space dimension can vastly reduce the number of DOFs in the coarse problem. In 2D, a single global coarsening results in a quarter of the DOFs as the fine level. However, one should note that they are introducing an interpolation cost between corrections on the coarse level and using them on the fine level, see Section 3.2.2 for how this affects MGRIT.

This can be thought of an anisotropic coarsening of the space-time grid (see a view of these as they split the domain among processors in Section 2.1) only in the direction of space. Note that for the explicit methods we might use, the CFL condition (3.30) also tends to coarsen the time dimension, but only because for hyperbolic PDEs the space and time mesh depend on one another. Therefore we term spatial coarsening a **Semi-Coarsening**, because it explicitly coarsens only the space mesh, but implicitly does both space mesh and time mesh.

Note that fully space-time MG methods, referenced in [2, 16], coarsen both space and time simultaneously, because they view space-time with a single mesh. In our case, coarsening space will implicitly coarsen time due to CFL-like stability constraints.

If we choose a semi-coarsening in space as the way we define coarse problems, we will assume that the fine mesh came from successive global refinements of some reference mesh $\mathbb{T}_0$. Then, we will denote our coarse problems with the number of global refinements of the reference mesh needed to create our mesh. We will call this type of coarse problem *r-type* coarsening.

Definition 5 (r-type Notation)**.** *Given a reference coarsest mesh $\mathbb{T}_0$, we will define our MGRIT hierarcy (and hence the coarse problems) by specifying how many global refinements from the*

reference are needed to create the desired coarse or our fine mesh. We will use the notation

$$ra_0a_1 \dots a_f \tag{6.11}$$

to denote this. Here, $a_f \in \mathbb{N}$ *is the number of refinements of* $\mathbb{T}_0$ *to result in the fine mesh, and* $a_i \in \mathbb{N}$ *is the number of refinements for the coarse level* $l = f - i$*. For example,* $r346$ *means the the coarsest level is created three repeated global mesh refinements of* $\mathbb{T}_0$*, the middle coarse level is created from one mesh refinement of the coarsest level, and the fine level two more mesh refinements from the middle. This notation also encapsulates the number of MGRIT levels. In the case of* $r346$ *we can see that there are three levels (here also note that we index from zero, so* $f = 2$ *meaning* $l = 2 - 2$ *is the finest level, while* $l = 2 - 1$ *is the middle, and* $l = 2 - 0$ *is the coarsest).*

Integrator Coarsening

The other choice for coarsening involves what we will term **integrator coarsening**. This type of coarsening involves using the same spatial mesh, hoping to avoid those interpolation costs incurred between different spatial meshes. In this case, the only way we can make our coarser levels cheaper is to loop over the number of DOFs in $\mathbb{T}$ fewer times. The way we do this is by changing the integration scheme on each level. If, for our fine level we use an explicit third order method in time, then perhaps we use an explicit first order method on the coarser level. We did something like this already in Section 3.1 where we present Parareal. This type of coarsening is cheaper because a third order method will require three evaluations of the right hand side of an IVP, while a first order method only requires one. Therefore we get a reduction of the number of DOFs we have to visit by a factor of three. This basically redefines F_l on each level by how many sub-steps of F_l are required for one full step, but F_l operates on the same mesh on every level. We will call this type of coarse problem *i-type* coarsening.

Definition 6 (i-type Notation)**.** *Given a mesh* $\mathbb{T}$ *which we use on each level of MGRIT, we will define our MGRIT hierarchy in integrator coarsening by specifying the order of accuracy in time*

each level uses. We will use the notation

$$ia_0a_1 \dots a_f \tag{6.12}$$

to denote this. Here, $a_f \in \mathbb{N}$ *is the order of accuracy for the integrator on the fine level, and* $a_i \in \mathbb{N}$ *is the order of accuracy for the integrator on coarse level* $l = f - i$*. For example,* $i123$ *means the the coarsest level uses a first order method like Explicit Runge Kutta 1, the middle uses a second order method like Explicit Runge Kutta 2, and the finest level uses a third order accurate method like Explicit Runge Kutta 3. Note we also know how many levels we have by looking at the shape of* $i123$*, where we identify three levels.*

Obviously, we don't have a zero order method, so we might expect that we will have a limit on the number of coarse levels while using integrator coarsening. For example, if the fine level that you want to use has a third order integrator, then the only MGRIT hierarchies we can get are i123, i13, and i23. MG methods shine when possibly many more levels can be used, so integrator coarsening might not be right.

Furthermore, while this type of coarse problem definition is good at avoiding the interpolation cost, it might introduce dispersion errors. Since out problems in this dissertation typically have shocks, the dispersion errors could result in different shock locations on every level of MGRIT, which might ultimately cause similar problems to those from interpolation error.

Which Type of Coarsening to Choose?

It is not clear which type of coarsening will be better. Therefore we will explore both and compare their results in a few numerical examples in Chapter 7. Ryujin offers easy specification of the mesh and integrator type we use on each level, so we were able to try each out separately and compare results.

One can also ask if the types of coarsening can be combined. The answer is yes, but we do not explore this. See Section 8.3 where we pose this as a future direction for research. In the next chapter, we apply MGRIT+Π to two test cases. First, we analyze MGRIT+Π under different

coarsening strategies in the same two cylinder setup we presented in Section 4.1. We analyze the results for the best performing strategy, and comment on overall trends and observations. Then, we select the best performing strategy and apply MGRIT+Π to a more realistic setting where we simulate an airfoil in landing conditions and compute the time average drag it experiences.

Chapter 7

Applications of Time Averaging MGRIT

In this chapter, we apply the algorithm defined in Section 6.6 for stabilized MGRIT with time averaging to some testcases and perform convergence analyses. We first apply the method to a 2D Mach 3 flow around two cylinders. Then, we take the best performing MGRIT+Π hierarchy (in terms of relative error and MGRIT cycles) from the two cylinder case and apply it to an airfoil in landing configuration and measure the time average pressure and drag on the airfoil.

Section 7.1 presents an analysis of MGRIT+Π applied to computing a time averaged pressure and drag around one of two cylinders. This analysis allows us to gain some insight into two key qualities we want to understand about our method. First, we will be able to tell if our method converges at all, and if not, what is the error we should expect. Second, we will be able to tell approximately how many MGRIT+Π cycles are needed to reach the final state.

Section 7.2 takes the number of cycles and best performing MGRIT+Π coarsening schemes from the two insights gained in Section 7.1 and applies it to a realistic example of an airfoil in landing configuration. This allows us to fairly evaluate the performance of MGRIT+Π and compare the coarsening schemes.

Finally, Section 7.3 presents a table of execution time for each numerical test compared to a sequential (not PinT) calculation for time averaged quantities of interest. We make observations about the performance of MGRIT+Π using these timings. Recall that the whole conception of PinT rested on the potential decreased time to solution afforded by parallelizing in the time dimension (see Chapter 2 for a refresher). We seek to evaluate whether we have gained any speedup using MGRIT+Π to compute our time averaged quantities.

7.1 Cylinder in Mach 3 Euler Flow Wake

In this section, we present computed time averaged pressure curves and drag on the downwind cylinder using MGRIT+Π and both spatial and integrator coarsening schemes. First, we look at

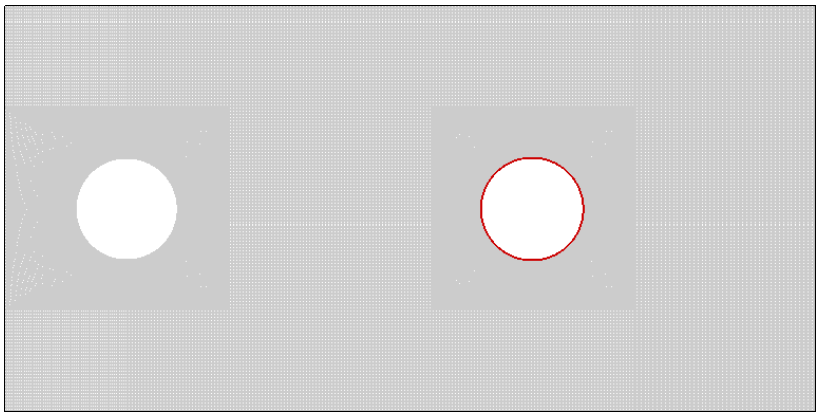


Figure 7.1: We will calculate the pressure and drag on the downwind cylinder (mesh in light grey, and the downwind cylinder highlighted in red), which is in the wake of the upwind one. Perhaps a time averaged drag can be inferred from a PinT computation. Restated from Figure 5.3.

r-type MGRIT+Π which defines the coarse levels by coarsening the spatial mesh. Then we look at two different i-type MGRIT+Π which defines coarser levels with "integrator coarsening" where coarse levels use less accurate time integration methods, without changing the mesh. We also present a direct comparison between how each MGRIT+Π hierarchy started and ended, as well as a direct comparison of the relative errors.

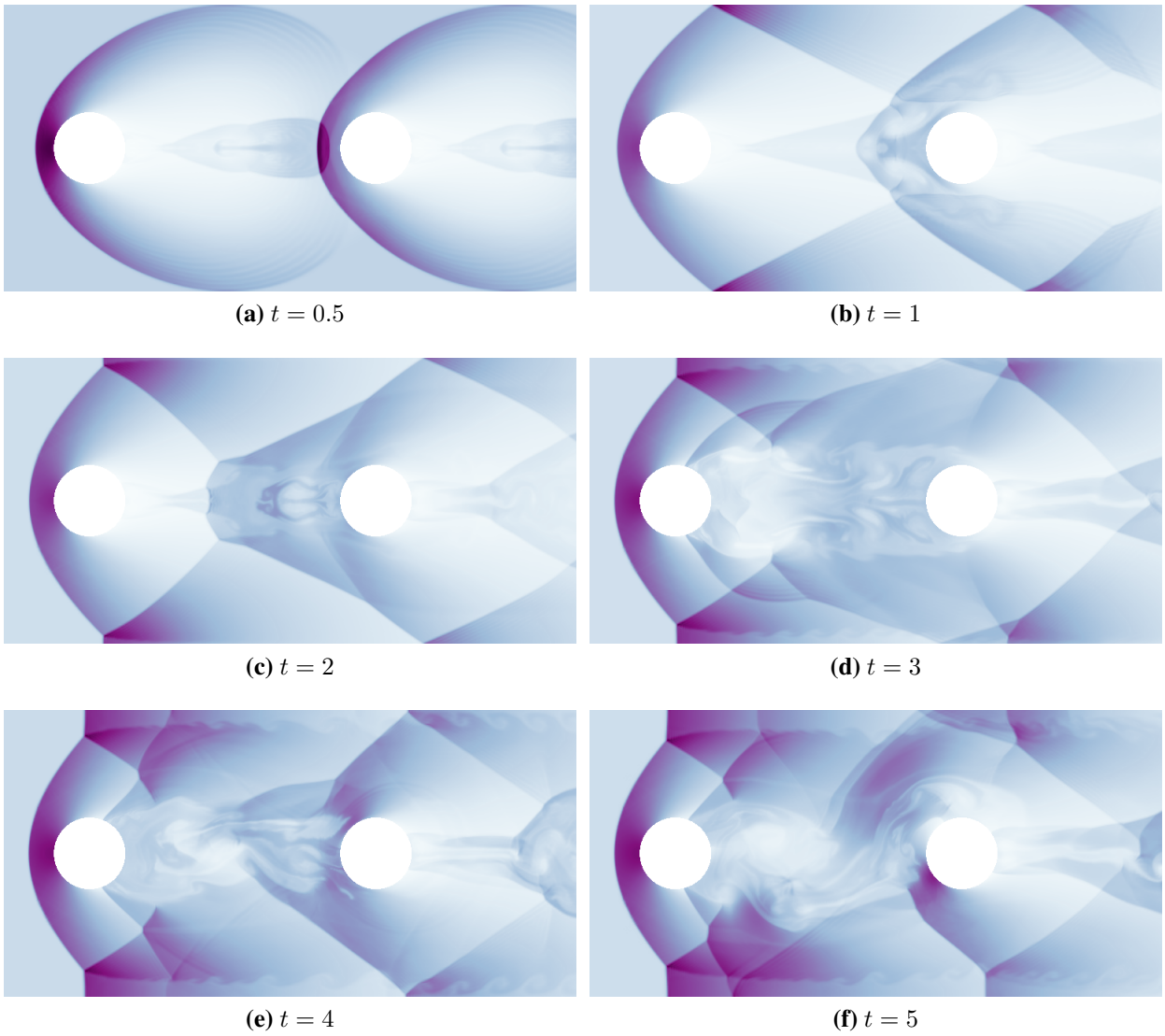
7.1.1 Problem Definition

We solve a problem which involves two cylindrical objects in a tube with air flowing through it at Mach 3. We previously mentioned this in Section 4.1. We restate the geometry in Figure 7.1 and what the solution looks like in Figure 7.2. Here, we specify in more detail the initial and boundary conditions, the exact data, and the mgrit methods chosen for the study.

Initial and Boundary Conditions

The initial condition for the exact and PinT testcases set the density, total energy and momentum at all positions to

$$U(t=0) = \begin{bmatrix} \rho_0 \\ \rho_0 \boldsymbol{v}_0 \\ E_0 \end{bmatrix} = \begin{bmatrix} 1.4 \\ 1.4 \cdot [3,0]^{\mathrm{T}} \\ 8.8 \end{bmatrix} \tag{7.1}$$



(a) $t = 0.5$ **(b)** $t = 1$

(c) $t = 2$ **(d)** $t = 3$

(e) $t = 4$ **(f)** $t = 5$

Figure 7.2: The time evolution of density in Mach 3 flow Euler Equations. Regions of darker color have higher density. Restated from Figure 4.1

where the state is consistent with a Mach 3 flow. In *ryujin*, this is known as uniform initial conditions.

For the boundary conditions, we categorize each edge in Figure 7.1. We have an inflow condition on the left edge $\partial\Omega_{\text{left}}$ of the domain. This condition is a Dirichlet condition which enforces

the same data as the initial condition:

$$U_{\partial\Omega_{\text{left}}} = \begin{bmatrix} 1.4 \\ 1.4 \cdot [3,0]^{\mathrm{T}} \\ 8.8 \end{bmatrix} = U(t=0). \tag{7.2}$$

On the right edge of the domain $\partial\Omega_{\text{right}}$, we prescribe an outflow condition. This condition is the equivalent of not doing anything specific in the weak formulation. This type of boundary condition is only valid when the degrees of freedom at the boundary have a velocity vector pointing out of the domain. For high speed flow, this is certainly the case at the downwind edge $\partial\Omega_{\text{right}}$.

The rest of the edges, those at the top, bottom and both surfaces of the cylinders (which we will call $\partial\Omega_{\text{slip}}$), we prescribe a slip condition. This enforces a vanishing normal component of the momentum, essentially enforcing that fluid cannot pass through the boundary. This condition is equivalent to enforcing

$$\rho\boldsymbol{v}(U(x,t)) \cdot \boldsymbol{n}(x) = 0 \tag{7.3}$$

for all $x \in \partial\Omega_{\text{slip}}$ and where $\mathbf{n}(x)$ is the normal vector to the surface. In (7.3), the notation $\rho\boldsymbol{v}(U(x,t))$ represents the momentum $\rho\boldsymbol{v}$ of U at a particular place in space and time (x,t).

Start-up Time

Setting a uniform initial condition everywhere is physically unrealistic, since it means that the two cylinders suddenly appear in a Mach 3 airflow. Therefore, we will measure all of our time averaged quantities after a "startup time" when the time varying behavior begins. To do this, we measure the drag on the downwind cylinder and select the start-up time past the initial transition period and where we observe the flow to oscillate, see Figure 7.3. In this case, we will measure all of our time averaged quantities from the time $t = 1.0$ since Figure 7.3 indicates that this is about the time that we enter into a more realistic flow regime than our initial conditions. We can tell this from Figure 7.3 because of the large spikes in the exact drag curve in the interval $t \in [0,1]$. After $t = 1$, the drag behaves more smoothly. Therefore we will toss out all data before $t = 1$.

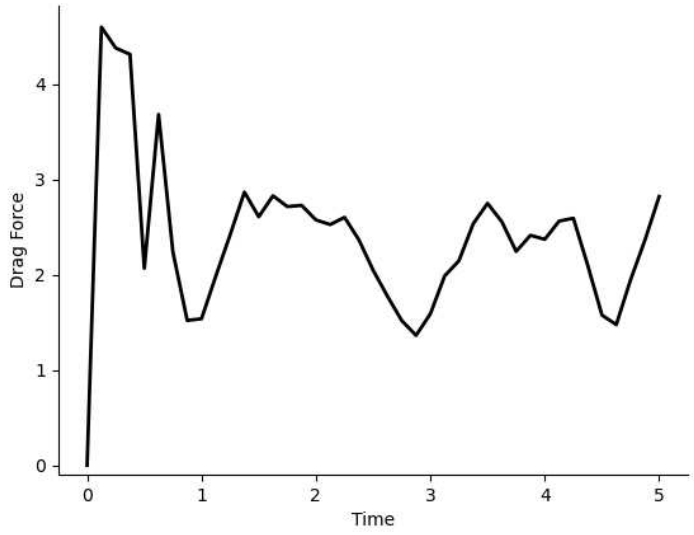


Figure 7.3: The exact drag curve, given in the natural units for the computation (non-dimensionalized Euler equations), shows us that to really measure a realistic setting we need to begin our measurements from a start time of $t = 1.0$, approximately. Around this time, the drag fluctuations behave in a more regular way. In the first second, the drag shoots to large numbers before dropping to within the range that it oscillates within for the remainder of the simulation.

MGRIT+Π Hierarchies

We test r456, i13, and i123 MGRIT+Π coarsening schemes. We compute the L1 norm for the pressure curves on the second cylinder, using 40 bricks, on the time interval $[1, 5]$. The MGRIT+Π hierarchies were chosen with between two and three levels. We favored three levels since we already knew that two level MGRIT is equivalent to Parareal, and we wanted to expand our studies to multi-level PinT where we expect that PinT methods to enable more speedup. For the spatial coarsening type PinT, we chose r456 because using higher accuracy refinements on the coarser levels was found empirically to give better corrections during initial testing. We chose i13 and i123 because if we use a third order time integrator on the fine level, the two prior hierarchies are the only available MGRIT+Π hierarchies for the integrator coarsening schemes. In the following section, we present an analysis of the problem with the setup we just detailed and using the hierarchies just detailed. We begin with a study of time averaged pressure on the downwind cylinder, and then perform an analysis on the drag computation.

Exact Data

For the exact data, we use a spatial mesh with 6 global refinements from a reference mesh, and integrate on the time interval $[0, 5]$ with 40 bricks, and using third order Runge-Kutta for the time integration. Figure 7.2 illustrates what the exact solution looks like in the case we present with matching initial and boundary conditions. We see an initial phase where bow shocks form early on, followed by vortices forming and interacting in the wake of the upwind cylinder. Regions of high density form on the leading edge of the downwind cylinder. The high density region oscillates from the top to the bottom of the downwind cylinder. Therefore, we expect high pressure in those two regions when we compute time average data. Indeed we do see high pressure near those regions in Figure 7.6, Figure 7.7, and Figure 7.8.

7.1.2 Pressure on Cylinder

In this section, we measure the difference in a time averaged pressure curve on the downwind cylinder. We compute a relative error. Let $\tilde{P}(\theta)$ be the time averaged pressure along the cylinder defined in Figure 7.4 where we average the pressure curve by computing

$$\tilde{p}(\theta) = \frac{\int_1^5 p(\theta, t)dt}{4}, \tag{7.4}$$

and we compute the pressure inside the integral with (4.7) given by

$$p(\theta, t) = (\gamma - 1)\left(E(\theta, t) - \frac{\rho(\theta, t)\boldsymbol{m}(\theta, t) \cdot \boldsymbol{m}(\theta, t)}{2}\right), \tag{7.5}$$

where $E(\theta, t)$, $\rho(\theta, t)$ and $\boldsymbol{u}(\theta, t)$ are the total energy, density, and momentum at (θ, t) on the cylinder in Figure 7.4. This cylinder is a collection of points that is a subset of $\partial\Omega_{\text{slip}}$, parameterized by θ.

In the results that follow, we measure error by computing an L1 norm relative to the exact time average pressure $\tilde{p}_{\text{exact}}(\theta)$

$$e = \frac{\int_{-\pi}^{\pi} |\tilde{p}(\theta) - \tilde{p}_{\text{exact}}(\theta)|d\theta}{\int_{-\pi}^{\pi} |\tilde{p}_{\text{exact}}(\theta)|d\theta}. \tag{7.6}$$

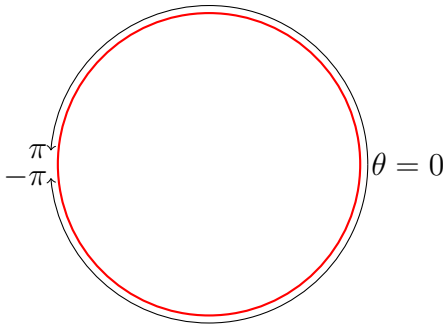


Figure 7.4: When computing solutions for pressure, we reference angles θ defined on the second cylinder (red in Figure 7.1) using the interval $\theta \in [-\pi, \pi]$. These angles are show in in this figure. Restated from Figure 5.4.

Results

We can make some general observations looking at the relative error in the pressure curves in Figure 7.5. First, we can see in Figure 7.5 that each MGRIT+Π case does not converge to the exact pressure curve. In general, there is a 3-5 percent relative error, and the large errors happen near the front of the cylinder (near $\theta = \pm\pi$) where we would expect most of the time dependent flow to have large impact since the flow direction is from the left to the right. Figure 7.5 also shows that the r-type MGRIT hierarchy converges since the error curve eventually flattens, while the i-type tends to oscillate between two states. However, neither i-type state reaches an error smaller than a few percentage points.

Second, we can see that the i-type MGRIT+Π ends up oscillating between two states. These states can be seen in Figure 7.7 and Figure 7.8, on the bottom. So, while the lowest error reached in the i-type MGRIT+Π is lower than the r-type error, we simply do not have any guarantee that on any given cycle the error will be the ideal one. Depending on the cycle we are on, we might get unlucky using the i-type MGRIT coarsening. This type of oscillation could be an indication that the i-type coarsening is bad at smoothing the problem. MG methods typically involve smoothers, and MGRIT is no different. A plausible but untested hypothesis for this behavior is that the i-type is a poor smoother which adds wild oscillations to or MG error. Shock location is shown to cause large differences in i-type corrections in Section 7.1.4.

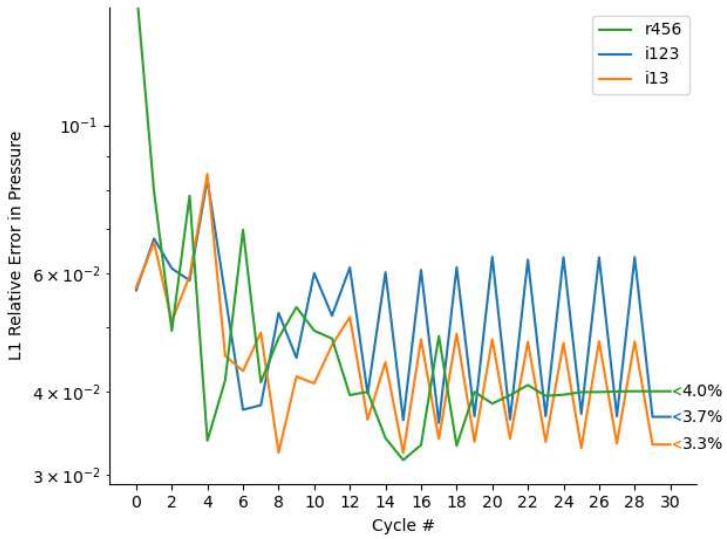


Figure 7.5: We compare the L1 error in pressure for the r456, i13, and i123 cases, see (7.6). We can see that cycling does reduce the error to a few percentage points. However, we note three things from these errors. 1) We can see that none of the hierarchies achieves convergence to the exact solution. Indeed, the relative errors are all a few percentage points. 2) The integrator coarsening methods i13 and i123 oscillate after about 13 cycles. 3) The spatial coarsening method r456 converges to something (no oscillation) after about 20 cycles.

Third, we observe that the r-type MGRIT+Π really does converge to a solution since we notice that the error eventually flattens, see Figure 7.6. Though the error it achieves in the end is higher than the lowest from the i-type, we at least see that we could reasonably expect that after some cycles the r-type will be close to its best state. r-type MGRIT coarsening seems to be a better smoother. This is likely due to the coarser spatial meshes involved. The coarse meshes likely smooth out shocks better than coarser time integrators.

The fourth observation we can make is that each hierarchy reaches its final error at approximately 13 cycles. We ran each simulation for 30 MGRIT cycles, well past the number of cycles which would offer speedup. Convergence in 13 cycles corresponds to an idealized speedup of $\frac{40}{13} \approx 3.07$ over a sequential method on 40 bricks. Recall that this number is an upper bound on the speedup, since there are multiple levels of time integration and inter-brick communications necessary for PinT that reduce that idealized speedup. Actually the MGRIT+Π methods are much slower than the sequential. Section 7.3 presents the actual speedup and observations about the speedup behavior of each MGRIT+Π hierarchy.

Finally, we observe that each MGRIT+Π hierarchy improves the initial time averaged pressure curve. Figure 7.9 shows the initial state and the final state for each choice of MGRIT+Π hierarchy. Each final curve is closer to the exact curve than the initial curve, but none match perfectly.

The big surprise is that the MGRIT+Π methods all have an error associated with them. We would expect, due to the exactness property outlined when discussing Parareal, that we eventually converge to the correct solution. This is not the case in our data. Section 7.1.4 presents some discussion about why inexactness might be the case. The next section computes a time averaged drag, and compares the behavior of drag to that of pressure curves.

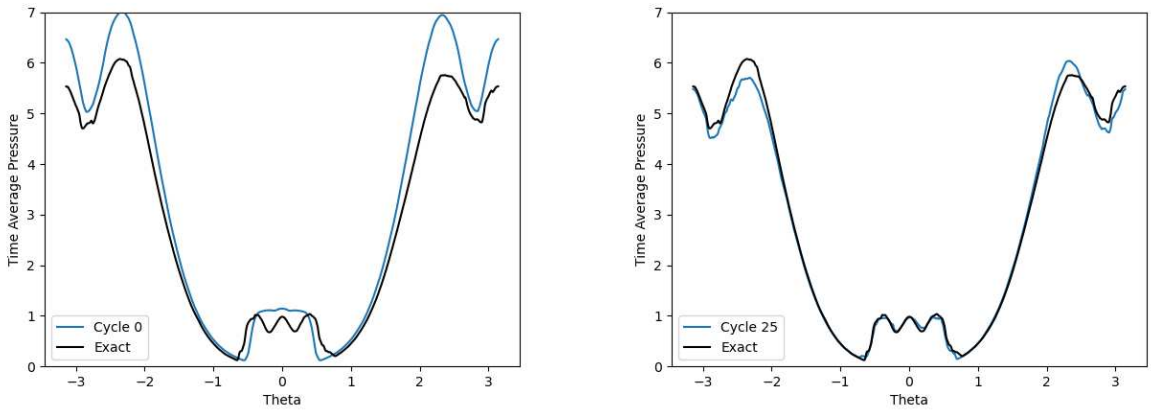


Figure 7.6: This figure showcases the different pressure curves that we get when cycling in MGRIT using spatial hierarchy r456. We see the initial state (left) and final state (right). See how most of the error is due to the leading edge of the cylinder (values near $-\pi$ and π).

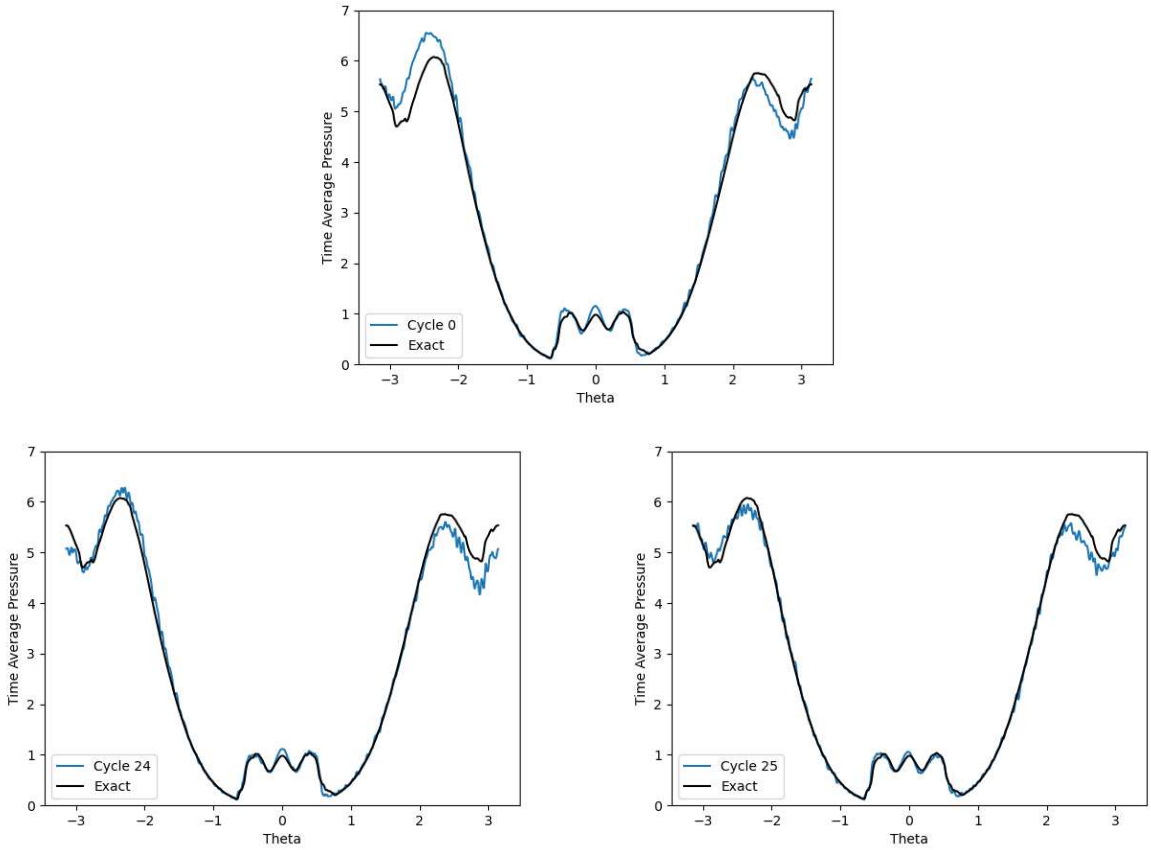


Figure 7.7: When using the scheme i13, We can see that eventually the solution oscillates between one state on even cycles (left bottom), and another on odd cycles (right bottom). Similarly to Figure 7.6 we see that most of the error is due to the leading edge of the cylinder (values near $-\pi$ and π).

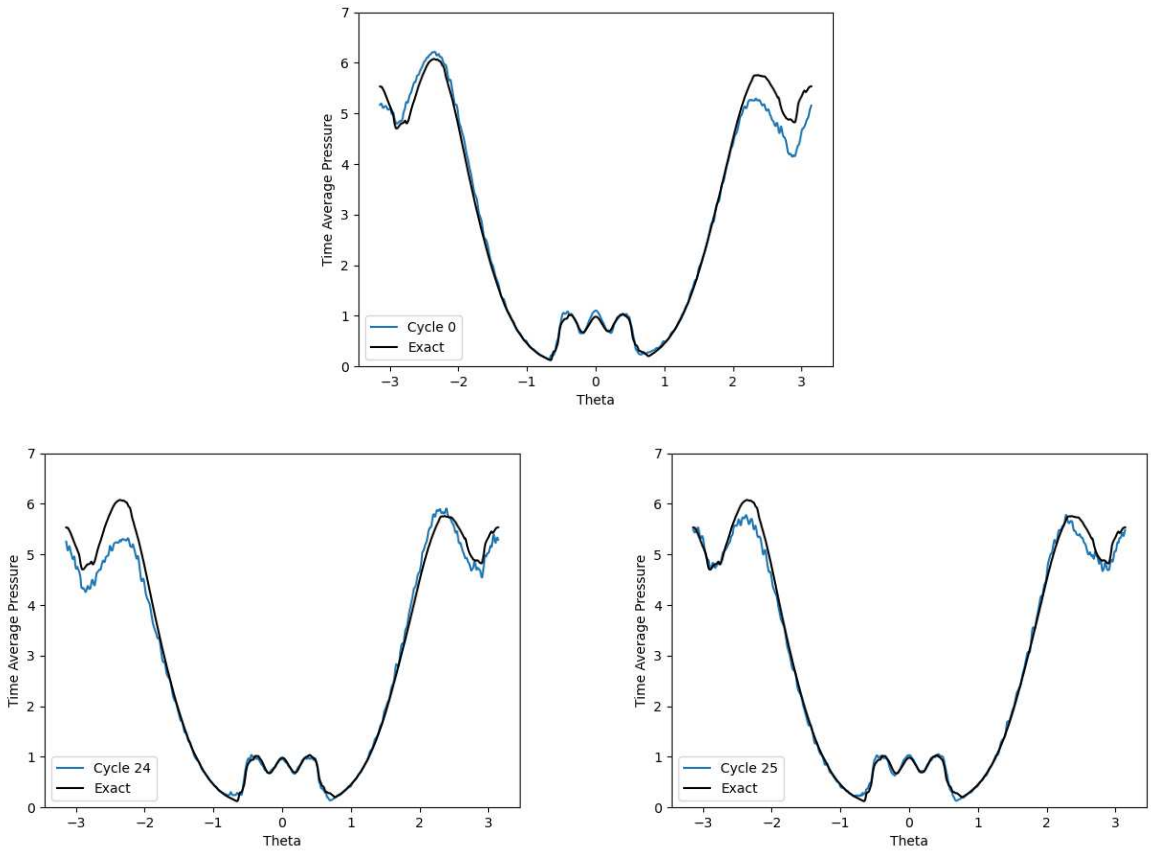


Figure 7.8: When using a three level i-type scheme i123 where we have an intermediate level using RK2 as an integrator, see that eventually the solution oscillates between even and odd cycle states. This behavior is consistent with Figure 7.7, but the results on even or odd cycles are slightly different.

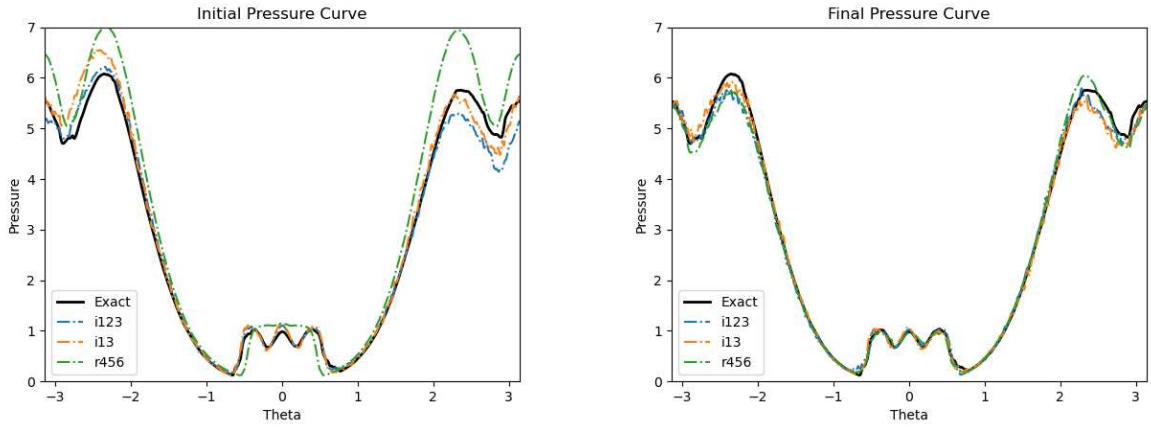


Figure 7.9: Each MGRIT+Π hierarchy improves the time average pressure curves, but none of them reach the exact solution.

7.1.3 Drag on Cylinder

The previous section showed us that MGRIT+Π can predict time averaged pressures on the boundary of objects in fluid flow to within a few percentage points error. This section computes the resulting drag on objects. Time averaged drag was one of our motivating examples to use MGRIT+Π. Since the Euler equations neglect fluid viscosity, the drag on an object only depends on integrating the pressure on the boundary of the object. Therefore it is reasonable to assume that the behavior of the drag should mimic that of the pressure in Section 7.1.2.

For this section, we integrate the pressure along the cylinder's boundary. The time averaged drag, in the direction of the fluid $\langle 1, 0 \rangle$, we will call $\tilde{d}$, and is given by

$$d = \int_{-\pi}^{\pi} \tilde{p}(\theta) \cos(\theta) d\theta. \tag{7.7}$$

The units for drag are the natural units for the Euler equations, meaning we do not specify them. The error in PinT averaged drag is just the difference in the computed drag

$$e = |\tilde{d} - d_{\text{exact}}| \tag{7.8}$$

for a MGRIT+Π method.

Using each of the same MGRIT+Π hierarchies as the pressure section, we observe in Figure 7.10 that the lowest relative error in the drag is on the same order of magnitude that the relative error in pressure reached. This makes sense. We also see that the same convergence behavior we saw for pressure happens for the drag. The i-type MGRIT+Π do not converge, and tend to oscillate between two drag values after about 13 cycles. The r-type MGRIT+Π does converge after about 16 cycles. Again, the drag depends directly on the pressure, so their convergence behavior should be linked. Then that their convergence behavior is similar is not surprising.

However, we note that unlike the pressure curves, only the r-type MGRIT+Π improves the error over the initial guess. Figure 7.12 shows that the i-type hierarchies worsen with cycling, while the r-type hierarchy improves with cycling. The r-type method also has a few lucky cycles where

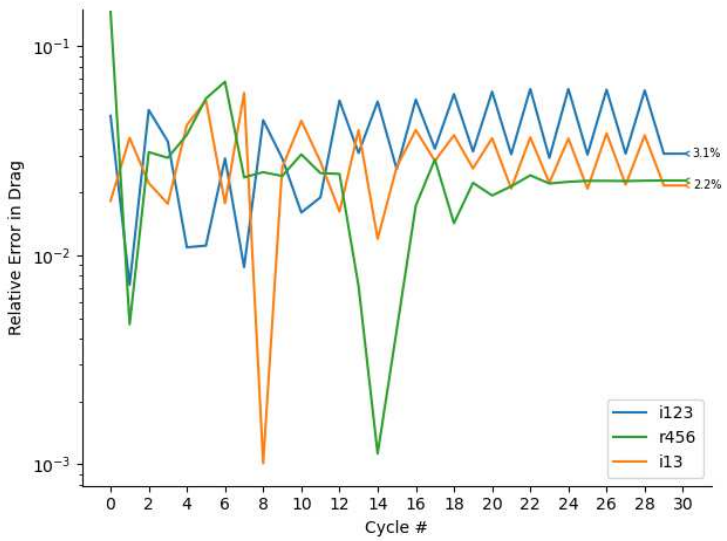


Figure 7.10: The percentage error in drag mirrors the error for pressure, but no method except the r-type MGRIT+Π achieves better error than the initial cycle except for some lucky cycles where the error is small.

the relative error is quite small. Looking at the time-dependent drag curve in Figure 7.12 we see that the i-type drag curves do not match the exact curve very closely after about 2.5 seconds of simulation, while the r-type method matches the exact drag curve for longer, leading to a better relative error. Still, each of the drag curves drift off of the exact drag curve which leads, ultimately, to a few percent of error. Moreover, Figure 7.11 shows that each MGRIT+Π method underestimates the exact time averaged drag. Like the time averaged pressure, we note that after about 13 cycles the error matches the error at cycle 30.

Either way, PinT exactness is broken for MGRIT+Π for both pressure and drag. In the next section, we take a deeper look into why the MGRIT+Π fails to be exact. To do so, we first look at the pressure as it evolves in time using MGRIT+Π to gain insight into why errors persist. Then, we look at the correction terms τ in MGRIT+Π to speculate on the nature of how parts of the correction propagate through space.

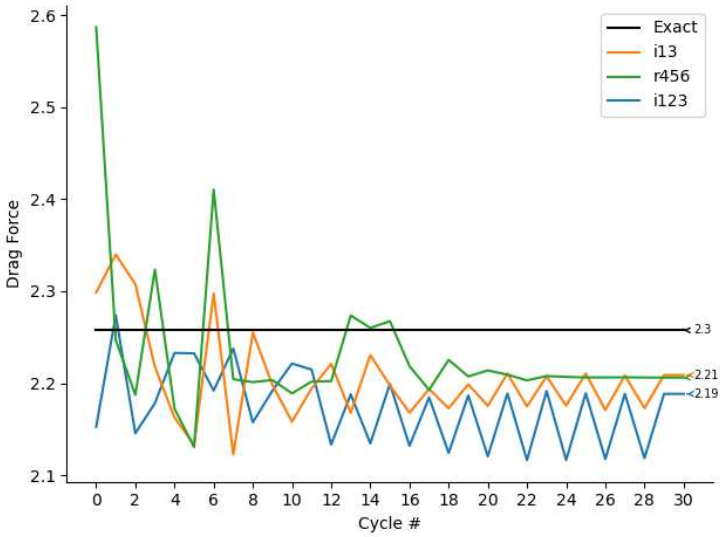


Figure 7.11: The time averaged MGRIT+Π drag force underestimates the exact drag force (in the natural units for the Euler equations.

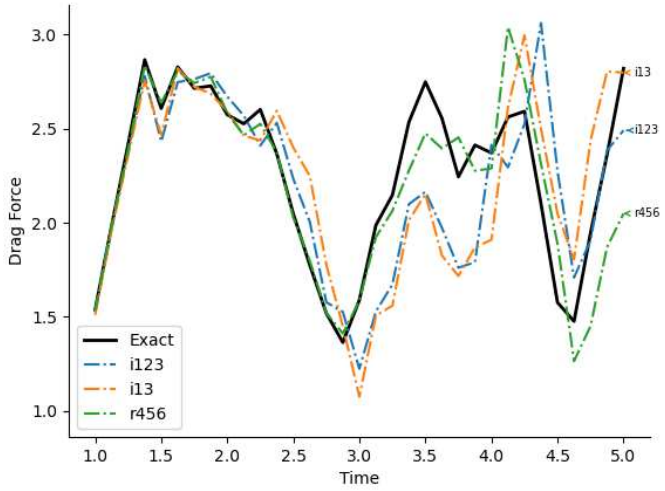


Figure 7.12: When we plot the drag curves in time at the final MG cycles, we see that there is wild variation in the calculated drag near the end of our computed time interval for all the methods. The r456 case is closer on the interval $[3.0, 4.0]$ than the other ones, leading to a more accurate time average.

7.1.4 MGRIT+Π and Exactness

The fact that the exactness property is not satisfied by either the r-type or i-type MGRIT+Π is a surprising result of this chapter. We need to try and understand why exactness fails. In this section, we will first look at surfaces which represent pressure errors along the boundary of the downwind

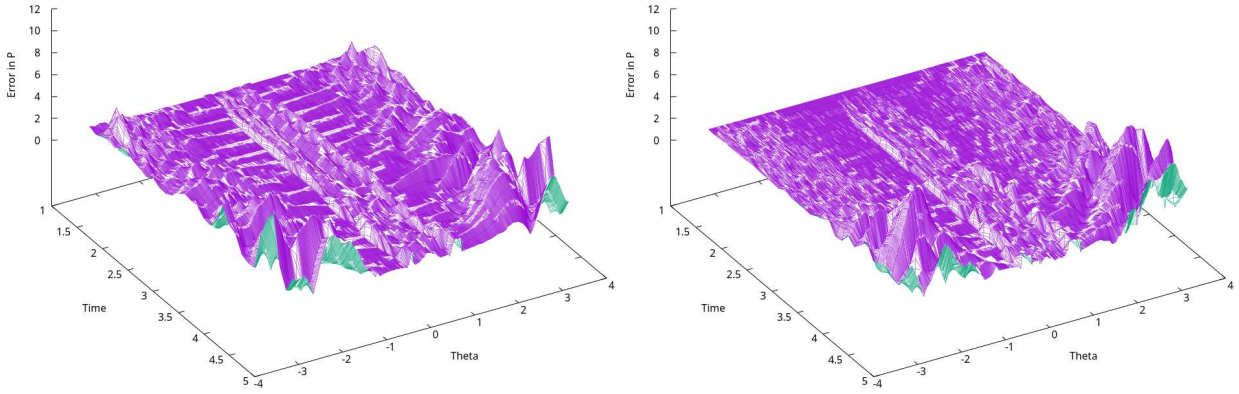
cylinder across the different bricks in our numerical examples for the r-type MGRIT+Π. These surfaces help us to speculate on why exactness is broken. Then, we look at the τ corrections to understand why the i-type MGRIT+Π methods might struggle to converge like the r-type method does.

Why is exactness broken?

We first present an image of pressure on the cylinder in space and time. Figure 7.13 plot the error in pressure at the end of each brick in time along the boundary of the downwind cylinder (Figure 7.4). The resulting plot is a surface that represents the error in time. From the PinT exactness property, we would expect that with continued MGRIT cycling the error on each brick would reduce to zero. As we should expect, the initial guesses have lots of error. By cycle 30, the last MGRIT+Π cycle we used, we expected that the surfaces would be zero on 30 out of 40 of the bricks. If we imagine the surface as a cloth that needs ironing flat, exactness on cycle 30 means that we would see that the surface has been ironed out up to $t = 3.75$ (brick 30 is $t = 3.75$). Figure 7.13 shows that the error in the r-type MGRIT+Π does get smoothed out on cycle 30 compared to cycle 0, but is not zero. Figure 7.13 also shows that while the initial errors for the i-type MGRIT+Π are smaller the smoothing of the error surface is less. In fact, the errors at the last cycle are basically the same as the initial.

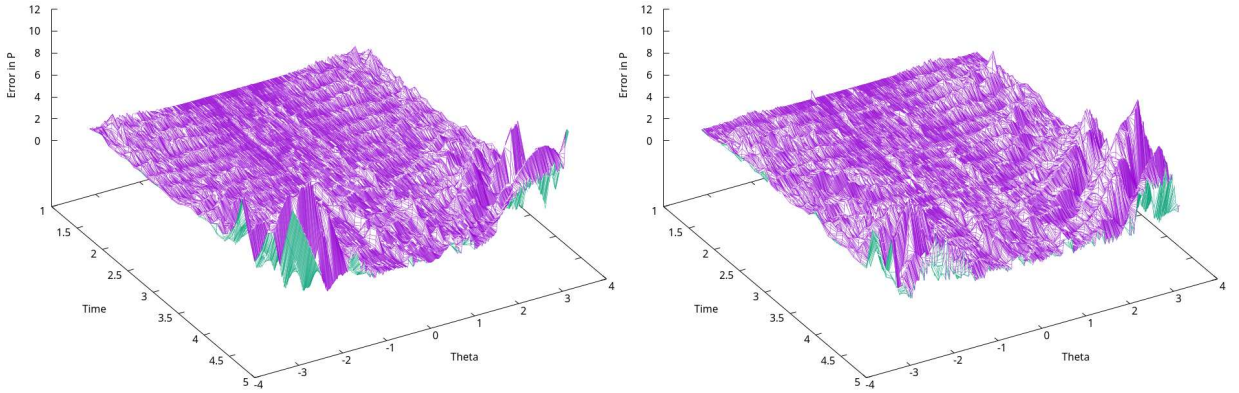
These surfaces show that MGRIT+Π has persistent errors beyond PinT exactness. Since small errors in the past can grow to large errors in the future for hyperbolic systems, Figure 7.13 provides evidence that our projection operation is the ultimate cause of the limiting relative errors noted in Section 7.1.2 and Section 7.1.3. Since MGRIT alone is exact, it must be that adding a projection as we did in Algorithm 7 causes the loss of exactness that we see. In summary, MGRIT+Π loses exactness as a consequence of the projection operator that we had to use in order to remain on our stable manifold $\mathcal{M}_r$.

In Chapter 8 we discuss possible work-arounds for the exactness issue. We note that the MGRIT+Π algorithm applies a projection operation Π all the time, but it is possible that we could toggle Π to turn off if we know that a brick ought to be exact on a given cycle.



(a) r-type Error Cycle 0

(b) r-type Error Cycle 30



(c) i-type Error Cycle 0

(d) i-type Error Cycle 30

Figure 7.13: These surfaces represent the error in pressure versus time and angle around the second cylinder θ for r-type MGRIT+Π hierarchies. At 30 cycles, we would expect three fourths of the bricks to be exact, up to $t = 3.75$. However we note instead that there are small persistent errors for all $t < 3.75$ indicating that MGRIT+Π breaks PinT exactness. We see the surface is not smoothed at all for the i-type surfaces, even on cycle 30. Indeed, the error for the i-type is worse than the r-type.

What do corrections τ look like?

The last part of the analysis is to posit possible reasons why the i-type methods have worse errors than r-type methods. This fact means that i-type methods generally require more projection than r-type methods.

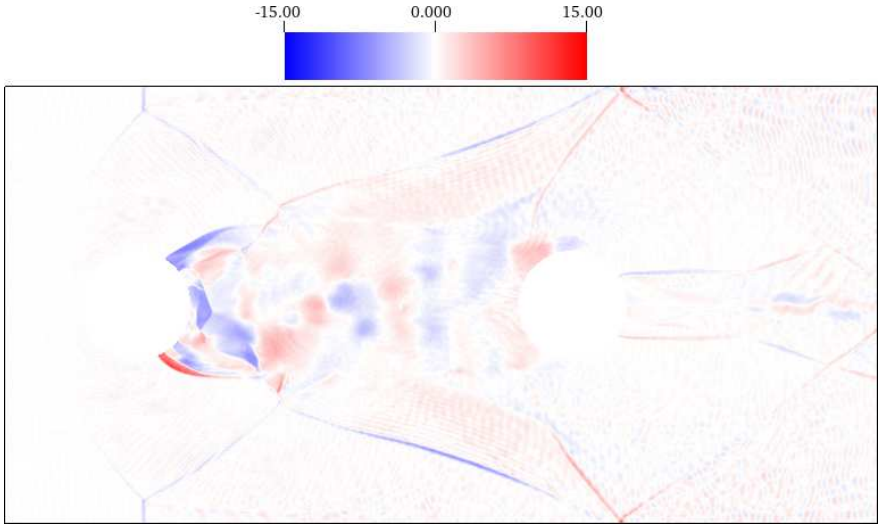
A way to study this is to show some representative images of the corrections τ that both the r- and i-type have. This will show us the fluid features that each type captures well or not well, and which MGRIT+Π coarsening scheme requires more projection use. Figure 7.14 shows the total energy E component of τ for the i-type MGRIT+Π at a fixed point in time. Notice that near

shocks and in down-wind locations (right side of the domain), we see a large range of corrections, roughly $[-15, 15]$. This indicates that the coarser levels in the i-type hierarchy do not accurately predict the locations of the shocks or in the regions down-wind of the cylinders. The low order time integrators have large dispersion errors that cause the transport speed to be off, hence the differences in shock locations. These large negative corrections near shocks can force the internal energy of $U + \tau$ to be negative in places where the total energy is already small. Unfortunately, these tend to be downwind of the cylinders and near the boundary where this dissertation measures time averaged quantities.

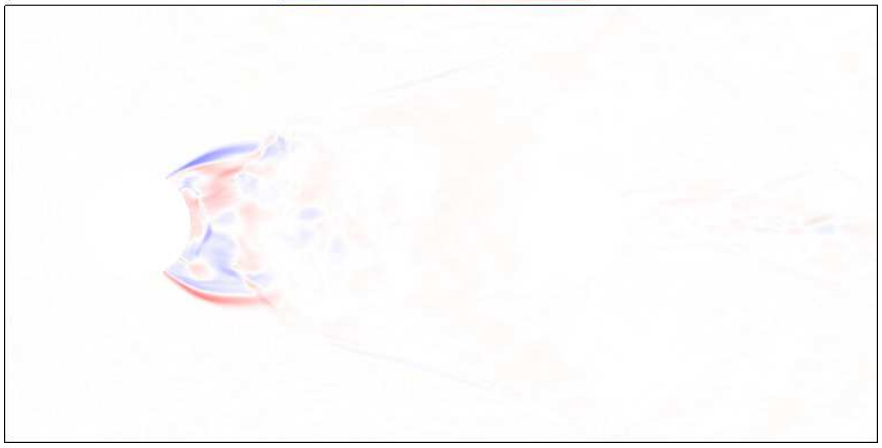
Figure 7.14 illustrates that if we plot the r-type total energy E component of τ, the correction terms are much smaller. For an r-type MGRIT+Π at $t = 2.5$, we see that the region where τ matters is also downwind like the i-type method. However, these corrections see a much smaller range than $[-15, 15]$ like the i-type hierarchy, and do not generally involve shocks as much (meaning that on the middle MGRIT+Π mesh with $r = 5$ the shock locations on the fine mesh $r = 6$ are well resolved, likely smoothed also).

The i-type MGRIT+Π method has large corrections near shocks. On the other hand, the r-type MGRIT+Π methods captures the shocks much better. This shock capturing difference means that the large negative corrections in the i-type MGRIT+Π methods would probably require more projection onto the stable set than those smaller negative corrections in the r-type methods. Since we already noted that projections are the likely culprit for in-exactness of out PinT method, the likely need to use more projections with the i-type method is probably the driving force behind the worse relative error for i-type methods compared to the r-type method.

In the next section, we take the best performing MGRIT+Π hierarchies for both i- and r-type methods, and apply them to a simulation of an airfoil in landing configuration. We want to understand how the MGRIT+Π performs on a sufficiently different and realistic situation where someone might want to understand the time averaged pressure and drag.



(a) i123 τ corrections in total energy E



(b) r456 τ corrections in total energy E

Figure 7.14: To get a sense of what sorts of corrections take place, we plot what the correction τ is for an i-type MGRIT+Π and an r-type MGRIT+Π at $t = 2.5$. For r-type MGRIT+Π τ corrections, the size of the correction is smaller than that if the i-type MGRIT+Π at $t = 2.5$.

7.2 Three Part Airfoil in Landing Configuration

In this section, we select an r-type MGRIT hierarchy and the best performing i-type MGRIT hierarchy from above to run a three part airfoil in Euler equations at approximately landing speed and angle of attack for a landing plane with mean aerodynamic chord length of about 3.8 meters. This chord length is representative of mid-sized aircraft. For example, a Boeing 747 has mean chord on the order of 9.8 meters [75, Appendices, Data A, Table 3].

The computational mesh comes from a 2D tetrahedral mesh in Galbraith et al.'s paper [76], with the exception that we split Galbraith's triangular mesh into a quadrilateral mesh for compatibility with *ryujin*, which only works on quadrilateral meshes at the time of generating results (fall 2025), but now would work with the original tetrahedral mesh (spring 2026). We also scale the mesh to have the correct chord length by scaling it up by 3.8 (the original mesh is non-dimensional in that the chord is 1). The whole mesh, seen in Figure 7.15 is 1000 chord lengths wide to avoid boundary conditions except at the airfoil and is specially designed for only a 16° angle of attack.

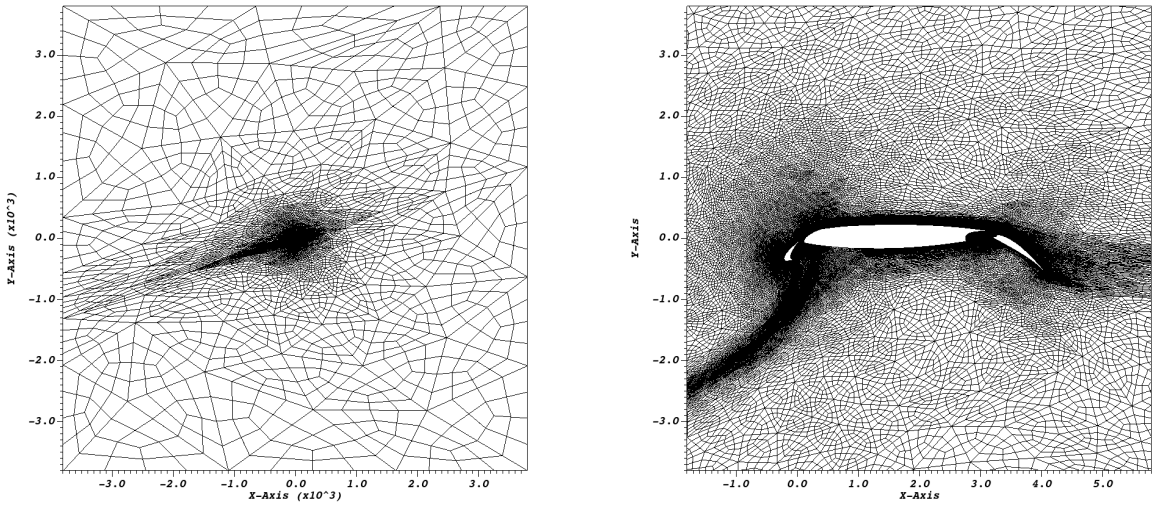


Figure 7.15: This figure shows the mesh used to generate results for an r-type MGRIT+Π for an airfoil in landing configuration and conditions. The length of the airfoil is 3.8 meters to match a mid-sized jet aircraft. This mesh is only suitable for an angle of attack of 16° [76]. The whole mesh is on the left, while a zoomed-in view is on the right. Notice the mesh refinement in the direction of 16 degrees.

The initial condition is similar to Section 7.1, where here the state is set to a constant initial state at all DOFs

$$U(t=0) = \begin{bmatrix} \rho_0 \\ \rho_0 \boldsymbol{v}_0 \\ E_0 \end{bmatrix} = \begin{bmatrix} 1.4 \\ 1.4 \cdot 0.23 \cdot [\cos(16^\circ),\, \sin(16^\circ)]^{\mathrm{T}} \\ 2.573 \end{bmatrix}, \tag{7.9}$$

consistent with Mach 0.23—about 80m/s, a typical landing speed—Euler flow at 16 degrees angle of attack.

For the MGRIT+Π hierarcy, we use both the r-type r01 and i-type i13. We chose to use a mesh that has been refined only once due to computational constraints. We found that we could simulate the mesh in Figure 7.15 with reasonable simulation times with one global mesh refinement, but not more. Therefore we could do a two level MGRIT+Π r-type hierarchy where the coarse level integrates the unrefined mesh. We chose the i13 method because between the two tested in Section 7.1 (i13, i123) it had the lowest error in pressure at 13 cycles.

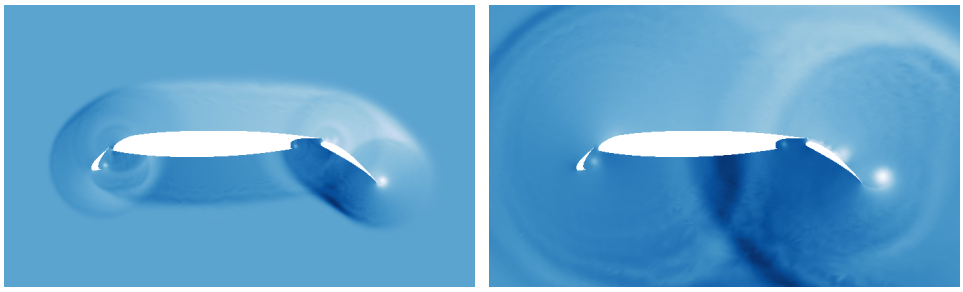
We generated the exact data by solving on a spatial mesh with one global refinement and using a third order Runge-Kutta for time integration on the interval time $[0, 6]$. Figure 7.16 illustrates what the exact solution looks like at different points in time.

The PDE we chose to solve for this numerical example is the Euler equations. The reason we chose this was computational feasibility to generate results in time for the deadlines associated with writing a dissertation. We recognize that for a realistic simulation, we would have to solve the full Navier Stokes equations. This is left to future directions for research in Chapter 8. (It is possible that the physical dissipation in the Navier Stokes equations could help convergence properties of our PinT method [52]).

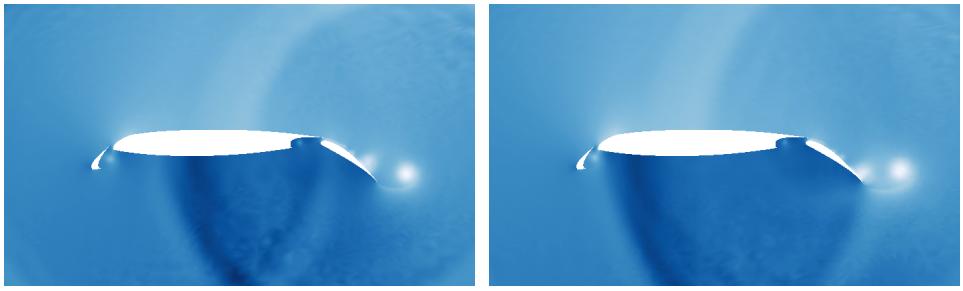
Since we are initializing with a uniform initial condition we will need to follow the discussion of Section 7.1 and define a lower bound for our time integrals. Therefore we will average over some interval $[t_{\text{start}}, 6]$. To define t_{start} Figure 7.17 presents the calculated drag from the airfoil with respect to time. Like in the previous section, we look at this curve for a visual indication that the effects of uniform startup are gone. We see in the figure that after about one second of simulation time the drag seems to settle into a less drastic looking curve. Therefore, we define $t_{\text{start}} = 1$.

In this section we measure the time averaged pressure along the airfoil $\partial\Omega_{\text{airfoil}}$ (where a slip condition is applied) as well as the time averaged drag on the airfoil. The average pressure is given by

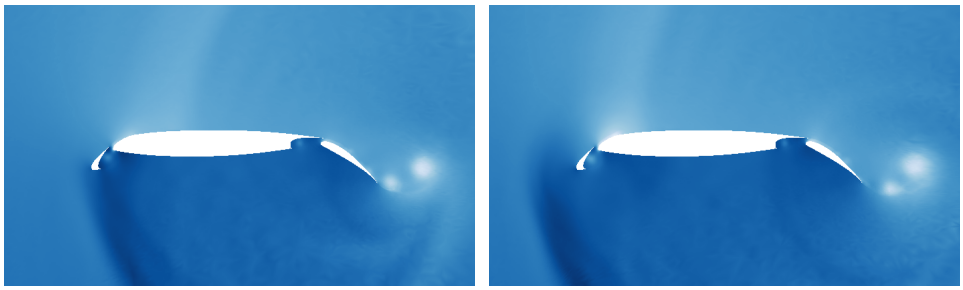
$$\tilde{p}(x) = \frac{\int_1^6 p(x,t)dt}{5} \tag{7.10}$$



(a) Pressure at $t = 0.75$ **(b)** Pressure at $t = 2.25$



(c) Pressure at $t = 3.0$ **(d)** Pressure at $t = 3.75$



(e) Pressure at $t = 4.5$ **(f)** Pressure at $t = 5.25$

Figure 7.16: The exact time evolution of pressure in Mach 0.23 flow Euler equations around an airfoil in landing configuration. Regions of darker color have higher pressure.

for $x \in \partial\Omega_{\text{airfoil}}$, ie. for all spatial locations along the airfoil. The time average drag force in the direction of the airflow $\langle \cos(16^\circ), \sin(16^\circ) \rangle$, is given by a surface integral of the pressure along $\partial\Omega_{\text{airfoil}}$ which can be computed as

$$\tilde{d} = \int_{\Omega_a} \langle \tilde{p}(x) \cos(16^\circ), \tilde{p}(x) \sin(16^\circ) \rangle \cdot \boldsymbol{n} dS. \tag{7.11}$$

Here, as in Section 7.1 the units of the drag force are the natural units in the Euler equations (4.5), and we don't bother with the units too much here. In the resulting analysis, we measure

convergence using the L1 relative norm of our PinT averages when compared to the exact time sequential solution, like in the previous section with the two cylinders.

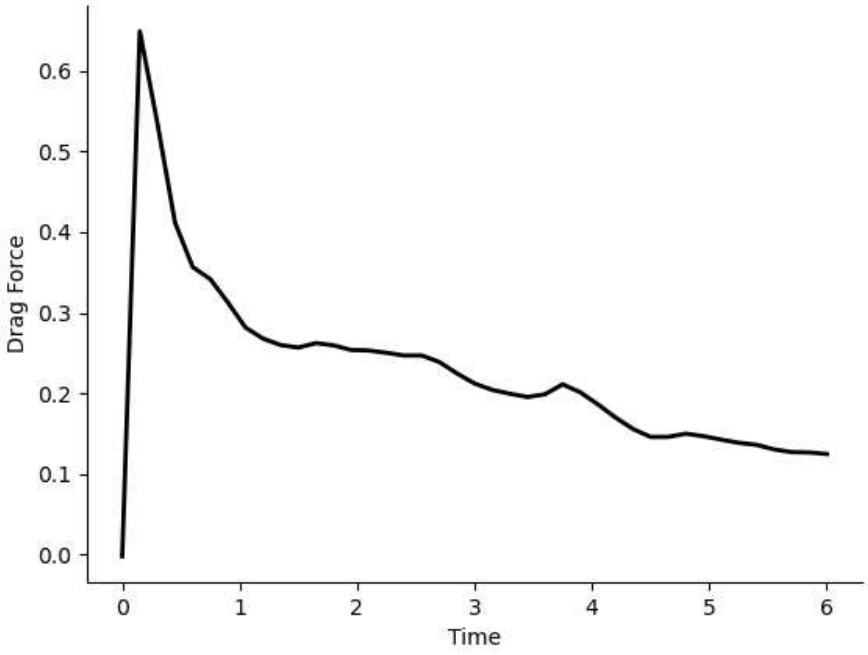


Figure 7.17: Looking at the exact drag curve, we see that there is a startup time of about one second here before the drag starts to behave in a more regular way. Compare this similarity to Figure 7.3 where we had a one second startup time for the two cylinder case.

7.2.1 Pressure on Airfoil

In this section we plot the pressure curves that we get using MGRIT+Π compared to the exact average pressures for the three part airfoil geometry. Figures 7.18 and 7.19 compare the average pressure curves over the top and bottom parts of the airfoil for an r-type MGRIT+Π and i-type MGRIT+Π respectively. On each figure, we plot measured time average pressure curves for the initial guess cycle 0 (blue), the final MG cycle 13 (red), and the exact time average pressure (black). On the top of each figure we plot the pressure above the airfoil, in the middle we plot the airfoil's shape and on the bottom we plot the pressure on the bottom of the airfoil.

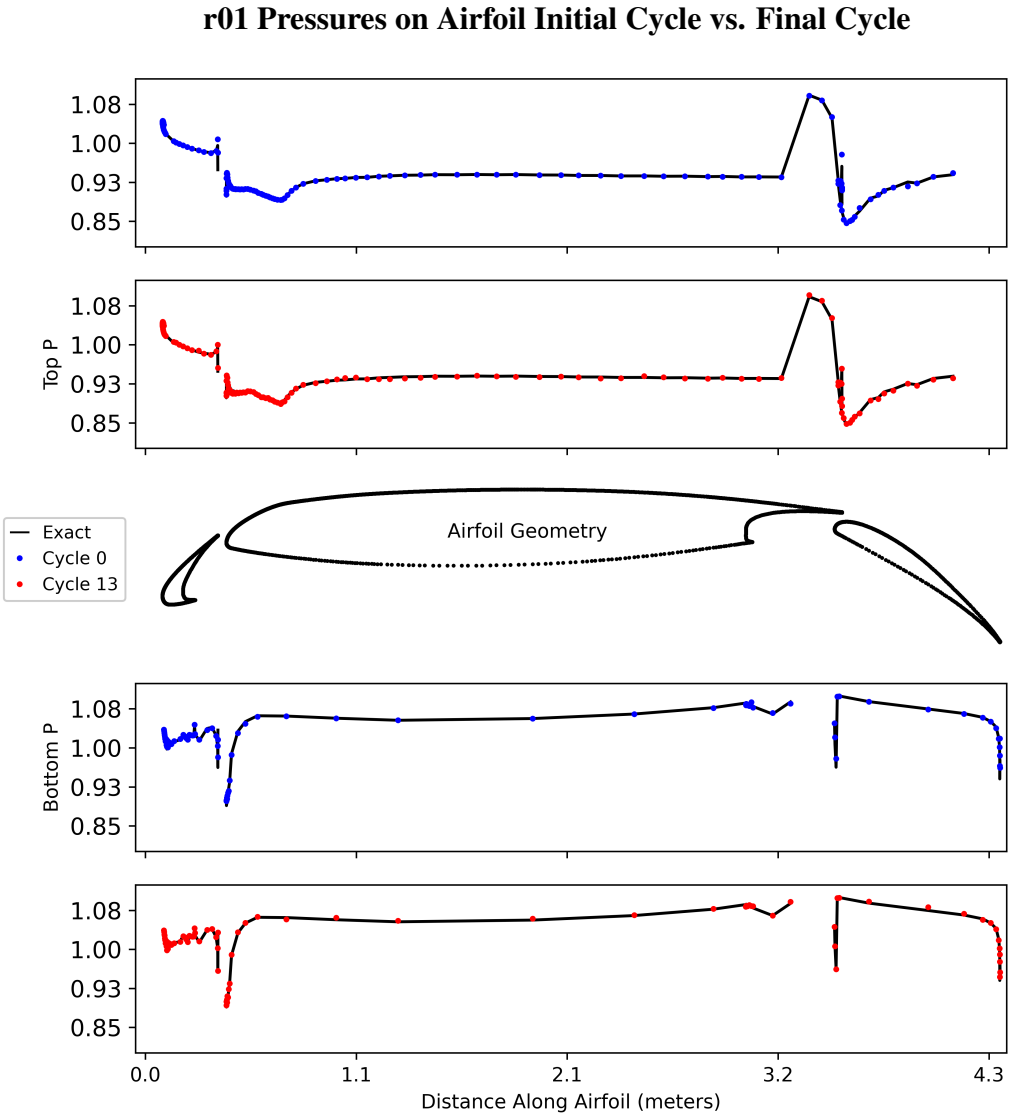


Figure 7.18: This figure presents a view of the average pressure curve above and below the airfoil using an r01 hierarchy. We compare the exact in black to the initial (in blue) and to the final (in red) at a subset of all the points. The pressures do not change much from the initial to the final, this is supported by Figure 7.20

Looking at the comparisons in Figure 7.18 we see that, for the r-type MGRIT+Π, for much of the airfoil, the average pressure is constant as a function of position along the airfoil. The initial average state (in blue), which is given by the coarse problem, captures this behavior quite well. However, in regions where the pressure is not constant with respect to position, the difference is larger. For example, looking at the third part near $x = 4$, we can see that the exact pressure (black line) has some fluctuations on top and bottom that are not captured by the coarse information. In

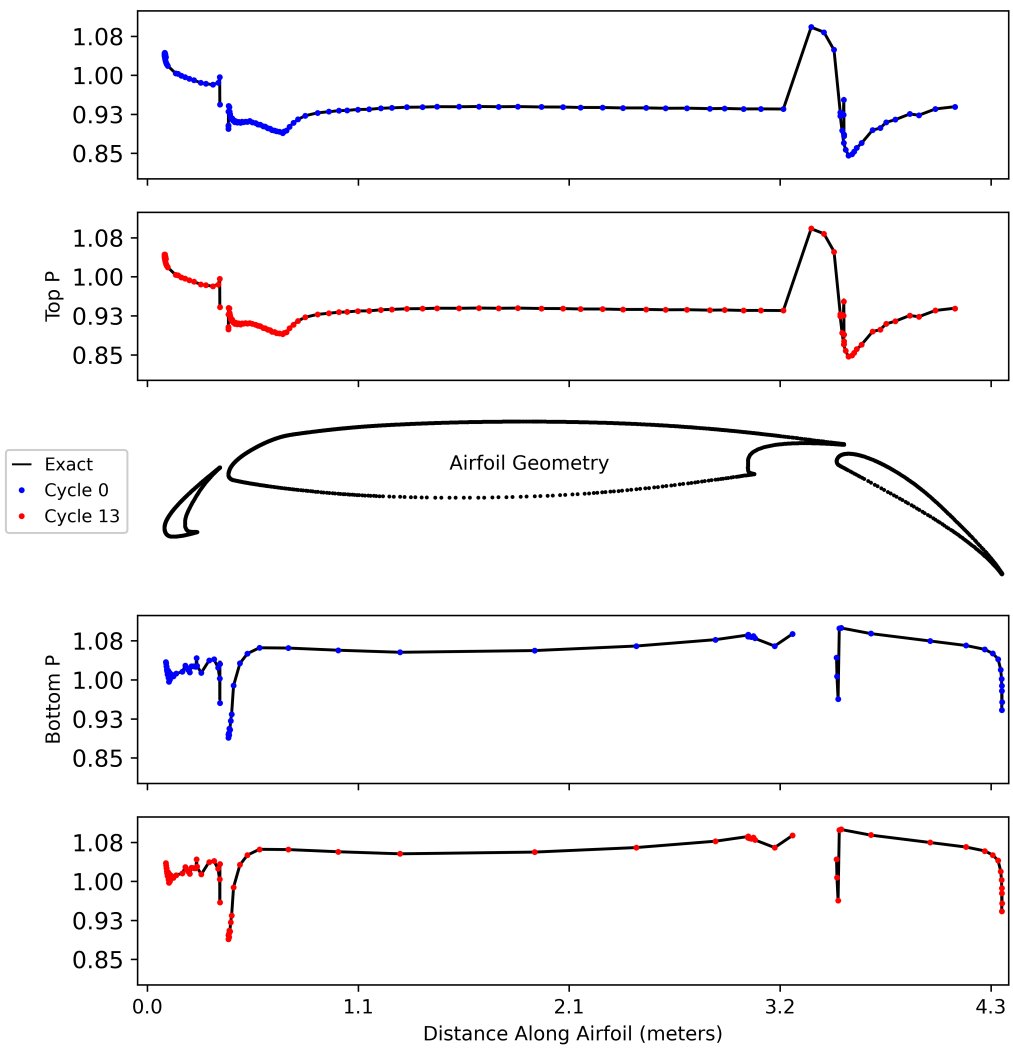


Figure 7.19: This figure presents a view of the average pressure curve above and below the airfoil using an i13 hierarchy. We compare the exact (in black) to the initial (in blue) and to the final (in red) at a subset of all the points. The middle plots shows the airfoil geometry. We can see that basically no change occurs from first to last cycle.

this area near $x = 4$, the final average state (in red) captures the fluctuations better, but introduces incorrect fluctuations on the more constant parts of the airfoil. For example look at the red plot on the bottom near $x = 1$ where we see that the pressure wiggles beyond what the exact pressure should be.

Figure 7.19 shows that for the i-type MGRIT+Π, the initial and final curves more or less match the exact time averaged pressure. This either means that MGRIT+Π does not help in this situation,

or that the initial guess is sufficiently good that PinT would not improve on it. Figure 7.20 shows us that the latter is more likely. The relative error of the time averaged pressure on cycle 0 is already quite small. PinT cycling does not improve the relative error. This is true for both the r-type and i-type coarsening methods. So, in the case that the initial guess is sufficiently good, it seems that MGRIT+Π does not drive the time average away from the initial guess. What Figure 7.20 does show us is the interpolation error that we have to pay for the r-type MGRIT+Π referenced in Section 3.2.2. Both types of MGRIT coarsening have flat error curves. The only difference between the i-type error and the r-type error is the magnitude of their error. The driving factor for the r-type being higher is that this type of coarsening involves an interpolation of the residual from the coarser spatial mesh to the finer spatial mesh for these two level methods.

Overall, the final pressure curve is more or less the same as the initial for both the r-type and i-type MGRIT+Π, see Figure 7.20. Next, we see if the calculated drag we get matches the exact time average. It is tough to say that these relative error are truly any better than the initial. For example, Figure 7.21 shows the same flat error vs. cycle curves for computed drag. The next section looks at the drag computed by the pressure and comments on what the drag computations tell us about MGRIT+Π.

7.2.2 Drag on Airfoil

Like in Section 7.1 we want to see if the drag value from the time averaged pressure is computed correctly. Figure 7.21 shows that the drag error is essentially flat for both r- and i-type which matches the behavior of the time averaged pressure. Again, we can see that the initial guesses for both the r-type and i-type MGRIT+Π are already pretty good (Figure 7.22 and Figure 7.23). However, we note that the r-type error actually seems to increase. On closer inspection of the time dependent drag for in Figure 7.23, we see that there is some instability driving the time dependent drag curve away from the initial guess. We do not know exactly what drives this instability here, but we suspect that the poor aspect ratios in the mesh near the boundary of the airfoil cause the interpolation error incurred while doing spatial coarsening to dominate. It could also be that during

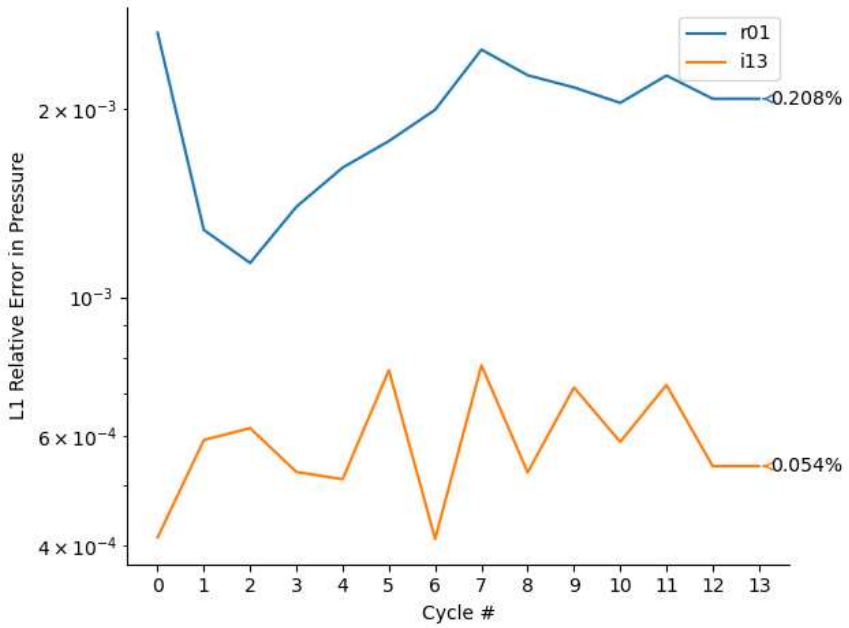


Figure 7.20: Here we see that the pressure on our final cycle is better by a slim margin than the initial for the r01 MGRIT+Π setup. For the i13 setup, we see a similar trend. However, this does not get affected by the interpolation error, so is many orders of magnitude smaller.

the computation of the drag, the long edges of some mesh elements could cause the surface integral to have poor error characteristics.

In summary, we can say that MGRIT+Π most often does not improve on an already good initial guess, and that if we are not careful in understanding interpolation on the mesh involved we can actually drive the time averaged quantities away from the exact state. As the final part of our numerical examples, we will present the wall clock time for each of our numerical examples and assess the possibility for speedup using PinT to compute our time averaged quantities of interest.

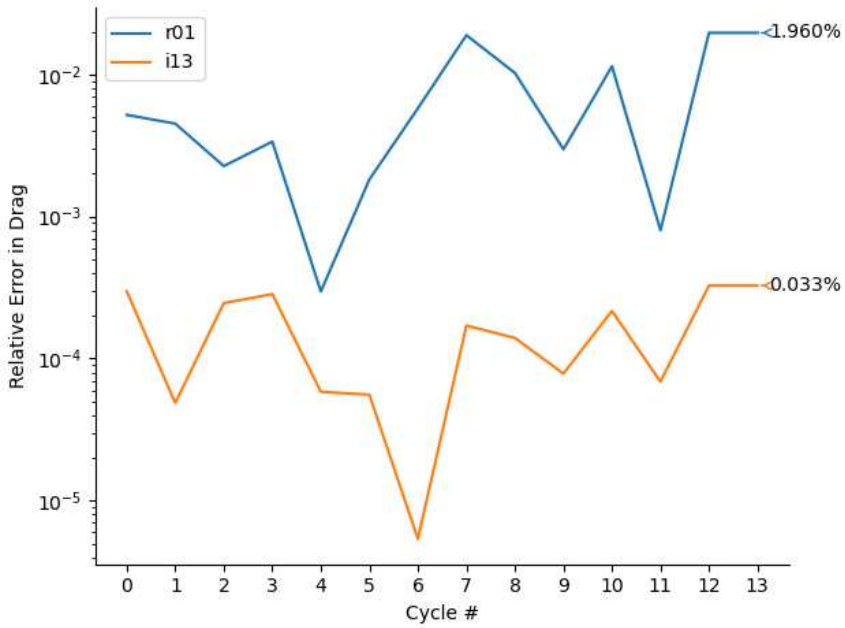


Figure 7.21: Here we see that the drag force erro on our final cycle is essentially flat for the r01 MGRIT+Π setup. For the i-type we see a similar behavior, but without the effects of interpolation.

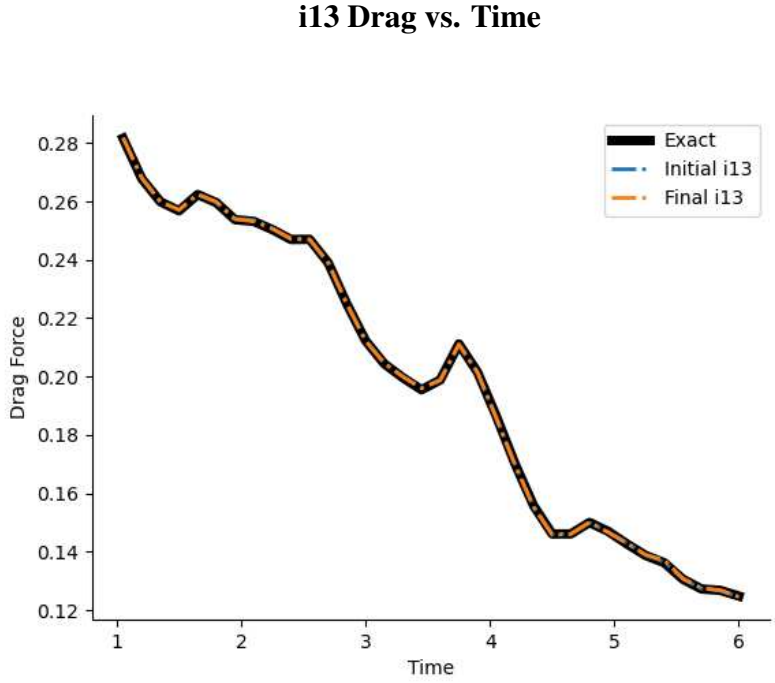


Figure 7.22: In this figure we see that the drag vs. time curve does not change much with MGRIT cycling for the i13 MGRIT+Π setup.

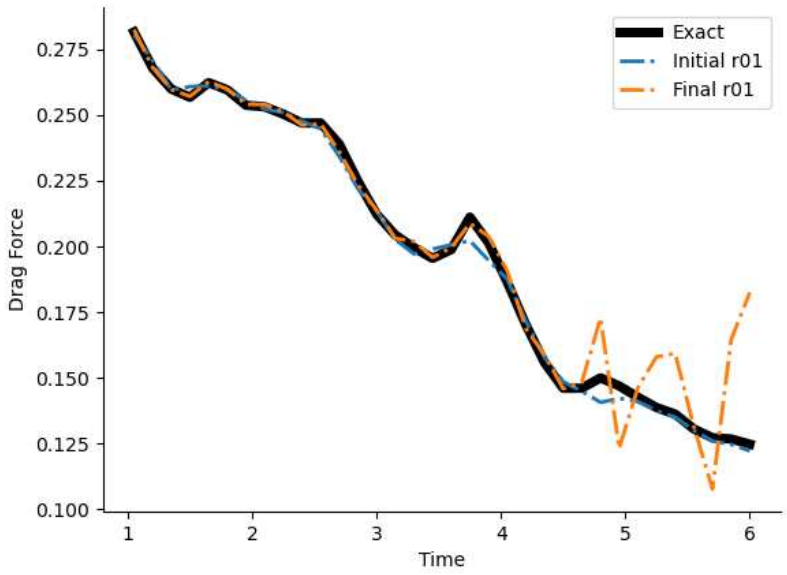


Figure 7.23: In this figure we see that the drag vs. time curve is worse after MGRIT cycling for the r01 MGRIT+Π setup. We hypothesize that the interpolation costs to take corrections from the coarse spatial mesh to the fine spatial mesh drive this growing instability.

7.3 Wall-Clock Timings

In this section we time MGRIT+Π by comparing the wall-clock time for the sequential solution with the overall wall-clock time for MGRIT+Π for each of the numerical results presented previously. For our exact results, we used one rank to solve the problem. Then, for the MGRIT+Π results, we used one rank per time brick. So, for our results with 40 bricks we used 40 ranks. This is a fair way to compare the sequential and PinT methods since it most closely matches the way PinT gets around sequential "saturation", see Section 2.1. We pretend that the sequential time integration saturates at one rank, and extend with more. The time integrator code **ryujin** also supports multi-threading, so we used 4 threads per rank.

Overall, we note that the MGRIT+Π simulations presented herein took significantly longer than the sequential solution did, see Table 7.1. This is not surprising since the communication cost is likely quite high for the MGRIT+Π, where bricks need to share the entire spatial mesh meaning that we have to communicate the state at every DOF in space, unlike spatial DD where

the information that needs to be communicated comes from a 'thinner' region of space which touches a fraction of the total spatial DOFs. Furthermore, we only use 40 time bricks, so based on the discussions in Section 3.1 we would need convergence in many fewer than 40 cycles. Table 7.1 presents the wall-clock time to perform 13 cycles.

Table 7.1 illustrates the timing from 13 cycles since this is when the minimal error is reached for the Two Cylinder case. If we had a well defined convergence criterion this is when we'd stop the MGRIT+Π. Across the board, the MGRIT+Π algorithm, converging in 13 iterations, is a few times slower than the sequential solution.

The table does tell us that generally the spatial coarsening is faster than the integrator coarsening. We can see that the runtime for the r-type MGRIT+Π are lower than the i-type MGRIT+Π simulations. Given that each type reaches approximately the same relative error, this means that r-type coarsening is a more efficient coarsening scheme.

Method	**Two-Cylinder**	**Airfoil**
Sequential RK3	223s	228,944s
r456	29,426s	
r01		494,580s
i13	37,395s	1,386,056s
i123	86,142s	

Table 7.1: Wall-time in seconds for numerical results. For all the numerical results presented herein, we used a third order Runge Kutta (RK3) as the fine integrator F giving the exact data. For the two cylinder results, we used a fine mesh that was 6 times refined from a reference mesh. For the airfoil results, we used a mesh that was globally refined once from a reference mesh. The table above presents wall-time for each numerical example compared to sequential time integration on the finest level. The row r456 uses a three level spatial coarsening MGRIT+Π with the coarsest mesh having 4 global refinements from a reference mesh, the middle having 5, and the finest 6. Likewise the row r01 is a two level method with the coarsest level being the coarse, reference mesh. The row marked i123 is a three level integrator coarsening MGRIT+Π with the spatial mesh being the same as the fine mesh on each level, but using a first order Runge Kutta integration on the coarse level, a second order on the middle, and a third order on the finest. Likewise, i13 is a two level method with first and third order integrators.

Table 7.1 also illustrates that MGRIT+Π is still far from demonstrating a faster wall-time compared to the sequential method. In Chapter 8, we discuss that we should perform a strong scaling study where we evaluate the performance of MGRIT+Π as we add more processors and

bricks to a fixed size problem. It is possible that the timings in Table 7.1, since they use a fixed number of 40 processors, could decrease as we utilize more more bricks and therefore smaller bricks with quicker time to solution.

7.4 Summary

In this chapter, we applied MGRIT+Π to two numerical examples. We answered our questions posed in Section 5.2.1. First, we see that MGRIT+Π can be used to calculate time averaged quantities of interest, but because of the added projection operations, we see a persistent error that breaks exactness. Second, we answered that MGRIT+Π seems to converge to a final state when using spatial (r-type) coarsening, while it fluctuates between states when using integrator (i-type) coarsening. This leads us to believe that the spatial coarsening is a better coarsening method. This is further supported by the observation in Table 7.1 which shows that r-type coarsening produces faster overall run times compared to i-type methods. However, all the MGRIT+Π hierarchies we tested were much slower than the sequential method. Finally, we note that convergence happens at 13 MGRIT cycles.

Chapter 8

Conclusions

This chapter provides an overview of the results and open questions that remain. First, Section 8.1 summarizes the novel contributions made herein. Then, Section 8.2 provides an analysis of using PinT in realistic applications. Finally, Section 8.3 discusses the future research directions and remaining open questions.

8.1 Novel Contributions

In our view, PinT has not found a home yet. This dissertation helps to expand PinT's usefulness beyond where people have used PinT so far. With the results in this dissertation, we have expanded the horizon of PinT by assessing how well it can be applied to a useful goal: the computation of time averaged quantities. Measuring these quantities helps to fill a major gap in the existing PinT literature by using PinT to compute useful quantities from PinT methods in hyperbolic PDEs. In particular, we made novel contributions by doing the following:

1. We analyzed how well PinT does when coupled with state of the art hyperbolic solvers in **computing time averaged quantities**, including a comparison between "spatial coarsening" and "integrator coarsening" to define coarse levels,

2. We developed a **MGRIT+Π** algorithm which enables the use of PinT methods to integrate hyperbolic PDEs;

3. We created **Stable Sets** to help us stabilize MGRIT+Π;

4. We wrote software that interested parties could use to assess how well MGRIT+Π might work in their own situations [77] and as a jumping off for future research in PinT.

In the next section, we summarize our findings from the numerical examples and give an assessment of MGRIT+Π's prospects.

8.2 Summary of Results and Efficacy

We began our study by asking 1) if PinT can be used to compute time averages accurately and 2) if there is any pattern to how many MG cycles we needed to compute these. To do so, we had to develop a stabilized PinT method, which we called MGRIT+Π. Then, we applied this new method to two different numerical examples.

For our first numerical example with two cylinders in Mach 3 airflow in the Euler equations, Section 7.1 showed that when MGRIT+Pi does converge, it converges to the wrong thing, for example see Figure 7.9 where the average pressure curves do not match the exact curves at the final MGRIT cycle. This implies that MGRIT+Π is not an exact PinT method. However, the resulting curves look pretty close. We saw that the limiting percentage of error was approximately 3-5% off of the exact time averaged pressure along one of the cylinders. We attributed this loss of exactness to the addition of a projection operator. The time averaged drag curves also had a similar magnitude of relative error, which is not surprising since drag and pressure are correlated. Finally, we noted that the final relative error happens at 13 MG cycles. Overall, this numerical example shows that MGRIT+Π can improve initial errors in pressure and drag.

In the second numerical example, we took two two-level MGRIT+Π hierarchies and used them to predict the time average pressure and drag on an airfoil in landing configuration. This served as a more 'realistic' example to test our method. For this method, we saw that the MGRIT+Π method seems to not help us in a consistent way. We saw that the drag error was reduced by a small amount for an i-type MGRIT+Π. Still, the initial errors from the coarse solve are small, so at least for this mesh and this set of initial condition, MGRIT+Π is probably not worth using. Further, we saw evidence that the r-type interpolation costs could drive instabilities (reference Figure 7.23).

Comparing coarsening schemes across both examples, we saw that r-type coarsening seems to be more robust than the i-type coarsening. This means that future PinT methods for solving hyperbolic PDEs should start with using spatial coarsening in their methodology. Table 7.1 shows us that r-type coarsening is generally faster than i-type coarsening. Coupled with its higher level of robustness, r-type coarsening is an appealing coarsening option.

However, Table 7.1 also shows that none of the MGRIT+Π hierarchies we tried are fast enough to beat sequential integration. Though we showed that PinT time averaging produces quantities that converge, it is unlikely that these methods will produce solutions faster than sequential time integration, at least on problems like those we tested.

8.3 Future Directions and Open Questions

This section outlines the avenues for extending the research results in this dissertation. Overall, the results in this dissertation illustrate the possibility that time averaged quantities are reasonably computed with PinT methods. However, we did notice that these averages computed from MGRIT+Π have some limiting errors. Since the MGRIT+Π algorithm we used here is very general, we have a sign that there is fruitful research to be done extending the results contained herein. The main barrier to extending the results for time averaged PinT will be the limiting error seen herein.

One of the main open questions has to do with the loss of PinT exactness. In the hyperbolic PinT literature, there is obscure mention of how in certain hyperbolic settings PinT might not be exact. For example, the authors of [78] state: "There is one additional curiosity in Fig. 18, the exactness property is not perfectly satisfied." They go on to offer that they believe that "This is thought to be a result of a small difference in floating point values exhibiting chaotic behavior. It may also be a result of some unknown and undesired interaction between levels in the three-level case." Nobody, to our knowledge, has studied this phenomena. Our results seem to follow this inexactness, but more work needs to be done discerning the origin of such phenomena. The future directions posed below would be good places to start in understanding where, exactly, does exactness go?

We highlight four overarching areas for future research to help answer this question. First, we discuss future directions in the study of PinT with projection applied to linear hyperbolic problems. Of particular interest is designing a better projection operator. Second, we highlight the parameters of MGRIT that could be optimized. One example would be the choice of relaxation scheme. Third,

we propose algorithmic directions like mixed spatial and integrator coarsening or a convergence criterion. Fourth and finally, we note the need for extensions into other dimensions and PDE systems.

8.3.1 Designing a Better Projection Operator

The first major issue that we noted with MGRIT+Π in Chapter 7 was that we had broken PinT exactness. We supposed that this was due to the projection operator. We have a few areas that one could study to understand why projection breaks exactness.

Study MGRIT+Π with a Linear PDE

We noted in Section 7.1.4 that the corrections τ for our two cylinder example had large negative values near shocks. Large negative correction terms tend to cause more need for projections–which we believe ultimately cause the limiting error.

These problematic shocks in the Euler equations arise from nonlinearities in the equations. Since our research questions try to understand how time averaged quantities behave with hyperbolic PinT methods, one should study whether it is the hyperbolic nature or the nonlinearities which drive the large negative parts of τ. To do so, it would be advantageous to study MGRIT+Π in the context of linear PDEs, as this isolates the hyperbolic behavior. One could study MGRIT+Π in the context of a scalar transport equation

$$\frac{\partial u}{\partial t} + c\frac{\partial u}{\partial x} = 0. \tag{8.1}$$

Doing this would provide a clearer understanding of the interaction between the corrections $u + \tau$ and their projected state $\Pi(u + \tau)$.

In hindsight, this would have been a better starting point for this dissertation since the projection operation breaks exactness which is hard to understand in the context on nonlinear PDEs.

Tune Parameters in Algorithm 5

The specific projection operation we used was quite unsophisticated. First of all, we used a reference density ρ_{ref} which we picked arbitrarily, as well as a scale factor $\alpha = \frac{1}{100}$ which we picked arbitrarily. The α parameter in particular could be optimized, since it effectively controls the amount that we truncate the PinT correction. The more truncation that we have, the greater the change that we will break exactness.

Further, we could try to define a projection operation that is more like an L2 projection. In Section 6.3.2 we note that a projection like that in Algorithm 5 is really not ideal in the sense that it can move us away from the point on our stable manifold that is closest to where the PinT correction $u + \tau$ is. An ideal or better optimized projection operator might alleviate some of the errors that persist.

Toggle Projection in MGRIT+Π

On reflection on the nature of the projection problem, we see that Algorithm 7 applies a projection on every brick, every cycle. We could modify this using our knowledge of exactness. If a brick B_i is exact on iteration $k > i$, then we know that $U + \tau$ should not need projection. We could modify our algorithm such that if $i > k$, then we would proceed to use MGRIT+Π as defined in Algorithm 7, but if we know exactness should happen to revert to MGRIT without projection like Algorithm 4. Perhaps this "toggle" of Π would allow the errors visible on early bricks in Figure 5.7 to be ironed out, and for exactness to be thusly restored.

Some possibly unclear questions in this are: Do we toggle on every level of MGRIT+Π, or only toggle on the finest level? Does the exactness property extend to coarse levels in that if we know a fine brick ought to be exact, then is a coarser brick also exact?

Projection Based on Physics

Our projection views the state U as a vector, and corrects based on the difference in this vector between a fine and a coarse level. However, each component of U is actually coupled to the others. Furthermore, our stable set $\mathcal{M}_*$ is based on physical constraints. Perhaps then we can

define a projection Π that is more physics informed in the sense that it is aware that the correct projected state $\Pi(U+\tau)$ might not be just the closest point on the set $\mathcal{M}$, naively defined, but one that respects physics. The PinT literature already has an example of this kind of physical thinking where researchers try to find optimal coarse propagators for hyperbolic PDEs by looking at propagating along characteristics in the PDEs [50, 57].

Local Projection

Finally, one idea to work on the projection operator as it exists in Algorithm 5 would be to swap out the global reference ρ_{ref} to a local one. This might allow the densities to be smaller in regions where they need to be small, and therefore allowing Π to be tighter to what $U+\tau$ is before projection. We actually did try this for a while, but found that we cannot expect that this local projection truly is a projection since $P^2(U+\tau)$ might not equal $P(U+\tau)$. (In the Algorithm 5, a global reference is used, so we can guarantee that $P^2 = P$.) However, it is possible that we do not really need a true projection operation for MGRIT+Π.

8.3.2 Study MGRIT Options

The MGRIT algorithm allows us to play with a few multilevel algorithm options. One option to play with would be the relaxation scheme used in MGRIT+Π. The other would be type of cycling or defining how we visit each of the coarser levels (we used a V-cycle Figure 3.4)

Different Relaxation Schemes

One of the unexplained parts of Chapter 7 was that the integrator coarsening bounced between two states after about 13 cycles (Figure 7.5). One thing we can do to try to make this behave better is to use a weighted relaxation where we smear out the state on each brick using the surrounding bricks. Some literature in this vein exists and show that a weighted relaxation performs better in certain situations than one that is not weighted [12], like we use in this dissertation. The idea here is that we can use information from bricks in the past and future to inform our construction of the state in the present. The ratio of what we take from the past/future/present would be the weights

we could tune for our time averaged computations. One could also deign schemes where the new state is a weighted average of the old state U and the proposed new state $U + \tau$.

Use Different MG Cycles

The way in which corrections move through the PinT hierarchy is probably quite important for convergence properties of MGRIT+Π. V-cycles are a natural starting point, but we should also explore other ways to visit other levels. An easy direction to extend the results in this thesis would be to try out other cycling shapes like the F-cycle (see Figure 3.5), or a W-cycle. It is possible that these cycle shapes could have different convergence properties, especially considering our projection scheme that we inject, than the V-cycles tested herein.

8.3.3 Algorithmic Extensions

Two areas to refine the MGRIT+Π algorithm would be to 1) study the effects of using both r-type and i-type coarsening simultaneously and 2) to define a convergence criterion useful to our goal of computing time averaged quantities of interest.

r-type and i-type Simultaneous Coarsening

We noted in Section 8.2 that the r-type coarsening seems to be the best bet for achieving a speedup over sequential time integration methods that actually converge to an answer, even though it incurs an interpolation cost, the full effects of which we saw in Section 7.2. Since we pay this interpolation cost anyway, we could make our coarser r-type levels faster to evaluate if we also used i-type MGRIT+Π coarsening at the same time. Consider, for example, a hierarchy of spatial meshes r456 where the integrator is different on each level like i123.

It would be useful to combine the two and see if this provides a closer speedup since coarsening in space and in integration method means less DOFs to loop over and fewer Runge-Kutta sub-step evaluations simultaneously resulting in a much coarser problem overall. Recall that PinT methods really only have hope to provide wall-clock speedup if G is much cheaper to evaluate than F, see Section 3.1.

Convergence Criterion

In Chapter 7 we ran our MGRIT+Π code for 30 cycles without checking for any convergence. Doing this will reduce any possible wall-clock gain to naught. We observed heuristically that 13 MG cycles was sufficient for convergence for out time averaged measurements. Future work remains to devise a meaningful convergence criterion to be used to stop MG cycling.

8.3.4 Extend Research to Other PDEs and Dimensions

MGRIT+Π should explore applications in other dimensions and in other PDEs. We have already proposed using a simple 1D problem to study the interaction of projection and PinT, but it would be nice to perform measurements in 3D. Furthermore, as we noted in Section 7.2 it would also be good to understand how MGRIT+Π behaves in the full Navier-Stokes equations. This could be beneficial to PinT computations as the literature notes that physical dissipation helps MGRIT convergence, even without any time averaging [19, 52].

8.4 Summary

PinT methods remain basically academic. I have not worked at a national lab, private company, or institution where PinT methods are even widely known, much less actively used. A major reason for this is PinT's overwhelming use in systems that are not all that interesting. (This is my opinion.) Another major reason is that there has not been a convincing result of actual wall-clock speedup using PinT methods. (Again, this is my opinion).

The results in this dissertation help to push PinT methods to their limit in those more interesting systems. Time averaged quantities are useful to engineers, and, up to some small error, can be accurately computed with MGRIT+Π. This fact helps to expand PinT's usefulness to more PDEs than previous PinT research.

The second issue, a lack of convincing speedup, remains a daunting fact of the matter for PinT methods. However, if PinT is to survive, results like this dissertation are helpful to keep people aware of the constraints of PinT.

Bibliography

[1] Charles E Leiserson et al. “There’s plenty of room at the Top: What will drive computer performance after Moore’s law?” In: *Science* 368.6495 (2020), eaam9744.

[2] Martin J. Gander. “50 Years of Time Parallel Time Integration”. In: *Multiple Shooting and Time Domain Decomposition Methods*. Springer International Publishing, 2015, pp. 69–113. ISBN: 9783319233215. DOI: 10.1007/978-3-319-23321-5_3.

[3] Jürg Nievergelt. “Parallel methods for integrating ordinary differential equations”. In: *Communications of the ACM* 7.12 (1964), pp. 731–733.

[4] Pierluigi Amodio and Luigi Brugnano. “Parallel solution in time of ODEs: some achievements and perspectives”. In: *Applied Numerical Mathematics* 59.3 (2009). Selected Papers from NUMDIFF-11, pp. 424–435. ISSN: 0168-9274. DOI: https://doi.org/10.1016/j.apnum.2008.03.024. URL: https://www.sciencedirect.com/science/article/pii/S0168927408000378.

[5] Giang D. Huynh and Reza Abedi. “Tent-pitcher spacetime discontinuous Galerkin method for one-dimensional linear hyperbolic and parabolic PDEs”. In: *Computers & Mathematics with Applications* 148 (Oct. 2023), pp. 26–40. ISSN: 0898-1221. DOI: 10.1016/j.camwa.2023.07.021.

[6] Elena Celledoni and Trond Kvamsdal. “Parallelization in time for thermo-viscoplastic problems in extrusion of aluminium”. In: *International Journal for Numerical Methods in Engineering* 79.5 (Mar. 2009), pp. 576–598. ISSN: 1097-0207. DOI: 10.1002/nme.2585.

[7] Robert D. Falgout et al. “Multigrid reduction in time with Richardson extrapolation”. In: *Electronic Transactions on Numerical Analysis* 54 (Feb. 2021), pp. 210–233. ISSN: 1068-9613. DOI: 10.1553/etna_vol54s210. URL: https://www.osti.gov/biblio/1838604.

[8] Xiaoying Dai and Yvon Maday. “Stable Parareal in Time Method for First-and Second-Order Hyperbolic Systems”. In: *SIAM Journal on Scientific Computing* 35.1 (Jan. 2013), A52–A78. ISSN: 1095-7197. DOI: 10.1137/110861002.

[9] Thibaut Lunet et al. “Time-parallel simulation of the decay of homogeneous turbulence using Parareal with spatial coarsening”. In: *Computing and Visualization in Science* 19.1–2 (May 2018), pp. 31–44. ISSN: 1433-0369. DOI: 10.1007/s00791-018-0295-0.

[10] David. A. Vargas et al. “Multigrid Reduction in Time for Chaotic Dynamical Systems”. In: *SIAM Journal on Scientific Computing* 45.4 (Aug. 2023), A2019–A2042. DOI: 10.1137/22m1518335.

[11] Federico Danieli and Scott MacLachlan. “Multigrid reduction in time for non-linear hyperbolic equations”. In: *ETNA -Electronic Transactions on Numerical Analysis* 58 (2022), pp. 43–65. DOI: 10.1553/etna_vol58s43.

[12] Masumi Sugiyama et al. “Weighted relaxation for multigrid reduction in time”. In: *Numerical Linear Algebra with Applications* 30.1 (Sept. 2022), e2465. DOI: 10.1002/nla.2465.

[13] Joshua C. Christopher. “Time integration for complex fluid dynamics”. PhD thesis. Colorado State University, 2021.

[14] David E. Keyes. “Exaflop/s: The why and the how”. In: *Comptes Rendus. Mécanique* 339.2-3 (Feb. 2011), pp. 70–77. ISSN: 1873-7234. DOI: 10.1016/j.crme.2010.11.002.

[15] Jack Dongarra et al. *Applied Mathematics Research for Exascale Computing*. U.S. Department of Energy Office of Scientific and Technical Information, Feb. 2014. DOI: 10.2172/1149042.

[16] Martin J. Gander and Thibaut Lunet. *Time Parallel Time Integration*. Society for Industrial and Applied Mathematics, Jan. 2024. ISBN: 9781611978025. DOI: 10.1137/1.9781611978025.

[17] Alberto Sangiovanni- Vincentelli and Jacob White. “Waveform relaxation techniques and their parallel implementation”. In: *1985 24th IEEE Conference on Decision and Control*. IEEE, Dec. 1985, pp. 1544–1551. DOI: 10.1109/cdc.1985.268773.

[18] Mariesa L. Crow and Marija D. Ilic. “The parallel implementation of the waveform relaxation method for transient stability simulations”. In: *IEEE Transactions on Power Systems* 5.3 (1990), pp. 922–932. ISSN: 0885-8950. DOI: 10.1109/59.65922.

[19] James Jackaman and Scott MacLachlan. *Space-time waveform relaxation multigrid for Navier-Stokes*. July 19, 2024. arXiv: 2407.13997v1.

[20] Ben S. Southworth. private conversation. July 2024.

[21] Alfredo Bellen and Marino Zennaro. “Parallel algorithms for initial-value problems for difference and differential equations”. In: *Journal of Computational and Applied Mathematics* 25.3 (May 1989), pp. 341–350. ISSN: 0377-0427. DOI: 10.1016/0377-0427(89)90037-x.

[22] Jacques- Louis Lions, Yvon Maday, and Gabriel Turinici. “Résolution d’EDP par un schéma en temps «pararéel»”. In: *Comptes Rendus de l’Académie des Sciences -Series I -Mathematics* 332.7 (Apr. 2001), pp. 661–668. ISSN: 0764-4442. DOI: 10.1016/s0764-4442(00)01793-6.

[23] Philippe Chartier and Bernard Philippe. “A parallel shooting technique for solving dissipative ODE’s”. In: *Computing* 51.3–4 (Sept. 1993), pp. 209–236. ISSN: 1436-5057. DOI: 10.1007/bf02238534.

[24] Robert D. Falgout et al. “Parallel Time Integration with Multigrid”. In: *SIAM Journal on Scientific Computing* 36.6 (Jan. 2014), pp. C635–C661. ISSN: 1095-7197. DOI: 10.1137/130944230.

[25] Manfred Ries, Ulrich Trottenberg, and Gerd Winter. “A note on MGR methods”. In: *Linear Algebra and its Applications* 49 (Feb. 1983), pp. 1–26. ISSN: 0024-3795. DOI: 10.1016/0024-3795(83)90091-5.

[26] Matthew Emmett and Michael Minion. “Toward an efficient parallel in time method for partial differential equations”. In: *Communications in Applied Mathematics and Computational Science* 7.1 (Mar. 2012), pp. 105–132. ISSN: 1559-3940. DOI: 10.2140/camcos.2012.7.105.

[27] Roberto Croce, Daniel Ruprecht, and Rolf Krause. “Modeling, Simulation and Optimization of Complex Processes -HPSC 2012”. In: *Modeling, Simulation and Optimization of Complex Processes -HPSC 2012*. Springer International Publishing, 2014. Chap. Parallel-in-Space-and-Time Simulation of the Three-Dimensional, Unsteady Navier-Stokes Equations for Incompressible Flow, pp. 13–23. ISBN: 9783319090634. DOI: 10.1007/978-3-319-09063-4_2.

[28] Debasmita Samaddar, David E. Newman, and Raul Sánchez. “Parallelization in time of numerical simulations of fully-developed plasma turbulence using the parareal algorithm”. In: *Journal of Computational Physics* 229.18 (Sept. 2010), pp. 6558–6573. ISSN: 0021-9991. DOI: 10.1016/j.jcp.2010.05.012.

[29] Martin J. Gander, Felix Kwok, and Julien Salomon. “PARAOPT: A Parareal Algorithm for Optimality Systems”. In: *SIAM Journal on Scientific Computing* 42.5 (Jan. 2020), A2773–A2802. ISSN: 1095-7197. DOI: 10.1137/19m1292291.

[30] Frédéric Legoll, Tony Lelièvre, and Upanshu Sharma. “An Adaptive Parareal Algorithm: Application to the Simulation of Molecular Dynamics Trajectories”. In: *SIAM Journal on Scientific Computing* 44.1 (Jan. 2022), B146–B176. ISSN: 1095-7197. DOI: 10.1137/21m1412979.

[31] Leonardo Baffico et al. “Parallel-in-time molecular-dynamics simulations”. In: *Physical Review E* 66.5 (Nov. 2002). ISSN: 1095-3787. DOI: 10.1103/physreve.66.057701.

[32] Guillaume Bal and Yvon Maday. “A parareal time discretization for nonlinear PDE’s with application to the pricing of an American put”. In: *Recent Developments in Domain Decomposition Methods*. Springer Berlin Heidelberg, 2002. ISBN: 9783642561184. DOI: 10.1007/978-3-642-56118-4.

[33] Yvon Maday. “The parareal in time algorithm”. In: *Not Published in a Journal* (2008).

[34] Martin J. Gander and Ernst Hairer. “Domain Decomposition Methods in Science and Engineering XVII”. In: *Domain Decomposition Methods in Science and Engineering XVII*.

Springer Berlin Heidelberg, 2008. Chap. Nonlinear Convergence Analysis for the Parareal Algorithm, pp. 45–56. ISBN: 9783540751991. DOI: 10.1007/978-3-540-75199-1_4.

[35] Veselin A. Dobrev et al. “Two-Level Convergence Theory for Multigrid Reduction in Time (MGRIT)”. In: *SIAM Journal on Scientific Computing* 39.5 (Jan. 2017), S501–S527. DOI: 10.1137/16m1074096.

[36] Van E. Henson. “Multigrid methods nonlinear problems: an overview”. In: *Computational Imaging*. Ed. by Charles A. Bouman and Robert L. Stevenson. SPIE, June 2003, pp. 36–48. DOI: 10.1117/12.499473.

[37] Achi Brandt and Oren E. Livne. *Multigrid Techniques: 1984 Guide with Applications to Fluid Dynamics, Revised Edition*. Society for Industrial and Applied Mathematics, Jan. 2011. ISBN: 9781611970753. DOI: 10.1137/1.9781611970753.

[38] XBraid Principal Developers. *XBraid Users' Manual*. v3.1.1. Lawrence Livermore National Laboratory. Mar. 2023. URL: https://github.com/XBraid/xbraid/releases.

[39] Robert D. Falgout et al. “Parallel time integration with multigrid reduction for a compressible fluid dynamics application”. In: *Lawrence Livermore National Laboratory Technical Report, LLNL-JRNL-663416* (2015).

[40] Andreas Hessenthaler et al. “Multilevel Convergence Analysis of Multigrid-Reduction-in-Time”. In: *SIAM Journal on Scientific Computing* 42.2 (Jan. 2020), A771–A796. ISSN: 1095-7197. DOI: 10.1137/19m1238812.

[41] Alok Dutt, Leslie Greengard, and Vladimir Rokhlin. “Spectral Deferred Correction Methods for Ordinary Differential Equations”. In: *Bit Numerical Mathematics* 40.2 (2000), pp. 241–266. ISSN: 0006-3835. DOI: 10.1023/a:1022338906936.

[42] Pietro Benedusi, Michael L. Minion, and Rolf Krause. “An experimental comparison of a space-time multigrid method with PFASST for a reaction-diffusion problem”. In: *Computers & Mathematics with Applications* 99 (Oct. 2021), pp. 162–170. ISSN: 0898-1221. DOI: 10.1016/j.camwa.2021.07.008.

[43] Sebastian Götschel and Michael L. Minion. “Domain Decomposition Methods in Science and Engineering XXIV”. In: *Domain Decomposition Methods in Science and Engineering XXIV*. Springer International Publishing, 2018. Chap. Parallel-in -Time for Parabolic Optimal Control Problems Using PFASST, pp. 363–371. ISBN: 9783319938738. DOI: 10.1007/978-3-319-93873-8_34.

[44] Joshua Christopher et al. “A space-time parallel algorithm with adaptive mesh refinement for computational fluid dynamics”. In: *Computing and Visualization in Science* 23.1-4 (Sept. 2020), A13.1–A13.20. DOI: 10.1007/s00791-020-00334-1.

[45] Martin Gander and Madalina Petcu. “Analysis of a Krylov subspace enhanced parareal algorithm for linear problems”. In: *ESAIM: Proceedings* 25 (2008). Ed. by E. Cancès, S. Faure, and B. Graille, pp. 114–129. ISSN: 1270-900X. DOI: 10.1051/proc:082508.

[46] Daniel Ruprecht. “Wave propagation characteristics of Parareal”. In: *Computing and Visualization in Science* 19.1–2 (May 2018), pp. 1–17. ISSN: 1433-0369. DOI: 10.1007/s00791-018-0296-z.

[47] Paul F. Fischer, Frédéric Hecht, and Yvon Maday. “Domain Decomposition Methods in Science and Engineering”. In: *Domain Decomposition Methods in Science and Engineering*. Springer-Verlag, 2005. Chap. A Parareal in Time Semi-implicit Approximation of the Navier-Stokes Equations, pp. 433–440. ISBN: 3540225234. DOI: 10.1007/3-540-26825-1_44.

[48] Matthias Maier and Martin Kronbichler. “Efficient parallel 3D computation of the compressible Euler equations with an invariant-domain preserving second-order finite-element scheme”. In: *ACM Transactions on Parallel Computing* 8.3 (2021), 16:1–30. DOI: 10.1145/3470637. URL: https://arxiv.org/abs/2007.00094.

[49] Jean- Luc Guermond et al. “On the implementation of a robust and efficient finite element-based parallel solver for the compressible Navier-Stokes equations”. In: *Computer Methods*

in Applied Mechanics and Engineering 389 (2022), p. 114250. DOI: 10.1016/j.cma.2021.114250. URL: https://arxiv.org/abs/2106.02159.

[50] Hans De Sterck et al. "Optimizing multigrid reduction-in-time and Parareal coarse-grid operators for linear advection". In: *Numerical Linear Algebra with Applications* 28.4 (Mar. 2021), e2367. ISSN: 1099-1506. DOI: 10.1002/nla.2367.

[51] Gunnar Andreas Staff and Einar M. Rønquist. "Domain Decomposition Methods in Science and Engineering". In: *Domain Decomposition Methods in Science and Engineering*. Vol. 40. Springer-Verlag, 2005. Chap. Stability of the Parareal Algorithm, pp. 449–456. ISBN: 3540225234. DOI: 10.1007/3-540-26825-1_46.

[52] Araz Eghbal, Andrew G. Gerber, and Eric Aubanel. "Acceleration of unsteady hydrodynamic simulations using the parareal algorithm". In: *Journal of Computational Science* 19 (Mar. 2017), pp. 57–76. ISSN: 1877-7503. DOI: 10.1016/j.jocs.2016.12.006.

[53] Qiqi Wang et al. "Towards scalable parallel-in-time turbulent flow simulations". In: *Physics of Fluids* 25.11 (Sept. 2013), p. 110818. ISSN: 1089-7666. DOI: 10.1063/1.4819390.

[54] Ann S. Almgren, John B. Bell, and William G. Szymczak. "A numerical method for the incompressible Navier-Stokes equations based on an approximate projection". In: *SIAM Journal on Scientific Computing* 17.2 (1996), pp. 358–369.

[55] Charbel Farhat et al. "Time-parallel implicit integrators for the near-real-time prediction of linear structural dynamic responses". In: *International Journal for Numerical Methods in Engineering* 67.5 (Jan. 2006), pp. 697–724. ISSN: 1097-0207. DOI: 10.1002/nme.1653.

[56] Daniel Ruprecht and Rolf Krause. "Explicit parallel-in-time integration of a linear acoustic-advection system". In: *Computers & Fluids* 59 (Apr. 2012), pp. 72–83. ISSN: 0045-7930. DOI: 10.1016/j.compfluid.2012.02.015.

[57] Hans De Sterck et al. "Efficient Multigrid Reduction-in-Time for Method-of-Lines Discretizations of Linear Advection". In: *Journal of Scientific Computing* 96.1 (May 2023), pp. 1–31. ISSN: 1573-7691. DOI: 10.1007/s10915-023-02223-4.

[58] Ben S. Southworth. “Necessary Conditions and Tight Two-level Convergence Bounds for Parareal and Multigrid Reduction in Time”. In: *SIAM Journal on Matrix Analysis and Applications* 40.2 (Jan. 2019), pp. 564–608. ISSN: 1095-7162. DOI: 10.1137/18m1226208.

[59] Terry Haut and Beth Wingate. “An Asymptotic Parallel-in-Time Method for Highly Oscillatory PDEs”. In: *SIAM Journal on Scientific Computing* 36.2 (Jan. 2014), A693–A713. ISSN: 1095-7197. DOI: 10.1137/130914577.

[60] Vyacheslav A. Bashkin et al. “Comparison of Calculated and Experimental Data on Supersonic Flow past a Circular Cylinder”. In: *Fluid Dynamics* 37.3 (May 2002), pp. 473–483. ISSN: 1573-8507. DOI: 10.1023/a:1019675027402.

[61] Forrest E. Gowen and Edward W. Perkins. *Drag of Circular Cylinders for a Wide Range of Reynolds Numbers and Mach Numbers*. Technical Report 2960. Ames Aeronautical Laboratory, Moffett Field, Calif., June 1953. URL: https://ntrs.nasa.gov/citations/19930084018.

[62] Charles Hirsch. *Numerical Computation of Internal and External Flows*. Vol. 1. Wiley, 1988. Chap. 2, pp. 34–43.

[63] Manuj Awasthi et al. “Supersonic cylinder wake dynamics”. In: *Journal of Fluid Mechanics* 945 (July 2022), A4.1–A4.45. ISSN: 1469-7645. DOI: 10.1017/jfm.2022.517.

[64] Rio Baidya et al. “Investigation of a Near-Field Cylinder Wake in the Subsonic, Transonic, and Supersonic Regimes”. In: *AIAA Journal* 61.12 (Dec. 2023), pp. 5415–5428. ISSN: 1533-385X. DOI: 10.2514/1.j063163.

[65] Manuj Awasthi et al. “Coherent oscillations and acoustic waves in a supersonic cylinder wake”. In: *Journal of Fluid Mechanics* 1012 (June 2025). ISSN: 1469-7645. DOI: 10.1017/jfm.2025.10190.

[66] Bryan E. Schmidt and Joseph E. Shepherd. “Oscillations in cylinder wakes at Mach 4”. In: *Journal of Fluid Mechanics* 785 (Nov. 2015). ISSN: 1469-7645. DOI: 10.1017/jfm.2015.668.

[67] Jacob M. Turner, Jung Hee Seo, and Rajat Mittal. "A high-order sharp-interface immersed boundary solver for high-speed flows". In: *Journal of Computational Physics* 500 (Mar. 2024), p. 112748. ISSN: 0021-9991. DOI: 10.1016/j.jcp.2023.112748.

[68] Jacob M. Turner and Jae Wook Kim. "Quadrupole noise generated from a low-speed aerofoil in near-and full-stall conditions". In: *Journal of Fluid Mechanics* 936 (Feb. 2022). ISSN: 1469-7645. DOI: 10.1017/jfm.2022.75.

[69] Antony Jameson. "Transonic, Shock, and Multidimensional Flows". In: *Transonic, Shock, and Multidimensional Flows*. Ed. by RIchard E. Meyer. Academic Press, 1982, pp. 37–70. ISBN: 978-0-12-493280-7. DOI: 10.1016/b978-0-12-493280-7.50008-5.

[70] Erdal Oktay and Cem O. Asma. "Fast Drag Prediction Method Using Euler Equations". In: *Journal of Spacecraft and Rockets* 37.5 (Sept. 2000), pp. 692–697. DOI: 10.2514/2.3620.

[71] Philip L. Roe. "Characteristic-Based Schemes for the Euler Equations". In: *Annual Review of Fluid Mechanics* 18.1 (Jan. 1986), pp. 337–365. ISSN: 1545-4479. DOI: 10.1146/annurev.fl.18.010186.002005.

[72] Sergei K. Godunov and Ihor O. Bohachevsky. "Finite difference method for numerical computation of discontinuous solutions of the equations of fluid dynamics". In: *Matematičeskij sbornik* 47(89).3 (1959), pp. 271–306. URL: https://hal.science/hal-01620642.

[73] Philip L. Roe. "Approximate Riemann solvers, parameter vectors, and difference schemes". In: *Journal of Computational Physics* 43.2 (Oct. 1981), pp. 357–372. ISSN: 0021-9991. DOI: 10.1016/0021-9991(81)90128-5.

[74] Jake Harmon et al. "A conservative invariant-domain preserving projection technique for hyperbolic systems under adaptive mesh refinement". In: *arXiv:2507.18717* (2025). DOI: 10.48550/ARXIV.2507.18717.

[75] Lloyd R. Jenkinson, Darren Rhodes, and Paul Simpkin. *Civil jet aircraft design*. eng. AIAA education series. Reston, VA: American Institute of Aeronautics and Astronautics, 1999.

ISBN: 156347350X. URL: https://booksite.elsevier.com/9780340741528/appendices/data-a/table-3/table.htm.

[76] Marshall C. Galbraith et al. “Comparing Multi-Element Airfoil Flow Solutions Using Multiple Solvers with Output-Based Adapted Meshes”. In: *AIAA Journal* 60.4 (Apr. 2022), pp. 2629–2643. ISSN: 1533-385X. DOI: 10.2514/1.j060861.

[77] Jerett Cherry. *MGRIT+Pi*. https://github.com/jerett-cc/ryujin.git. 2025.

[78] Stephen M. Guzik et al. “On the use of a multigrid-reduction-in-time algorithm for multiscale convergence of turbulence simulations”. In: *Computers & Fluids* 261 (July 2023), p. 105910. ISSN: 0045-7930. DOI: 10.1016/j.compfluid.2023.105910.