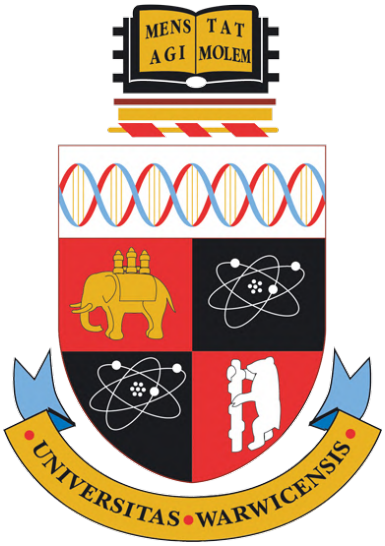
Towards probabilistic time-parallel algorithms for solving initial value problems



Kamran Pentland

A thesis submitted in fulfilment of the requirements

for the degree of

Doctor of Philosophy

Mathematics for Real-World Systems CDT

University of Warwick

September 2023

Contents

List of acronyms

FHN	FitzHugh-Nagumo
FLOPS	Floating Point Operations Per Second
GP	Gaussian Process
HPC	High Performance Computer/Computing
i.i.d.	Independent Identically Distributed
IVP	Initial Value Problem
LHS	Latin Hypercube Sampling
MPSD	Maximal Posterior Standard Deviation
NN	Neural Network
ODE	Ordinary Differential Equation
PC	Predictor-Corrector
PDE	Partial Differential Equation
PINN	Physics-Informed Neural Network
PinT	Parallel-in-Time
PN	Probabilistic Numerics
RK	Runge-Kutta
RKHS	Reproducing Kernel Hilbert Space
SDE	Stochastic Differential Equation
SE	Square Exponential

List of figures

List of tables

List of algorithms

Acknowledgements

Before we get into the thick of it, I must dedicate the next page or so to all those who have directly (and indirectly) contributed to the writing of this thesis. While these acknowledgements only account for approximately 1% of the total pages in this thesis, the contribution of everyone mentioned below (and those I have inevitably forgotten) is truly immeasurable.

First and foremost I must thank my parents for their endless help over the past few years as I have pursued this PhD. Even though you may not fully understand what this research is about (I will explain again don't worry), your continual support has meant everything to me and if not for you both, I would certainly not be where I am today. Secondly, I want to thank Melissa for having to put up with me 24/7, not just in the office, but at home and everywhere in between. I cannot overstate how much your help and support in all manner of things has meant to me. I suppose I should also shout out my brother Tarun on the other side of the world and thank him for continually asking me if I'm "...finished with the PhD yet?".

I want to also thank both of my supervisors at Warwick, Dr. Massimiliano Tamborrino and Dr. Tim Sullivan, for their guidance and insight over the past few years as we have attempted to traverse the trials and tribulations of parallel-in-time methods. I am immensely grateful for your help with this research and for your patience and encouragement when I (frequently) ran into dead ends. I would also like to thank you for dealing with the copious amounts of administrative work that comes with having a PhD student and for pushing me out of my comfort zone to speak in various seminars and workshops over the years. Many thanks must also go to my external collaborators at the Culham Centre for Fusion Energy: Dr. Lynton Appel, Dr. James Buchanan, and Dr. Debasmita Samaddar. Your comments and suggestions on how I can improve my work and our (sometimes lengthy!) discussions about Parareal, SParareal, and GParareal have been invaluable. To all, I hope we stay in touch and continue to work together in the future.

Outside of Warwick, I would like to express my sincere thanks to Dr. Jemma Shipton, Dr. Sebastian Götschel, and Prof. Chris Oates for inviting me to speak at their respective workshops and conferences around Europe. I really enjoyed meeting everyone in both the parallel-in-time and probabilistic numerics communities and

I have no doubt that this will lead to collaborative work in the future. My thanks and appreciation must also go to my examiners Dr. Jere Koskela and (again) Dr. Sebastian Götschel for performing the unenviable task of reading this entire thesis and for taking the time to provide me with insightful comments and corrections.

To my friends in office D1.13, thank you for the year-round entertainment and procrastination sessions, of which there were almost certainly too many. I am particularly proud of how overly elaborate our world cup, eurovision, and snooker sweepstakes scoring systems became and I'm almost certain this is the reason why I didn't win any of them! To my friends outside Warwick, thank you for providing me with well-needed escapes from time to time and reminding me that there's more to life than work. Lastly, thanks go to my friends in MathSys, the Mathematics Institute, and Department of Statistics both past and present: long may the tradition of 5-a-side football continue!

Finally, I would like to thank the staff in MathSys for providing a relaxed inclusive working environment and for their tireless efforts in holding the CDT together during some especially trying times. I am also grateful to the Engineering and Physical Sciences Research Council, the Culham Centre for Fusion Energy, the EUROfusion Consortium and Euratom for funding this research. I would also like to thank the University of Warwick Scientific Computing Research Technology Platform for providing access to their high performance computing facilities, without which none of the numerical results in this thesis would exist.

Declarations

I declare that the material contained in this thesis is my own work (except where otherwise indicated, cited, or commonly known) and has not been submitted for a degree at another university.

Chapters 1 and 2 contain snippets from each of the following works.

Chapter 3 is formed of work from:

- K. Pentland, M. Tamborrino, D. Samaddar, and L. C. Appel. Stochastic parareal: An application of probabilistic methods to time-parallelization. *SIAM Journal on Scientific Computing*, 45(3):S82–S102, 2023a. doi:10.1137/21M1414231.

Chapter 4 is formed of work from:

- K. Pentland, M. Tamborrino, and T. J. Sullivan. Error bound analysis of the stochastic parareal algorithm. *SIAM Journal on Scientific Computing*, 45(5): A2657–A2678, 2023b. doi:10.1137/22M1533062.

Chapter 5 is formed of work from (except Section 5.4, which is unpublished):

- K. Pentland, M. Tamborrino, T. J. Sullivan, J. Buchanan, and L. C. Appel. GParareal: a time-parallel ODE solver using Gaussian process emulation. *Statistics and Computing*, 33(1):23, 2023c. doi:10.1007/s11222-022-10195-y.

Chapter 6 is formed of unpublished work.

Abstract

This thesis concerns the development of probabilistic time-parallel algorithms for solving initial value problems (IVPs) that are computationally expensive to simulate using traditional (serial) time-stepping methods. We begin by considering Parareal, a well-studied deterministic time-parallel algorithm that combines solutions from cheap (coarse) and expensive (fine) time-steppers within a predictor-corrector (PC) scheme, to solve the IVP in parallel. Our goal is to derive, analyse, and test our own probabilistic time-parallel algorithms that incorporate sampling- and learning-based techniques from the field of probabilistic numerics into Parareal. These techniques enable us to exploit valuable information contained within the fine and coarse solution data generated during a Parareal simulation. We aim to accelerate the convergence of Parareal (i.e. increase numerical speedup), generate probabilistic solutions to the IVPs (to quantify numerical uncertainty explicitly), and verify the accuracy of these solutions both numerically and analytically.

We first propose *SPararealal*, a sampling-based algorithm that provides the PC with candidate solution values drawn from probability distributions constructed using the *most recent* fine and coarse solution data. Increased sampling in SPararealal leads to accelerated convergence vs. Parareal for low-dimensional IVPs, returning stochastic solutions that are accurate (in the mean-square sense) with respect to the (exact) serially obtained fine solver solution. Next, we propose *GPararealal*, a learning-based algorithm that models part of the PC using a Gaussian process emulator, trained on *all* previously collected fine and coarse solution data. GPararealal achieves accelerated convergence for low to moderately sized IVPs, attains accurate solutions, and has the ability to re-use legacy solution data from prior simulations—something that existing time-parallel methods do not do. After introducing both algorithms, we investigate their performance and analyse their limitations, assessing whether or not they are viable methods for solving large-scale IVPs in parallel and discussing what can be done to improve them in their current form.

"For the scientist, at exactly the moment of discovery—that most unstable existential moment—the external world, nature itself, deeply confirms his [or her] innermost fantastic convictions. Anchored abruptly in the world, Leviathan gasping on his hook, he is saved from extreme mental disorder by the most profound affirmation of the real."

Richard Rhodes, *The Making of the Atomic Bomb*

Chapter 1

Introduction

This thesis concerns the development of probabilistic time-parallel algorithms for solving initial value problems (IVPs). The principle aim is to investigate whether we can embed techniques from the field of *probabilistic numerics* (PN) into the framework of existing *parallel-in-time* (PinT) algorithms. The key idea is to more effectively utilise the IVP solution data obtained throughout a parallel-in-time simulation to reduce simulation runtimes and (perhaps) better quantify solution uncertainty.

In this chapter, we give a general introduction to both PinT algorithms and PN methods, providing an overview of how they work and have developed, and motivate why research in these areas is important in the context of solving IVPs. We will describe which PN methods we plan to incorporate into the framework of a popular PinT algorithm, provide a detailed summary of our principal research aims, and outline the rest of this thesis.

1.1 Parallelism for differential equations

1.1.1 Why use parallel computing?

In its most basic form, parallel computing is the process by which a computationally expensive task is partitioned into a number of cheaper sub-tasks that can be solved simultaneously without prior knowledge of one another (Trobec et al., 2018). By computationally expensive, we mean that the task takes a long time to solve, e.g. minutes, hours, or days. The goal is to reduce the total time taken to solve this expensive task, which is run on a single compute *core*, by re-distributing the computational load over a number of cores that can run in parallel at the same time. For example, consider the time-consuming task of trying to invert a number of very large matrices using a single core—clearly this task will be completed much faster by assigning different matrices to their own cores.

A modern laptop computer typically contains a single *processor*, e.g. a central processing unit, made up of between $4 - 20$ compute cores (hence why they are

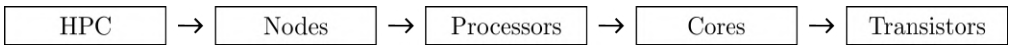
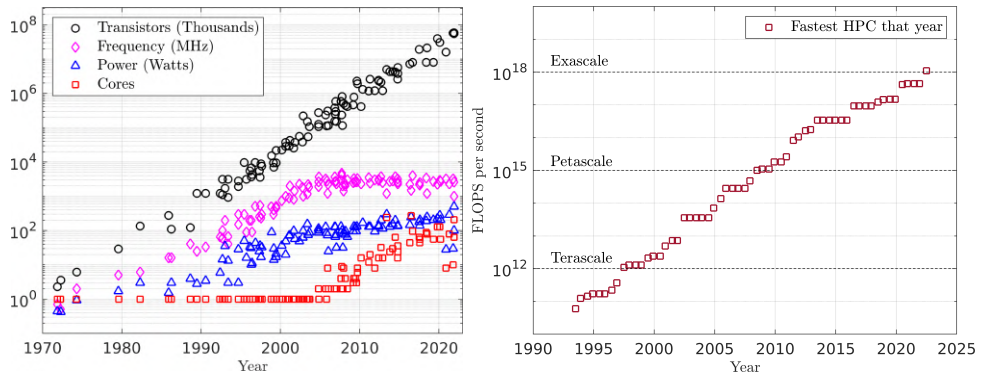


Figure 1.1: A hierarchy of computing terminology. A *high performance computer* (HPC) cluster is a collection of compute *nodes* (sometimes referred to as servers) linked together by fast interconnects. These nodes typically contain a few *processors*, each made up of a number of compute *cores* that carry out computations. Each of these cores are, in turn, made up of billions of *transistors*.

often referred to as a multi-core processors). These cores carry out computations on our behalf via a complex network of billions of *transistors*—see Figure 1.1 for a hierarchy of computing terminology. In Figure 1.2(a), we can see that (historically) the number of transistors that fit onto a core doubled approximately every two years (in line with Moore's law (Moore, 1965)). However, processors clock speeds (i.e. the typical frequency at which computations are carried out) have plateaued over the past twenty years or so. This is because faster (serial) computations require more power, generating heat that can damage the core if not properly cooled (which would require even more power). Focus has therefore turned to parallel computing where multi-core processors, with more balanced clock speeds and power consumption, can be used to carry out multiple serial computations in parallel. Parallel computing is becoming increasingly necessary in a number of disciplines to reduce excessive simulation runtimes and overcome the aforementioned physical limitations arising on machine hardware. These trends, as well as Moore's law coming to an end (Shalf, 2020), are driving the need to develop new algorithms, or convert existing serial ones, that can exploit parallel computing architectures.

High performance computers (HPCs), perhaps more commonly known as supercomputers or computing clusters, are the typical platform for deploying large-scale parallel algorithms and contain a number of compute *nodes*, each made up of hundreds, if not thousands, of interlinked compute cores. The performance of the fastest HPCs in the world, typically measured by the number of floating point operations per second (FLOPS) taken to solve a very large dense linear system (i.e. $A\boldsymbol{x} = \boldsymbol{b}$), is increasing year on year—see Figure 1.2(b). For reference, a typical modern laptop can carry out $\mathcal{O}(10^{12})$ FLOPS while the fastest HPC in the world[1] contains over 8 million cores and can carry out $\mathcal{O}(10^{18})$ FLOPS. HPCs are currently used to simulate solutions to some of the world's most computationally demanding problems in numerical weather prediction (ECMWF, 2023), genome sequencing (Yale, 2023), machine learning (Sevilla et al., 2022), and plasma physics (Dudson et al., 2009). Numerical simulations, often referred to as the third pillar of science (alongside theory and experimentation), aid the study of physical phenomena that are analytically or

[1] As of 2023, the fastest HPC in the world was *Frontier*. Hosted at the Oak Ridge Leadership Computing facility in Tennessee, United States, it was the first HPC to breach the exascale barrier—more details available at https://www.olcf.ornl.gov/frontier/.



(a) (b)

Figure 1.2: (a) The history of computer processor properties. Shown are the number of transistors that can fit onto a compute core (black circles), typical clock speeds/frequencies of a processor (purple diamonds), typical power consumption of a processor (blue triangles), and the number of cores that fit onto a given processor (red squares). (b) The performance of the fastest high performance computer each year since 1993, measured in terms of floating-point operations per second (FLOPS). Data sources: (a) Rupp (2022) and (b) TOP500Project (2022).

experimentally intractable. These problems can take a very long time to simulate (on the order of hours, days, and weeks), even being computationally infeasible in some cases, and so there is a demand for algorithms that can exploit the maximum possible degree of parallelism to reduce simulation runtimes. At present, HPCs with increasing numbers of compute cores are being built to tackle these gargantuan tasks, however, we have yet to fully develop the necessary parallel algorithms that can exploit such architectures. The development of faster and easier-to-use parallel algorithms is becoming increasingly important[2] and so a particular focus of this thesis will be on improving existing parallel algorithms for solving differential equations. Note that throughout this thesis, we will slightly abuse terminology by instead referring to "compute cores" as "processors"—a common switch made throughout parallel computing literature.

1.1.2 What are parallel-in-time methods?

The bedrock of many complex models in science involve solving systems of ordinary, partial, or stochastic differential equations (ODEs, PDEs, or SDEs) using numerical methods (Danby, 1997; Kloeden and Platen, 1992; Trefethen et al., 2017). Of particular interest to us are IVPs, which model the evolution of some quantity of interest over time (as well as space for PDEs) from a single *initial condition* or *initial state* under the action of a differential equation. To solve an IVP numerically, one

[2]The *ExCALIBUR* research programme is aiming to deliver the next generation of HPC software in areas such as particle hydrodynamics, materials simulation, and plasma turbulence in fusion modelling. Further use cases can be found at https://excalibur.ac.uk/.

typically uses a time-stepping scheme (more on these in Chapter 2), in which the solution state of the IVP at a given time step is used to approximate the solution at the next time step—an inherently sequential process that is carried out on a single processor. The computational time taken to solve an IVP depends on a myriad of factors including, but not limited to, its type (linear/nonlinear/stiff/non-stiff), size (number of equations), the accuracy of the time-stepping method, the length of the interval of integration, and the number of discrete time steps taken (in addition to a discrete spatial mesh, if present). We are interested in problems where the interval of integration and/or the number of time steps required to solve the problem is large, leading to lengthy simulation runtimes.

For IVPs that have a spatial component (typically spatio-temporal PDEs), boundary conditions are prescribed on the spatial domain and so one can solve the problem in space either serially or in parallel, using well established *domain decomposition methods* (Quarteroni and Valli, 1999; Toselli and Widlund, 2005). Domain decomposition methods partition the spatial domain into smaller subdomains, solving on each subdomain in parallel using its own processor (with the aid of boundary conditions[3]), and then combine the calculated solutions at the subdomain interfaces (which can overlap) in an iterative manner—see Dolean et al. (2015) for further details. Although very efficient for high dimensional systems, yielding high parallel speedup, spatial parallelism eventually begins to saturate, i.e. using additional processors does not result in increasing speedup due to excessive overhead serial communication costs between the processors. The total numerical speedup gained from the spatial parallelism will then bottleneck in the temporal domain, limited by the serial nature of the time-stepping scheme in use. For example, modern algorithms used to simulate Edge Localised Modes in turbulent fusion plasmas can take anywhere between 100-200 days to integrate over a time interval of just one second (Samaddar et al., 2019).

Sequential bottlenecks in time have motivated the research and development of PinT methods, providing ways to integrate IVPs over long time intervals where solutions would be unobtainable (in realistic time-frames) using sequential time-stepping schemes. One way to integrate in a non-sequential manner, similar to spatial parallelisation, is to discretise the time interval of an IVP into N "slices" upon which N smaller IVPs are solved in parallel using existing sequential time-stepping schemes. The principle of *causality*, however, poses a fundamental problem. When solving IVPs, solution states at later times are determined by solution states at earlier times, starting from some prescribed initial condition at the beginning of the interval of integration. Therefore, prior to the parallel integration of the N smaller IVPs, we require N initial values from which to begin integration in each time slice, of which $N-1$ are unknown *a priori*—see Figure 1.3. This contrasts with spatially-

[3]Boundary conditions are typically constraints on the value (e.g. Dirichlet conditions) and/or the derivatives (e.g. Neumann conditions) of the solution at the boundaries of the spatial domain.

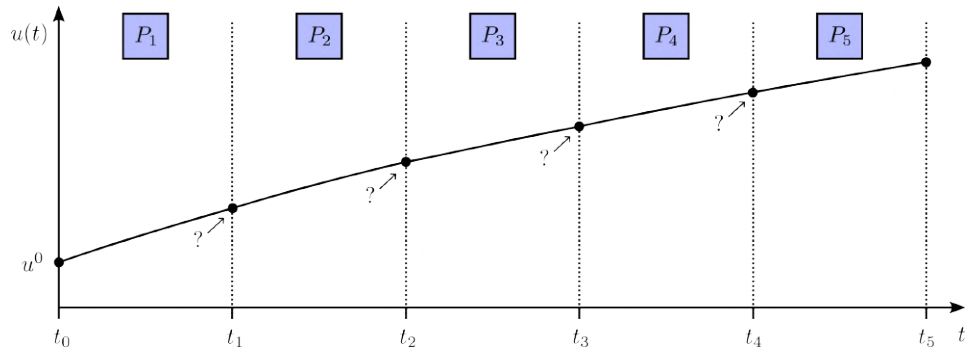


Figure 1.3: Exact solution $u(t)$ to a scalar ODE at discrete times $t_0, \ldots, t_5$ (black dots), typically obtained using a serial time-stepping scheme on one processor. To calculate the solution in parallel, each time slice $[t_{n-1}, t_n]$ is assigned its own processor P_n and each smaller IVP is solved—the issue being that $N-1$ of the N initial values from which to begin integration are unknown *a priori* to simulation.

parallel methods which can take advantage of the fact that there are typically at least two boundary conditions prescribed in space, e.g. left and right boundaries in one-dimension. Using domain decomposition methods, the solution can be calculated from the left and right boundary conditions simultaneously and iteratively matched across some overlapping or non-overlapping interface in the middle. This not so subtle difference makes the time-parallelisation problem much less intuitive to solve and suggests why domain decomposition methods have a rich history dating back to Schwarz (1870) while PinT methods have only been studied for the past 60 years.

The first PinT algorithm was proposed by Nievergelt (1964), the idea being that one could locate approximate initial conditions for the $N-1$ smaller IVPs by solving the entire IVP using a cheap (coarse) time-stepping scheme. The N IVPs are then solved in parallel using the original (more expensive) time-stepping method and the resulting trajectories are matched across time slice boundaries using an interpolation scheme—a more detailed account of his algorithm will be given in Chapter 3. This seminal contribution spawned an entirely new field of work, where there has been extensive work on a number of different PinT methods[4] which were categorised by Gander (2015) into four (not necessarily distinct) groups, namely: multiple shooting, multigrid, waveform relaxation, and direct methods. Our focus will be on multiple shooting algorithms for ODEs (e.g. Nievergelt's method), where one partitions the time domain into a set of slices and attempts to solve in each one in parallel, enforcing continuity conditions at the slice boundaries—a number of which have been successfully developed and tested (Bellen and Zennaro, 1989; Chartier and Philippe, 1993; Lions et al., 2001; Saha et al., 1997). Within the other categories, PinT methods have been designed to solve IVPs with a hierarchy of coarser

[4] A collection of resources on PinT methods including code repositories, publications, workshops, and other materials can be found at http://parallel-in-time.org/.

time/space grids (multigrid), domain decomposition methods (waveform-relaxation), and with modified existing sequential time-stepping schemes (direct methods). For an overview of these methods and other state-of-the-art in PinT algorithms, refer to Ong and Schroder (2020). Being able to solve computationally intractable (with sequential methods) IVPs that evolve on different timescales over very long time intervals whilst possibly exhibiting metastability and chaotic behaviour are challenges that modern PinT methods need to be able to handle. Promisingly, a number of PinT methods have already been demonstrated to work with spatial parallelisation techniques (Christlieb et al., 2012; Gander et al., 2013a; Haynes and Ong, 2014; Samuel, 2012). With the advent of exascale HPCs on the horizon (Mann, 2020), increased attention is being paid to develop large-scale PinT methods[5] and we too will focus our efforts on furthering this goal. x

1.1.3 Our focus: Parareal

In this thesis, we focus on a particular PinT method known as the *Parareal* algorithm, an easy-to-use multiple shooting method first proposed by Lions et al. (2001). Since its inception, Parareal has become increasingly popular due to its relatively straightforward implementation and demonstrable effectiveness in providing IVP speedup for a range of problems spanning molecular (Baffico et al., 2002; Engblom, 2009; Legoll et al., 2020, 2022) and fluid dynamics (Fischer et al., 2005; Garrido et al., 2005; Trindade and Pereira, 2006), to geophysical processes (Clarke et al., 2020; Samuel, 2012) and nuclear physics (Baudron et al., 2014a,b; Grigori et al., 2021; Samaddar et al., 2010). Based on ideas from Nievergelt (1964) and Chartier and Philippe (1993), Parareal partitions the time domain into N slices, assigning a processor to each one, and solves the smaller IVPs in parallel using a computationally expensive, high accuracy, serial time-stepping method referred to as the *fine* solver—recall Figure 1.3. To locate the $N-1$ unknown initial values from which to begin integration and therefore avoid the pitfall created by the causality principle, Parareal uses a second serial time-stepping method (referred to as the *coarse* solver) that is computationally cheaper and of lower numerical accuracy compared to the fine solver. The algorithm iteratively locates a solution to the IVP by combining the coarse and fine solutions using a predictor-corrector (PC) scheme, derived by discretising the Newton-Raphson method—a more detailed derivation and exposition of Parareal is given in Chapter 2. The algorithm stops after $k \leqslant N$ iterations, once a pre-specified tolerance is met.

The defining metric of performance when using Parareal is the number of iterations k taken until the algorithm stops. The aim being to calculate a solution to the

[5]The *TIME-X* project is one particular example of an initiative that aims to showcase parallel-in-time methods that have been developed in academic settings on massively parallel HPC architectures, see https://www.time-x.eu/.

IVP that has numerical accuracy of the order of the high accuracy fine solver faster than the fully serial computation. In terms of runtime, each iteration of Parareal is approximately equal to a single (computationally expensive) run of the fine solver over a time slice, and so the smaller k is, the higher the parallel speedup realised by Parareal. Speedup is calculated by taking the ratio of the time taken for the fine solver to run sequentially over *all* time slices and the time taken for Parareal to run—roughly approximated by the ratio N/k. Therefore, if we can reduce the number of iterations by even just a few, we can dramatically reduce Parareal runtimes and increase parallel speedup if the cost of a fine solve is sufficiently expensive.

When Parareal uses a low accuracy coarse solver, it typically requires more iterations to converge and so there is an increasing demand for fast, but numerically accurate, coarse solvers for particular IVPs—we will revisit the importance of having a cheap coarse solver in Chapter 2. Whilst certainly a worthwhile goal, our aim is not to develop faster and more accurate coarse solvers but to instead try to make better use of the simulation data already available in Parareal, specifically the solution data generated by the coarse and fine solvers. The primary aim of this thesis is to investigate whether ideas from PN (next section) can make more efficient use of the Parareal simulation data to guide Parareal to the $N-1$ unknown solution states in fewer iterations than is currently possible. In addition, the hope is that the PN methods may also be useful in returning a measure of uncertainty over the Parareal solution, something that does not happen in Parareal and is typically calculated *a posteriori*. We chose to work with Parareal (rather than another PinT method) because the two solvers return distinct datasets that we can manipulate with our probabilistic techniques whereas other PinT methods do not seem to generate such datasets. In the next section, we describe what PN methods are, how they are used to solve differential equations, and how we can exploit them within the PinT framework.

1.2 Probabilistic numerics for differential equations

1.2.1 What are probabilistic numerical methods?

The purpose of many (classical) numerical algorithms is to locate approximate solutions to problems that typically have no closed-form solution. Consider for example, the sequential time-stepping methods mentioned in the previous section, which are designed to approximate the solution to an IVP on a discretised set of points in time (Hairer et al., 1993). Most time-stepping methods are derived in such a way that the error, with respect to the unknown exact solution at each time step, is bounded by a scalar multiple of the time step size to some power (i.e. the local/global truncation error). In practice, one can often choose a very small time step, solve the IVP and be done, postponing any error estimation indefinitely (often assuming the

error is negligible). This assumption has traditionally taken root among practitioners, however, if there exists a computational budget constraining how small the step sizes can be, it may not be feasible. In general, errors bounds from time-stepping schemes are assumed to be 'uniform', i.e. the probability that the exact solution lies within the bound is equally likely for any value in the bound. Therefore, if the IVP solution is to be used to initialise a future (non-probabilistic) computation, e.g. initial conditions for a different IVP, then the user needs to make a choice about which solution (within the scalar bound) to use—whose error may not have been accounted for, and therefore amplified, in the second computation.

PN provides a way to quantify and structure the uncertainty arising from numerical computations by formulating the problem under consideration within the framework of statistical (often, Bayesian) inference (Hennig et al., 2022). Initially, the unknown solution to the problem (e.g. the IVP) that we wish to *infer*, sometimes referred to as the *latent* quantity, is assigned a *prior* probability distribution. This distribution expresses a belief about the latent quantity and is often informed by any *a priori* knowledge about its structure or behaviour. "Data" arising from the calculations within a simulation, e.g. the vector field evaluations for an IVP, are treated as *observations* and are related to the latent quantity by a *likelihood function*. This likelihood prescribes a measure of uncertainty (with respect to the latent quantity) over the observations. The prior and likelihood can then be combined using Bayes' theorem, which may require complex (possibly analytically intractable) calculations, to compute a *posterior* distribution over the latent quantity, i.e. a probability distribution over the solution to the problem.

PN methods have many advantages over classical numerical methods. They return a (posterior) probability distribution over the solution, rather than the point-wise estimates (on a mesh) given by classical methods, providing an approximation of the solution (i.e. the "location" of the posterior) and its associated numerical error (i.e. the "width" of the posterior) if correctly calibrated[6]. The posterior provides a much richer description of the numerical uncertainty over the solution and samples can be drawn that each represent a realistic estimate of the true solution[7]. In fact, many classical numerical methods can be recovered as mean or maximum *a posteriori* estimates of the corresponding PN method. For example, the posterior mean of the PN ODE solvers proposed in Schober et al. (2014a) coincide exactly with solutions obtained by low-order Runge-Kutta methods. Two additional benefits of PN methods,

[6]Calibration of the numerical uncertainty refers to the "width", i.e. the variance/support, of the posterior distribution and it is important to question "...whether the posterior can indeed be endowed with an interpretation as a notion of uncertainty, connected to the probable error of the numerical method" (Hennig et al., 2022, Pg. 12). In other words, can we trust whether the uncertainty returned by the posterior distribution is a true representation of the error in the numerical method? Much work is ongoing to show that this is the case (Bosch et al., 2021).

[7]For IVPs, many PN methods can actually infer the solution at time steps that are not on the original mesh, something that classical methods cannot do without some form of interpolation scheme (Bosch et al., 2021; Krämer et al., 2022; Schober et al., 2019).

not possible with classic methods, are that known information about the solution can be encoded into the prior before simulation and that PN methods can handle probability distributions as both inputs (priors) and outputs (posteriors), affording them the ability to propagate uncertainty throughout a sequence of computational tasks.

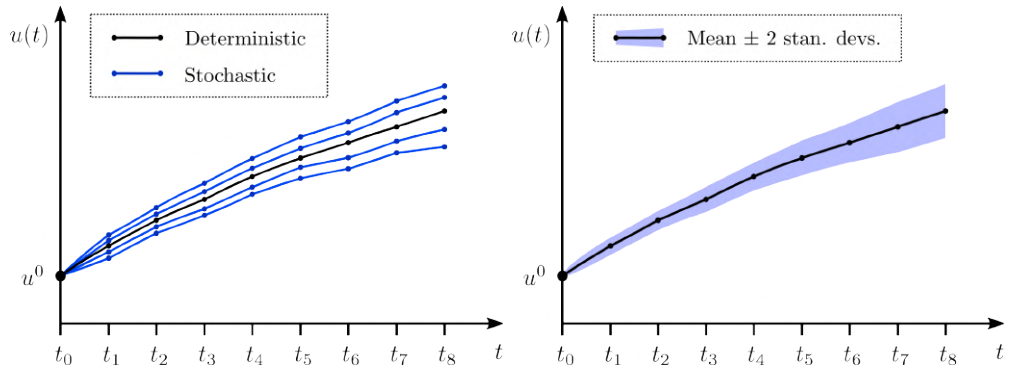
This is not to say PN methods have no drawbacks. As mentioned before, appropriate priors and likelihoods must be specified in order to yield calibrated posterior uncertainty estimates—a challenging task if unfamiliar with the problem at hand. In addition, calculating a posterior distribution is usually more costly than calculating a point-wise estimate of the solution. However, the benefits of having the calibrated uncertainty estimates mentioned above can be worth the additional cost if the total computational budget for a task is limited—some PN methods for differential equations are now reaching competitive cost in line with classical methods. We will revisit some of these points in the next section.

1.2.2 Our focus: sampling- and learning-based methods

PN methods play an an important role in solving numerical problems and quantifying epistemic uncertainty, i.e. the uncertainty due to a lack of information, in a number of different application areas and have deep-seeded roots in history (Hennig et al., 2015; Oates and Sullivan, 2019). With the aforementioned advances in computing technology over the last half century, there has been renewed interest in PN[8], with work being undertaken to strengthen its mathematical foundations (Cockayne et al., 2019) and showcase its capabilities in many different areas of classical numerical computation, including quadrature, linear algebra, optimisation, and most importantly for us, differential equations—a breakdown of these methods can be found in Hennig et al. (2022). There are two main approaches to solving IVPs using PN methods which we will refer to as *sampling-* and *learning-based* methods.

Sampling-based methods solve IVPs by using random perturbations within classical (deterministic) one-step numerical integrators, e.g. forward/backward Euler schemes. One of the first such methods was proposed by Conrad et al. (2017), in which they perturbed solutions (at each time step) from a deterministic integrator using a sample from an appropriately scaled Gaussian distribution. These samples were assumed to represent the numerical error generated by the deterministic integrator and so convergence results were derived showing that stochastic numerical solutions were of the same accuracy as those obtained deterministically—see Figure 1.4(a) for an illustration comparing the deterministic and stochastic solutions. Therefore, by solving the ODE multiple times (in an embarrassingly parallel manner) one can

[8]There exists a strong community dedicated to PN research with extensive work going into developing ProbNum (https://probnum.readthedocs.io/en/latest/), a Python package for carrying out PN computations.



(a) Sampling-based **(b)** Learning-based

Figure 1.4: Illustrative solutions to a scalar ODE, located using PN solvers (on one processor). (a) Deterministic time-stepping scheme solution (black) vs. stochastic solutions (blue) generated by a sampling-based solver that perturbs solutions from a deterministic time-stepping scheme. Simulating sufficiently many independent stochastic solutions using the the sampling-based solver forms a distribution over the solution to the ODE. (b) Posterior distribution over the ODE solution generated by a learning-based solver. The distribution is depicted by its mean (black) and two standard deviations either side of it (light purple regions).

obtain a (non-Gaussian) distribution of solutions that represent all[9] possible ODE trajectories obtainable with the deterministic integrator—useful for solutions which exhibit chaotic behaviour or exist near invariant manifolds in phase space. In follow up work, Lie et al. (2019, 2022) derived stronger convergence results, allowing for more general (non-Gaussian) perturbations, and in Abdulle and Garegnani (2020) the authors instead perturb the time step to preserve geometric features of the deterministic integrator.

On the other hand, learning-based methods formulate the IVP from a Bayesian point of view. Work on these methods began with Skilling (1992), where he developed the first "ODE filter", a solver that uses Gaussian process (GP) regression (O'Hagan, 1978; Rasmussen, 2004) to calculate direct probabilistic solutions to the IVP. GP regression models are statistical models that can infer the value of an unknown (expensive-to-simulate) function using multivariate Gaussian distributions and a finite number of evaluations of the function—an explanation of GPs will be given in Chapter 5. ODE filters solve sequentially in time, conditioning the GP prior distribution on solution and derivative evaluations (observations), to obtain a posterior over the ODE solution (Hennig and Hauberg, 2014; Schober et al., 2014a,b). Figure 1.4(b) illustrates what a posterior distribution over the solution to an ODE looks like. The inefficiency of standard GP regression methods lead to the devel-

[9]In reality, not all possible trajectories can be realised as this would take an infinite number of independent simulations. A well-approximated distribution of trajectories can be obtained with a reasonable number of independent simulations.

opment of more advanced ODE filters that instead use Gauss-Markov processes (Øksendal, 2013) and can harness the computational speed of Kalman filters (Särkkä, 2013) to calculate the posterior distribution using observations sequentially. Instead of calculating a numerical solution on the mesh, as classical integration methods do, these ODE filters can also return a probability measure over the solution at times off the mesh (Bosch et al., 2021; Schober et al., 2019; Tronarp et al., 2019; Wenger et al., 2021) and are becoming computationally competitive with respect to classical solvers (Kersting et al., 2020; Krämer et al., 2022). At any point in time, one can take a 'slice' (vertically) and obtain a distribution over the solution to the ODE. This direct method of calculating a posterior distribution differs from the sampling-based approach in which the ODE has to be solved numerous times to obtain a distribution. Sampling- and learning-based methods, however, are not necessarily mutually exclusive—PN ODE solvers have been developed that use ideas from both (Chkrebtii et al., 2016; Tronarp et al., 2019).

As previously hinted at, the reason we discuss these two approaches is that Parareal generates a lot of simulation data, i.e. fine and coarse solution information, most of which (after being used once) is "forgotten" by Parareal in a Markovian-like manner. By this, we mean that Parareal uses the solution data only once in the PC update (the coarse data is used twice) before generating new simulation data for the next update. We wish to take advantage of the valuable information that this data gives us about how the fine and coarse solvers propagate a solution state from one time step to the next. Using the sampling-based approach, we will use the simulation data to construct probability distributions to model where we believe the true $N-1$ unknown initial values are. From these distributions we can sample candidate solutions and explore the solution space much more freely than is possible with just Parareal and try to reduce the number of iterations taken until convergence—this work will form the bulk of Chapters 3 and 4. With regard to the learning-based approach, we plan to use the simulation data as observations in a GP emulator[10] to try to infer solution states from the solvers without having to run them (at expensive cost). In effect we will attempt to "learn" how the solvers behave and then try to infer the $N-1$ solution states faster than Parareal can—this work will be presented in Chapters 5 and 6. In the next section, we will explicitly state our aims, describe how we plan to achieve them by combining PinT methods with some of the PN techniques discussed, and outline the remainder of the thesis.

[10]GP emulators work in the same way as GP regression models except that the observations are typically outputs of computer-based simulations (in our case, the fine and coarse solvers) and have zero observation error, i.e. they are "noise-free".

1.3 Thesis aims and outline

Until now, PinT methods have been derived from the classical numerical analysis viewpoint, yielding deterministic solutions and fixed rates of speedup when solving a given IVP. This works well in many situations, however, when speedup for a particular type of IVP is limited, these methods do not make full use of the solution data they generate to reduce runtimes—something mentioned in Maday and Mula (2020) in the context of domain decomposition and high-order time-stepping but yet to be investigated in much detail. In a similar vein, PN methods have been designed to solve IVPs in a purely sequential manner, suffering the same computational intractability issues as sequential time-stepping methods when solving IVPs with long time intervals—it has been hinted that parallelising these methods is a next natural next step in their development (Kersting and Hennig, 2016).

We wish to fuse these ideas together to accelerate the convergence of Parareal by making more efficient use of the fine and coarse solution information generated throughout a simulation. The PN framework for solving IVPs is built in a way that naturally allows us to use the simulation data from Parareal and perhaps even re-use it in future simulations—something that existing PinT methods do not do. The long-term goal is to develop a fully probabilistic PinT method that can solve an IVP in parallel and return a posterior distribution with a meaningful measure of uncertainty over the IVP solution.

In this thesis, we aim to:

I. derive, analyse, and test the first probabilistic PinT algorithms by incorporating some of the sampling- and learning-based techniques from PN into the Parareal algorithm.

II. demonstrate that our algorithms yield additional numerical speedup vs. the standard Parareal algorithm when solving nonlinear IVPs on HPC facilities.

III. show that solutions obtained from our algorithms are accurate by verifying convergence both numerically and analytically.

IV. if possible, generate probabilistic solutions to the IVPs (like the aforementioned PN methods do).

V. assess the long-term viability of our algorithms to determine whether they can in fact provide numerical speedup for large-scale IVPs.

How we achieve these aims and structure the remainder of the thesis is as follows.

In Chapter 2, we begin by setting up the IVP, a time-dependent system of (potentially nonlinear) ODEs that are computationally intractable to solve over a long time interval using serial methods. We then discuss what kind of ODE systems

fall under this category and which serial time-stepping methods are available to us. Following this, we derive the Parareal algorithm from first principles using the work of Gander and Vandewalle (2007). We explain how it works in practice (along with a numerical experiment), analyse its computational complexity, review recent work on error bound analysis, and explore possible options for choosing the fine and coarse solvers. Furthermore, we describe a number of Parareal variants that have emerged over the years, discussing which issues (with respect to the classic Parareal scheme) they attempt to remedy and how these approaches vastly differ from what we are trying to do.

In Chapter 3, we present our first algorithm, a sampling-based PinT method we refer to as *Stochastic Parareal* or *SParareal* for short (Pentland et al., 2023a). Instead of integrating forward in time from a single value (in each time slice) given by Parareal's PC, we use a pre-specified probability distribution to sample and propagate a number of candidate initial values forward in time in parallel. The idea is that, using distributions constructed from known coarse and fine solution information, this stochastically generated set of initial values explores the solution space more efficiently and provides a better guess to the solution (the $N-1$ unknown initial values) than those found purely deterministically, accelerating convergence compared to the classic Parareal algorithm. We begin by examining the sampling-based PN techniques we propose to use and provide some background for how we came up with the scheme itself. As with Parareal, we then detail how SParareal works (now generating stochastic, not deterministic, solutions), analyse its computational complexity, and present numerical experiments. We demonstrate that, for low-dimensional ODE systems, SParareal converges in fewer iterations than Parareal when the number of samples is large enough and can return a measure of uncertainty over the solution upon multiple simulations of the algorithm.

All numerical experiments in this thesis are conducted in MATLAB with parallel simulations run on HPC facilities at the University of Warwick. In particular, we make use of the HPC system known as *Avon*[11], allowing us to run experiments on up to a maximum of 512 compute cores (using more cores with MATLAB was not possible due to software limitations). In the past few decades, simulation methods have moved to the forefront of scientific research and increasing pressure is being placed on researchers to uphold standards on computational reproducibility. In light of this (and to enable further scrutiny of our work), all code written to generate the computational results in this thesis is available in a public repository[12]. This should allow interested readers to reproduce the numerical experiments and, where

[11] *Avon* is made up of 180 *Dell PowerEdge C6420* compute nodes each equipped with two *Intel Xeon Platinum 8268 (Cascade Lake)* 2.9 GHz 24-core processors (i.e. 48 cores $\times$180 nodes = 8640 total) each with 192 GB *DDR4-2933* RAM. For more information, visit https://warwick.ac.uk/research/rtp/sc/.

[12] Please visit https://tinyurl.com/KP-PhD-Thesis (or see Pentland (2023) for full URL). If there are any issues accessing this repository, please do get in contact.

simulations may take some time to run on HPC facilities, regenerate the same results using stored data files.

In Chapter 4, we derive rigorous error bounds for SParareal, demonstrating that the stochastic solutions converge in the mean-square sense (Pentland et al., 2023b). In particular, we derive both superlinear and linear mean-square errors bounds when using SParareal with different types of random perturbation, e.g. perturbations that may or may not depend on the current state of the system at which they are generated. In order to do this, we formulate SParareal in a slightly different way to the version presented in Chapter 3, however, the analysis is still valid in the case where we take single random sample at each time slice (this will become clear later on). We perform a number of numerical experiments, validating our theoretical results by showing they align with the numerical errors generated by simulations of SParareal.

In Chapter 5, we approach the PinT problem from a more Bayesian (rather than frequentist) perspective, addressing some of the issues associated with SParareal. We present our second algorithm, a learning-based PinT method we refer to as *GParareal* (Pentland et al., 2023c). To find the $N-1$ unknown initial values in fewer iterations, we model the correction term in Parareal's PC with a GP emulator, trained on all coarse and fine solution information obtained throughout the simulation (the amount of which increases with each iteration). The idea is to locate more accurate corrections and avoid throwing away valuable solution data. As before, we provide an overview of how the scheme was conceptualised and discuss some of the background material associated with how other learning-based methods have been used in PN and Parareal. We then derive GParareal, analyse computational complexity, provide an error bound on the solution at a fixed iteration (that depends on how well the emulator is trained), and carry out extensive numerical experiments to show that GParareal can converge faster than Parareal for a number of low-dimensional nonlinear ODE systems. In addition, we demonstrate how GParareal can re-use simulation data from a previous simulation (legacy data) of the same IVP, whether it be from solving from a different initial condition or over a different time interval, to converge in even fewer iterations than Parareal. We also show that GParareal can converge for IVPs for which Parareal cannot, i.e. when the coarse solver is too coarse for standard Parareal to converge. We note in the complexity analysis and corresponding numerical experiments that training the GP emulator comes at an additional cost compared to Parareal and explore options to work around this.

In Chapter 6, we push GParareal to its limits by investigating how it performs when used to solve the one-dimensional viscous Burgers' equation and the two-dimensional FitzHugh–Nagumo system. We aim to analyse how much the GP emulation process impacts the realisable speedup that can be obtained from GParareal by running experiments with different types of legacy data (more on this later) and

an increasing number of spatial discretisation points (which increases the size of the ODE system to be solved). These results help us pinpoint where algorithm performance is hindered and enable us to suggest a few ways to optimise GPararealʼs implementation.

We conclude in Chapter 7 with a short discussion on the significance and repercussions of the algorithms developed in this thesis, collectively assessing the quality of our results with respect to our original aims. We then finish by setting the stage for future work in the area of probabilistic PinT algorithms.

Chapter 2

The Parareal algorithm

Overview

The purpose of this chapter is to lay the foundations for deriving, analysing, and testing our probabilistic PinT algorithms. We begin by stating the general form of the IVP to be solved repeatedly throughout this thesis and highlight the types of IVP that fall within this category. The objective is to find a numerical solution to such IVPs on a discrete temporal mesh. This solution, however, is assumed to be computationally intractable to calculate in real time using (expensive) sequential time-stepping schemes. We will outline a few classic time-stepping methods that we make use of, justifying why they have been selected over more advanced methods. This will set the stage for calculating a solution to the IVP in parallel using the Parareal algorithm.

Next, we show how Parareal can be derived from a multiple-shooting perspective and explain how it works in practice, making use of not just one, but two sequential time-stepping schemes. We outline its computational complexity, how it (typically) schedules computational tasks in the simulation, and how it stops once a solution is found—all aspects of which will be important when comparing to our algorithms in later chapters. We state and prove an existing error bound on the Parareal solution with respect to the fine solver solution, the assumptions and proof techniques of which we will use to derive bounds for our own algorithms. A short discussion on how to chose the two sequential time-stepping schemes will precede a numerical experiment in which we demonstrate how Parareal runs in practice. To conclude, we discuss variants of Parareal that have emerged since its inception that try to improve certain aspects of its performance, both in general and for specific types of IVP. This will provide a starting point for discussing related works that have attempted to use sampling- or learning-based methods to improve Parareal—more details of which will be provided in Chapters 3 and 5, respectively.

2.1 Initial value problem setup

Throughout this thesis, we will be concerned with solving systems of $d \in \mathbb{N}$ ODEs of the form

$$\frac{\mathrm{d}\boldsymbol{u}}{\mathrm{d}t} = \boldsymbol{f}\big(t, \boldsymbol{u}(t)\big) \quad \text{over} \quad t \in [t_0, T], \quad \text{with} \quad \boldsymbol{u}(t_0) = \boldsymbol{u}^0 \in \mathcal{U} \subset \mathbb{R}^d, \tag{2.1}$$

where $\boldsymbol{f} \colon [t_0, T] \times \mathcal{U} \to \mathbb{R}^d$ is a (potentially nonlinear) vector field, $\boldsymbol{u} \colon [t_0, T] \to \mathcal{U}$ is the time-dependent solution, and $\boldsymbol{u}^0$ is the initial value at time t_0. We will assume that $\boldsymbol{f}$ is sufficiently smooth such that the IVP (2.1) has a unique solution for all initial conditions of interest and that $[t_0, T] \subset \mathbb{R}$ such that $T < \infty$. We seek numerical solutions $\boldsymbol{U}_n \approx \boldsymbol{u}(t_n)$ to (2.1) on a pre-defined mesh $\boldsymbol{t} = (t_0, \ldots, t_N)$, where $t_{n+1} = t_n + \Delta T$ for fixed $\Delta T = (T - t_0)/N$. In the forthcoming sections, each $[t_n, t_{n+1}]$ for $n = 0, \ldots, N-1$ will be referred to as a "time slice", where N is the total number of time slices.

IVPs of the form (2.1) occur frequently when modelling physical processes throughout nature. They arise in diverse applications areas ranging from mathematical epidemiology (Murray, 2002) and neuroscience (Brzychczy and Poznanski, 2013) to fluid and orbital mechanics (Danby, 1997). In many applications, such as the simulation of plasma dynamics, spatio-temporal PDEs need to be solved and are usually spatially discretised via a *method of lines*, e.g. finite differences or spectral methods, among others (Trefethen, 2000). This discretisation process results in a large system of ODEs such as (2.1), where d scales with the number of spatial locations the PDEs are being resolved at.

In any case, we are concerned with IVPs where one or more of the following aspects:

(i) the interval of integration, $[t_0, T]$,

(ii) the number of mesh points, $N + 1$,

(iii) or the wallclock time (in seconds) to evaluate the vector field, $\boldsymbol{f}$,

is so large that numerical solutions $\boldsymbol{U}_n$ take hours, days, or even weeks to obtain using classical sequential time-stepping schemes (Butcher, 2016; Hairer et al., 1993). Henceforth, the term *time-stepping scheme* may be used interchangeably with the terms *numerical integrator*, *solver*, *method* or *flow map*—more on these in the next section.

Many different types of IVP fall into a subset of the three categories above, with the majority facing issues related to a combination of (i) and (ii) (as they are often intrinsically linked). The number of mesh points $N+1$ required to integrate over very long time intervals $[t_0, T]$ typically needs to be sufficiently large (i.e. ΔT needs to be sufficiently small) such that stability conditions for the chosen solver are satisfied.

For very long time intervals, ΔT may need to be orders of magnitude smaller than the length of $[t_0, T]$, drastically increasing the number of computations required and making the calculation infeasible. For example, systems where the dynamics are *stiff*, i.e. systems in which the solution varies very rapidly at certain times but very slowly at other times, tend to require a very high number of mesh points to resolve. This is because the solver becomes numerically unstable unless ΔT is very small in the stiff intervals, leading to an (unnecessarily) slow computation when integrating in the non-stiff intervals. This occurs in many applications, e.g. fluid mechanics and chemical reaction modelling, where dynamics evolve on both slow and fast time scales, often orders of magnitude apart. While there do exist solvers that can adaptively increase/decrease the size of the time step to deal with stiff dynamics, they still suffer from computational intractability when $[t_0, T]$ is large enough (relative to the fast timescale).

A particularly challenging application area for numerical time-stepping methods is simulating magnetically confined fusion plasmas (i.e. ionised gases) in tokamak devices (Goldston, 1995). Designing a working thermonuclear fusion reactor that can control plasmas at extreme temperatures and densities is the key to generating a clean and sustainable global energy source (CCFE, 2023). Simulating IVPs that describe plasma behaviour provides us with a way to avoid the costly and difficult task of running physical experiments in a tokamak, however, they are notoriously time-consuming to simulate. One aim is to understand the evolution of temperature and particle densities on a macroscopic time-scale (i.e. on the order of seconds/minutes), as they are the primary quantities of interest governing reactor performance. However, the highly nonlinear transport processes (e.g. heat, particle, and momentum fluxes) which drive macroscopic behaviour are, in turn, governed by turbulent processes which evolve on a microscopic time-scale (i.e. microseconds). Simulating plasma IVPs over a number of seconds (at high spatial resolution) is therefore an extremely costly multi-scale problem (in both time and space), making plasma simulation a perfect candidate for PinT algorithms.

Many other motivating examples for long time integration can be found in systems containing an element of randomness, e.g. SDEs. Although we are solving deterministic ODEs here, the problems caused by (i), (ii), and (iii) extend to SDEs because the computational time to solve them still becomes infeasible when using sequential time-stepping schemes for SDEs. Molecular dynamics is one such example, in which ensemble averages of trajectories, i.e. solutions to the IVP, are required to observe dynamics that evolve (or may not even appear) over very long time scales (Legoll et al., 2022). Such dynamical systems may also exhibit *metastability*, in which trajectories spend a long time in certain regions of phase space before transitioning to another (Grafke et al., 2017). Simulations of these long-time trajectories require small time steps (limited by the stability of the solver), too many of which lead to

computational intractability.

Other dynamical systems (which do not necessarily contain randomness) can exhibit *chaos*, in which trajectories with initial conditions close to one another diverge exponentially over time. Simulations of these trajectories require small time steps, not to maintain stability of the solver, but to minimise the divergence of the numerical solution from the true solution when errors begin to accumulate over time. With regards to the final point (iii), if a spatio-temporal PDE contains particularly difficult terms involving nonlinearities, integrals, or high-order derivatives, the cost of evaluating the vector field $\boldsymbol{f}$ can be very high. Reducing the number of calls to the vector field $\boldsymbol{f}$ can become very significant if it is expensive to evaluate, as many (high-order) numerical methods require large numbers of calls to $\boldsymbol{f}$.

2.1.1 The objective

Regardless of the type of IVP being solved, we can write the solution to (2.1) at time t_{n+1} in integral form

$$\boldsymbol{u}(t_{n+1}) = \boldsymbol{u}(t_n) + \int_{t_n}^{t_{n+1}} \boldsymbol{f}\big(s, \boldsymbol{u}(s)\big) ds, \quad n = 0, \dots, N-1, \tag{2.2}$$

where the integral is unknown because it depends on the solution $\boldsymbol{u}$. Different time-stepping schemes use different methods to approximate the integral in (2.2). The simplest example being the forward Euler method, i.e. approximating the integral by $\Delta T \boldsymbol{f}(t_n, \boldsymbol{u}(t_n))$. Suppose we have access to a computationally expensive serial time-stepping scheme, known henceforth as the *fine solver* $\mathcal{F}_{\Delta T} \colon \mathcal{U} \to \mathbb{R}^d$, which propagates an initial value $\boldsymbol{U}_n$ at time t_n over an interval of length ΔT, returning a terminal state $\boldsymbol{U}_{n+1}$ with high numerical accuracy at time t_{n+1}[1]. For the time being, we will assume that $\mathcal{F}_{\Delta T}$ provides sufficient numerical accuracy to the user such that the solution to (2.1) calculated by $\mathcal{F}_{\Delta T}$ can be considered 'exact', i.e. $\boldsymbol{U}_n = \boldsymbol{u}(t_n)$. The objective is to calculate the exact solutions

$$\boldsymbol{U}_{n+1} = \mathcal{F}_{\Delta T}(\boldsymbol{U}_n), \quad \text{for} \quad n = 0, \dots, N-1, \tag{2.3}$$

where $\boldsymbol{U}_0 := \boldsymbol{u}^0$, *without* running $\mathcal{F}_{\Delta T}$ N times sequentially, as this calculation is assumed to be computationally intractable. For perspective, one should expect that running $\mathcal{F}_{\Delta T}$ over a single time slice, or equivalently calculating (2.3) for a single n, will take on the order of minutes or hours, thereby making the computation of (2.3) for all n infeasible in real time. Next, we discuss some possible options for choosing $\mathcal{F}_{\Delta T}$.

[1] We should mention that if (2.1) is nonautonomous, then $\mathcal{F}_{\Delta T}$ should depend explicitly on t_n (and possibly t_{n+1}), however, the extra notation is burdensome and so we drop it. Note, however, that all of the algorithms described in this thesis work for nonautonomous IVPs.

Sequential time-stepping methods

Numerical schemes for integrating IVPs, i.e. approximating the right hand side of (2.2), have been extensively developed and analysed over the past century (Butcher, 2016; Hairer et al., 1993). They are typically categorised into two classes: single- and multi-step methods. Our focus will be on single-step methods (such as $\mathcal{F}_{\Delta T}$), that take a single initial state and propagate it forward in time (under the action of the differential equation), returning a terminal state[2]. In particular, we will let $\mathcal{F}_{\Delta T}$ be a single-step method that is allowed to take multiple (smaller, $\delta t \leqslant \Delta T$) single-steps in order to calculate the terminal state—it will become clear as to how this works when we detail the Parareal algorithm in the next section (see Figure 2.1).

One-step methods take many forms and some have properties that make them more suited to solving particular types of IVPs than others. Conceptually, the simplest to understand and implement are pth-order *explicit Runge-Kutta* methods (denoted RKp), which return a solution state at t_{n+1} which depends only on the state of the system at time t_n (Kutta, 1901; Runge, 1895). By pth-order, we mean that the method accumulates local truncation error $\mathcal{O}(\Delta T^{p+1})$ after each step forward in time. An explicit s-stage RKp method[3] is written

$$\boldsymbol{U}_{n+1} = \boldsymbol{U}_n + \Delta T \sum_{i=1}^{s} b_i \boldsymbol{k}_i \quad \text{for} \quad n = 0, \dots, N-1, \tag{2.4}$$

where

$$\boldsymbol{k}_i = \boldsymbol{f}\left(t_n + c_i \Delta T, \boldsymbol{U}_n + \Delta T \sum_{j=1}^{i-1} a_{ij} \boldsymbol{k}_j\right). \tag{2.5}$$

The term $A = (a_{ij})_{i,j=1,\dots,s}$ is known as the RK matrix, $\boldsymbol{b} = (b_i)_{i=1,\dots,s}$ is the weight vector, and $\boldsymbol{c} = (c_i)_{i=1,\dots,s}$ is the vector of nodes (all of which can be collated in a Butcher tableau). For example, the *forward Euler* (or explicit RK1) scheme, which has local truncation error $\mathcal{O}(\Delta T^2)$, is a one-stage method given by

$$\boldsymbol{U}_{n+1} = \boldsymbol{U}_n + \Delta T \boldsymbol{f}\big(t_n, \boldsymbol{U}_n\big) \quad \text{for} \quad n = 0, \dots, N-1.$$

Explicit RK methods tend to be the simplest to implement, requiring little work to calculate each step, however, they typically require very small time steps to remain

[2]Multi-step methods are very powerful tools that use multiple initial states between $[t_n, t_{n+1}]$ to calculate the terminal state at t_{n+1}. Whilst they can be used to solve (2.1), they are not compatible with the classic formulation of Parareal (see Section 2.2) without some form of adaptation (Ait-Ameur et al., 2020, 2021). In addition, they do not work with our probabilistic PinT algorithms (yet!) and so we do not consider them here.

[3]The number of stages s in an explicit RKp method refers to the number of intermediate evaluations of the vector field $\boldsymbol{f}$ (between $[t_n, t_{n+1}]$) that are required in order to estimate the solution at t_{n+1}. It has been proven that $s \geqslant p$ (Butcher, 2016, Theorem 324A) and so we omit the s-stage notation when referring to an RKp method.

numerically stable—particularly problematic for stiff systems.

On the other hand, we have pth-order *implicit Runge-Kutta* methods (denoted implicit RKp), which return a solution state at time t_{n+1} which depends on *both* the current and future state of the system. $\boldsymbol{U}_{n+1}$ is calculated the same way as in (2.4), except that the summation inside the $\boldsymbol{k}_i$ term (2.5) ranges over $j = 1, \dots, s$, meaning that both the left and right hand sides of (2.4) depend on the state $\boldsymbol{U}_{n+1}$. To see this more clearly, consider the *backward Euler* (or implicit RK1) method

$$\boldsymbol{U}_{n+1} = \boldsymbol{U}_n + \Delta T \boldsymbol{f}\big(t_n + \Delta T, \boldsymbol{U}_{n+1}\big) \quad \text{for} \quad n = 0, \dots, N-1.$$

To calculate $\boldsymbol{U}_{n+1}$, a (nonlinear) system of equations needs to be solved, making implementation of implicit methods slightly more involved and costly than an explicit method. The method does, however, remain numerically stable when taking much larger time steps, meaning that fewer overall time steps are needed to solve over the entire time interval. High order numerical accuracy is of course desirable when using either explicit or implicit methods, however, as the accuracy of the method increases the computational cost per time step (typically) increases too.

Besides RK methods, there are a wide variety of highly specialised solvers available for tackling particular types of IVP. Examples include adaptive RK methods that alter step sizes to meet a pre-defined error tolerance (useful for stiff systems) and geometric/symplectic methods that preserve certain features of the exact flow prescribed by the vector field (e.g. energy). In this thesis, however, we will use our own purpose-built pth-order explicit and implicit RK methods to carry out numerical experiments on Parareal, SParareal, and GParareal. More often than not, ODE solvers provided in packages and built-in software are highly optimised for speed and performance, making use of adaptive time-stepping, stiffness detection, and error control that we avoid as not to interfere with our experiments. In particular, we wish to be able to control the number of time steps in each time slice, knowing exactly how long each slice takes to run in real time (which increases linearly with the number of time steps in the case of explicit RK methods).

Whilst more complex methods such as adaptive solvers may be more optimal for integrating certain (e.g. stiff) IVPs, they can result in *load imbalance* when deployed in PinT algorithms. Load imbalance arises when some processors finish their tasks faster than others, having to wait idly before being allowed to execute the next task and therefore leading to speedup degradation. When using adaptive solvers in Parareal, for example, integration in some time slices may be faster than in others (if the dynamics are stiff in some but not others) and so they can interfere with wallclock time and speedup estimates without proper parallel scheduling workflows (which we do not focus on). To more clearly contrast and compare our probabilistic PinT algorithms with Parareal we postpone implementation of more advanced solvers to a

future work[4].

In the next section, we will explain how Parareal iteratively locates approximations $\boldsymbol{U}_n^k$ to $\boldsymbol{U}_n$ (where $k = 0, 1, 2, \ldots$ is the iteration number) using *two* such sequential time-stepping schemes, one fine- and one coarse-grained.

2.2 The algorithm

2.2.1 Derivation

Parareal is a deterministic multiple shooting algorithm[5], first proposed by Lions et al. (2001), for numerically integrating IVPs such as (2.1) in a time-parallel manner. To derive the algorithm from first principles, we will follow the steps outlined in the work of Gander and Vandewalle (2007). Recall that the goal is to calculate the solution states $\boldsymbol{U}_n$, that would typically be obtained by applying the fine solver serially over $[t_0, T]$ (i.e. calculating (2.3) sequentially), in parallel. To do this, the IVP (2.1) is partitioned into N smaller IVPs

$$\frac{\mathrm{d}\boldsymbol{u}_n}{\mathrm{d}t} = \boldsymbol{f}\big(t, \boldsymbol{u}_n(t \mid \boldsymbol{U}_n)\big) \quad \text{on} \quad t \in [t_n, t_{n+1}], \quad \text{with} \quad \boldsymbol{u}_n(t_n) = \boldsymbol{U}_n, \tag{2.6}$$

for $n = 0, \ldots, N-1$, that can (theoretically) be solved in parallel. Each time slice $[t_n, t_{n+1}]$ in (2.6) is then assigned its own processor, denoted $P_1, \ldots, P_N$ (an illustration of this assignment was given in Figure 1.3). We denote $\boldsymbol{u}_n(t \mid \boldsymbol{U}_n)$ to be the solution over $[t_n, t_{n+1}]$ *given* the initial value $\boldsymbol{U}_n$ at $t = t_n$ (this dependence on the initial values will become clear shortly). Note, however, that only the initial value $\boldsymbol{U}_0 = \boldsymbol{u}^0$ is known, whereas the rest ($\boldsymbol{U}_n$ for $n \geqslant 1$) need to be determined before (2.6) can be solved in parallel. These initial values must satisfy continuity conditions at the time slices boundaries, i.e.

$$\boldsymbol{U}_0 = \boldsymbol{u}^0 \quad \text{and} \quad \boldsymbol{U}_n = \boldsymbol{u}_{n-1}(t_n \mid \boldsymbol{U}_{n-1}) \quad \text{for} \quad n = 1, \ldots, N. \tag{2.7}$$

This (nonlinear) system of $N+1$ equations ensures the solutions $\boldsymbol{u}_n(t)$ match at each t_n. Chartier and Philippe (1993) can be credited with suggesting that (2.7) can be solved for the unknown $\boldsymbol{U}_n$ using the Newton-Raphson method, forming the iterative system

$$\boldsymbol{U}_0^{k+1} = \boldsymbol{u}^0, \tag{2.8a}$$

$$\boldsymbol{U}_{n+1}^{k+1} = \boldsymbol{u}_n(t_{n+1} \mid \boldsymbol{U}_n^k) + \frac{\partial \boldsymbol{u}_n}{\partial \boldsymbol{U}_n}(t_{n+1} \mid \boldsymbol{U}_n^k)\big(\boldsymbol{U}_n^{k+1} - \boldsymbol{U}_n^k\big), \tag{2.8b}$$

[4]By doing this we try our best to avoid "fooling the masses" when presenting our theoretical and numerical parallel speedup results in later chapters (Götschel et al., 2021). We assume that if our algorithms work for the most basic of time-stepping schemes then they should work (potentially with some adaptation) for more complex schemes.

[5]It was shown by Gander and Vandewalle (2007) that Parareal is also a multigrid method.

Figure 2.1: Schematic of the Parareal time domain decomposition. Three levels of temporal discretisation are shown: time slices (size ΔT), coarse intervals (size δT), and fine intervals (size δt). Note how the discretisations must align with one another in such a way that $\delta t \leqslant \delta T \leqslant \Delta T$ and $(N_{\mathcal{F}}/N)\delta t = (N_{\mathcal{G}}/N)\delta T = \Delta T$, where $N_{\mathcal{G}}$ and $N_{\mathcal{F}}$ are the total number of coarse and fine steps in $[t_0, T]$.

for $n = 0, \ldots, N-1$, and iteration number $k = 0, 1, 2, \ldots$. This system contains the unknown solutions $\boldsymbol{u}_n$ and their partial derivatives, which even if known, would be computationally expensive to compute[6]. Crucially, however, the parallel shooting method developed by Chartier and Philippe (1993) was limited to solving dissipative ODEs, i.e. (2.1) with a right hand side whose largest eigenvalue is negative. The invention of Parareal bridged the gap to solve problems with more general right hand sides.

To solve (2.8) and iteratively calculate the values $\boldsymbol{U}_n^k$ without the partial derivatives, Parareal utilises two (deterministic) serial numerical integrators. These solvers take an initial state $\boldsymbol{U}_n^k$ at time t_n and propagate it, over a time slice of size ΔT, to a terminal state $\boldsymbol{U}_{n+1}^k$ at time t_{n+1}. The *fine solver*, as described in Section 2.1.1, is denoted by $\mathcal{F}_{\Delta T} \colon \mathcal{U} \to \mathbb{R}^d$. It returns a terminal solution state at very high numerical accuracy, at very high numerical cost, and is allowed to take many small intermediate steps $\delta t \leqslant \Delta T$ to do this. The *coarse solver*, denoted similarly by $\mathcal{G}_{\Delta T} \colon \mathcal{U} \to \mathbb{R}^d$, returns a terminal state with much lower numerical accuracy and at much lower computational cost than $\mathcal{F}_{\Delta T}$, using (typically) larger intermediate time steps $\delta t \leqslant \delta T \leqslant \Delta T$ (see Figure 2.1). The essential condition is that $\mathcal{G}_{\Delta T}$ must be much cheaper to run than $\mathcal{F}_{\Delta T}$, i.e. $\mathcal{G}_{\Delta T}$ must be able to run serially across multiple time slices to provide relatively cheap low accuracy states whilst the slower $\mathcal{F}_{\Delta T}$ solver is only permitted to be run in parallel over multiple time slices. This is a strict requirement for Parareal, or else numerical speedup will not be realised (more on this in Section 2.2.3).

Returning to (2.8), we can see that (2.8a) is known *a priori* for all k. To calculate (2.8b), Lions et al. (2001) proposed approximating the first term in (2.8b) using the fine solver $\mathcal{F}_{\Delta T}(\boldsymbol{U}_n^k)$ and the second term by a coarse finite difference of the derivative $\mathcal{G}_{\Delta T}(\boldsymbol{U}_n^{k+1}) - \mathcal{G}_{\Delta T}(\boldsymbol{U}_n^k)$ [7]. Using the coarse approximation enables the

[6] In fact, Bellen and Zennaro (1989) first suggested solving (2.7) using Steffensen's method, a variant of the Newton-Raphson method that does not require exact calculations of the partial derivatives. The convergence of their algorithm was highly dependent on the accuracy of the initial guess to the solution, i.e. $\{\boldsymbol{U}_0^0, \ldots, \boldsymbol{U}_N^0\}$, which is difficult to know *a priori* to simulation. As we shall see shortly, Parareal provides a reliable method for calculating this initial guess (see (2.9b)).

[7] One could instead approximate the derivative by a fine finite difference $\mathcal{F}_{\Delta T}(\boldsymbol{U}_n^{k+1}) - \mathcal{F}_{\Delta T}(\boldsymbol{U}_n^k)$, however, (2.8b) simply becomes the sequential calculation in (2.3) we are trying to avoid!

fine computations in (2.8b) to be calculated in parallel, giving rise to the Parareal algorithm.

Definition 2.1 (Parareal)**.** For the two numerical flow maps $\mathcal{F}_{\Delta T}$ and $\mathcal{G}_{\Delta T}$ described above, the Parareal scheme is given by

$$\boldsymbol{U}_0^0 = \boldsymbol{u}^0, \tag{2.9a}$$

$$\boldsymbol{U}_{n+1}^0 = \mathcal{G}_{\Delta T}(\boldsymbol{U}_n^0), \qquad 0 \leqslant n \leqslant N-1, \tag{2.9b}$$

$$\boldsymbol{U}_{n+1}^{k+1} = \underbrace{\mathcal{G}_{\Delta T}(\boldsymbol{U}_n^{k+1})}_{\text{Predictor}} + \underbrace{\mathcal{F}_{\Delta T}(\boldsymbol{U}_n^k) - \mathcal{G}_{\Delta T}(\boldsymbol{U}_n^k)}_{\text{Corrector}} \qquad 0 \leqslant k \leqslant n \leqslant N-1. \tag{2.9c}$$

The result is that a coarse initial guess (2.9b) is improved with successive Parareal iterations using the predictor-corrector (PC) update rule (2.9c). From this derivation, it still may not be clear how Parareal solves (2.1) in parallel and so we shall now describe how it works in practice.

2.2.2 How it works

Pseudocode for an implementation of Parareal is given in Algorithm 1 alongside a graphical illustration of the first iteration in Figure 2.2. To begin (iteration $k = 0$), a coarse approximation (2.9b) to (2.1) is calculated by applying $\mathcal{G}_{\Delta T}$ sequentially to the exact initial condition (2.9a) on a single processor. Following this, the fine solver is used to propagate each approximation in (2.9b) *in parallel*, on N processors, to obtain $\mathcal{F}_{\Delta T}(\boldsymbol{U}_n^0)$, $n = 0, \ldots, N-1$. These values are then ready to be used (during iteration $k = 1$) in the PC update (2.9c). Starting with the known exact solution at t_1, $\mathcal{G}_{\Delta T}$ is applied to 'predict' the state at t_2 and is then 'corrected' using the residual between the fine and coarse states from the prior iteration. This prediction and correction process is repeated *sequentially* up to time t_N. The next step is to check the stopping criteria, to determine whether Parareal has "converged" or not.

For a pre-defined stopping tolerance $\varepsilon > 0$, the Parareal states $\boldsymbol{U}_n^k$ are deemed to have converged up to time t_I if

$$\|\boldsymbol{U}_n^k - \boldsymbol{U}_n^{k-1}\|_\infty < \varepsilon \quad \forall n \leqslant I, \tag{2.10}$$

where $\|\cdot\|_\infty$ denotes the usual infinity norm. This criterion is standard for Parareal (Gander and Hairer, 2008; Garrido et al., 2006; Maday and Turinici, 2002), however, other choices are available. For example, one could instead take the relative (instead of absolute) error, the average relative error between fine solutions over a time slice (Samaddar et al., 2010, 2019), or measure the total energy of the system at each iteration (Dai et al., 2013). Unconverged states, i.e. $\boldsymbol{U}_n^k$ for $n > I$, are updated in future iterations $k > 1$ by carrying out further parallel $\mathcal{F}_{\Delta T}$ runs on each $\boldsymbol{U}_n^k$, followed by an update using the PC (2.9c). Once $I = N$, we say that Parareal has taken k

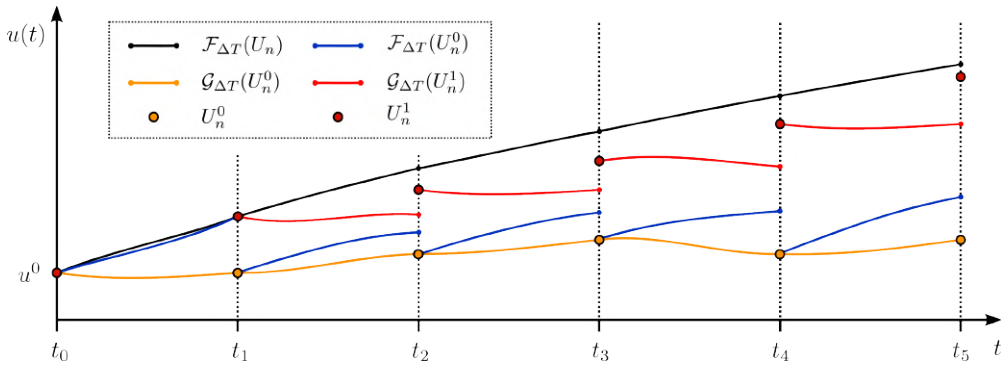


Figure 2.2: First iteration of Parareal to numerically approximate the exact solution (2.3) of a scalar ODE, obtained via a serial run of the fine solver (black line). The coarse initial guess found using $\mathcal{G}_{\Delta T}$ (yellow lines and dots) is followed by the parallel runs of $\mathcal{F}_{\Delta T}$ from these guesses (blue lines). The coarse predictions from $\mathcal{G}_{\Delta T}$ (red lines) are then used in the PC update (2.9c) (red dots). Note that the blue and black trajectories in the first interval will overlap, we distinguish them here for clarity.

(out of a maximum N) iterations to converge. By saying that Parareal *converges* in k iterations, we do not necessarily mean that Parareal has recovered the fine solution (2.3). In fact, we are slightly abusing terminology by instead meaning that Parareal *stops* after k iterations—this convention is widely accepted and embedded within the Parareal literature. Whilst Parareal should in fact recover the fine solution, it will not recover it exactly. A more detailed discussion on what it means for the Parareal solution to converge to the fine solution will be given in Section 2.2.4. In its original formulation, Parareal iteratively improves solutions across *all* time steps, regardless of whether they have converged or not. The version of Parareal described here does not iterate over solutions which have already converged, avoiding the waste of computational resources—this has no effect on the final number of iterations (Elwasif et al., 2011; Garrido et al., 2006). This modification allows us to incorporate stochastic sampling in Chapter 3 and the emulation processes in Chapter 5.

2.2.3 Computational complexity

Estimating computational complexity is an important stage in the analysis of any algorithm (especially those designed to run in parallel), enabling one to compare the performance of different algorithms that are designed to solve the same problem. Complexity analysis usually involves calculating the number of FLOPS required to solve a given problem, however, this is difficult for Parareal due to the freedom in the choice of the solvers, number of time slices and time steps. Instead, we will measure computational complexity in terms of wallclock time, parallel speedup and parallel efficiency. We can then use these three quantities to directly compare the performance of Parareal, SParareal, and GParareal.

Algorithm 1: Parareal

Initialise: Set counters $k = I = 0$ and define $\boldsymbol{U}_n^k$, $\hat{\boldsymbol{U}}_n^k$ and $\tilde{\boldsymbol{U}}_n^k$ as the PC, coarse, and fine solutions at the n^{th} time step and k^{th} iteration respectively (recall $\boldsymbol{U}_0^k = \hat{\boldsymbol{U}}_0^k = \tilde{\boldsymbol{U}}_0^k = \boldsymbol{u}^0 \ \forall k$).

%Calculate initial guess using $\mathcal{G}_{\Delta T}$ serially on processor P_1.

for $n = 1$ **to** N **do**
  $\hat{\boldsymbol{U}}_n^0 = \mathcal{G}_{\Delta T}(\hat{\boldsymbol{U}}_{n-1}^0)$;
  $\boldsymbol{U}_n^0 = \hat{\boldsymbol{U}}_n^0$;
end
for $k = 1$ **to** N **do**
  %Propagate the PC states (from iteration $k-1$) on each slice by running $\mathcal{F}_{\Delta T}$ in parallel on processors $P_{I+1}, \ldots, P_N$.
  for $n = I + 1$ **to** N **do**
    $\tilde{\boldsymbol{U}}_n^{k-1} = \mathcal{F}_{\Delta T}(\boldsymbol{U}_{n-1}^{k-1})$;
  end
  %Propagate the PC states (at iteration k) with $\mathcal{G}_{\Delta T}$ on any processor. Then, correct this value using coarse and fine states obtained during iteration $k-1$ (*cannot* be carried out in parallel).
  for $n = I + 1$ **to** N **do**
    $\hat{\boldsymbol{U}}_n^k = \mathcal{G}_{\Delta T}(\boldsymbol{U}_{n-1}^k)$;
    $\boldsymbol{U}_n^k = \hat{\boldsymbol{U}}_n^k + \tilde{\boldsymbol{U}}_n^{k-1} - \hat{\boldsymbol{U}}_n^{k-1}$;
  end
  %Check if stopping criterion met, saving all solutions up to time step t_I before next iteration.
  $I = \max_{n \in \{I+1,\ldots,N\}} \|\boldsymbol{U}_i^k - \boldsymbol{U}_i^{k-1}\|_\infty < \varepsilon \ \forall i < n$;
  %If tolerance is met for all time steps, algorithm stops.
  if $I = N$ **then**
    return k, $\boldsymbol{U}_\cdot^k$;
  end
end

After k iterations in Parareal, the exact initial condition ($\boldsymbol{u}^0$) will have been propagated forward in time by $\mathcal{F}_{\Delta T}$ k times. Therefore, the solution up to time t_k (at minimum) will have converged to the fine solver solution—this property will be shown rigorously in Section 2.2.4. It should then be clear that if Parareal converges in $k = N$ iterations, the solution will be equal to the one found by calculating (2.3) serially, at even higher computational cost. This means that to realise significant parallel speedup, Parareal needs to converge in $k \ll N$ iterations. We will now show explicitly why this requirement is necessary.

Without loss of generality, assume running $\mathcal{F}_{\Delta T}$ over any time slice $[t_n, t_{n+1}]$, $n \in \{0, \ldots, N-1\}$, takes wallclock time $T_{\mathcal{F}}$ seconds—denote time $T_{\mathcal{G}}$ similarly for $\mathcal{G}_{\Delta T}$. Therefore, calculating (2.3) using $\mathcal{F}_{\Delta T}$ serially, takes approximately $T_{\text{serial}} = NT_{\mathcal{F}}$

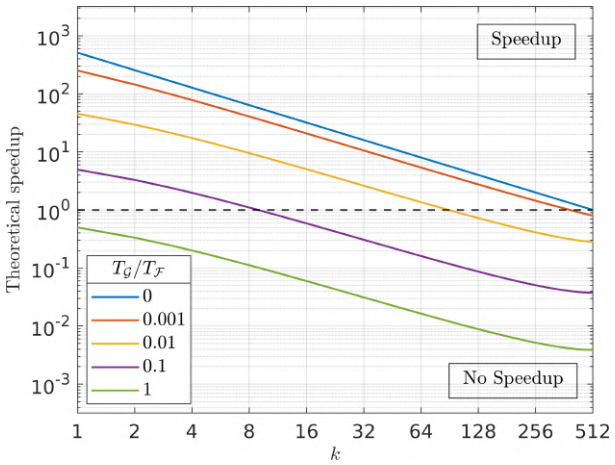


Figure 2.3: Theoretical speedup (2.12) of Parareal against iteration number k for problems with a varying ratio of coarse to fine solver wallclock runtimes ($T_\mathcal{G}/T_\mathcal{F}$) using $N = 512$ processors. The dashed line separates regions in which speedup and no speedup would be achieved.

seconds. Using Parareal, the total wallclock time (in the worst case, excluding any serial overheads and communication time) can be approximated by

$$\begin{aligned} T_{\text{para}} &\approx \underbrace{NT_\mathcal{G}}_{\text{Iteration 0}} + \underbrace{\sum_{i=1}^{k} \big(T_\mathcal{F} + (N-i)T_\mathcal{G}\big)}_{\text{Iterations 1 to } k} \\ &= kT_\mathcal{F} + (k+1)\big(N - \frac{k}{2}\big)T_\mathcal{G}. \end{aligned} \tag{2.11}$$

This estimate provides an approximate lower bound on the theoretical runtime of Parareal. This can then be used to estimate the parallel speedup

$$S_{\text{para}} \approx \frac{T_{\text{serial}}}{T_{\text{para}}} = \Big[\frac{k}{N} + (k+1)\big(1 - \frac{k}{2N}\big)\frac{T_\mathcal{G}}{T_\mathcal{F}}\Big]^{-1}. \tag{2.12}$$

Using the parallel speedup, we can also quantify the parallel efficiency

$$E_{\text{para}} \approx \frac{S_{\text{para}}}{N} = \left[k + (k+1)\big(N - \frac{k}{2}\big)\frac{T_\mathcal{G}}{T_\mathcal{F}}\right]^{-1}. \tag{2.13}$$

The efficiency provides a measure of how well the parallel computing resources are utilised in a given simulation. Typically for Parareal, the number of iterations required to converge to a solution is at least $k \geqslant 2$, meaning that the parallel efficiency can never exceed 0.5 (assuming negligible $T_\mathcal{G}$).

To maximise (2.12), both the number of iterations k and the ratio $T_\mathcal{G}/T_\mathcal{F}$ should be as small as possible. In Figure 2.3, we vary $T_\mathcal{G}/T_\mathcal{F}$ to examine the effect on theoretical speedup (2.12) for fixed $N = 512$ and varying k. We can see that if $T_\mathcal{G}$

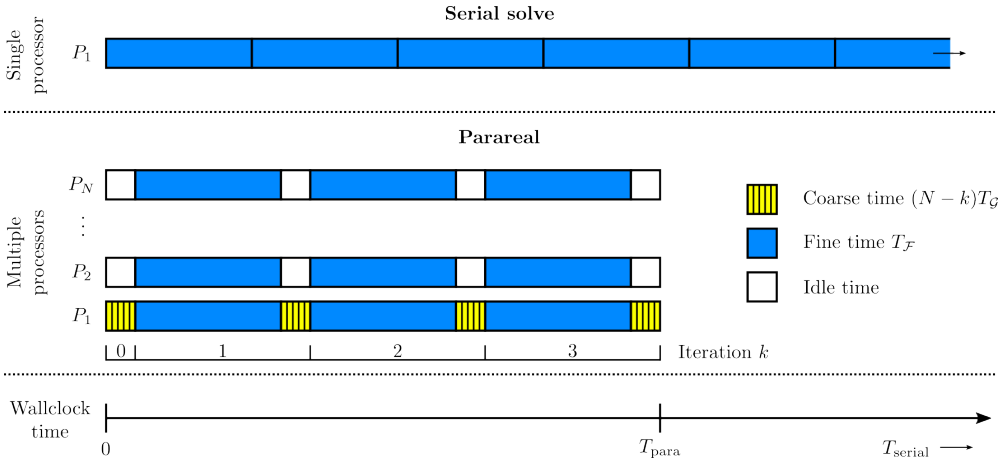


Figure 2.4: Computational task scheduling during three iterations of Parareal compared with a full serial integration. The coloured blocks represent the wallclock time any given processor spends on a task. Coarse runs are shown in yellow, fine runs in blue, and any idle time in white. The wallclock time is given on the axis at the bottom, indicating both T_{para} and T_{serial}.

is negligible compared to $T_{\mathcal{F}}$ then the speedup behaves like N/k and is realised for any $k < N$. However, once this ratio increases we see an increasing need for a lower iteration count in order to achieve any speedup (the lower the better). If $T_{\mathcal{G}}/T_{\mathcal{F}}$ is too large then no speedup is realised for any k. In practice, however, there is no way to *a priori* estimate k without actually carrying out the Parareal simulation. In addition, there is a trade-off between k and the ratio $T_{\mathcal{G}}/T_{\mathcal{F}}$, as fast $\mathcal{G}_{\Delta T}$ solvers (with sufficient accuracy to still guarantee convergence) typically cause Parareal to require more iterations to converge, increasing k. In any case, we can see that the best and worst cases are convergence in either $k = 1$ or $k = N$ iterations respectively. The speedup relation in (2.12) is useful as it can be used to estimate speedup for a range of k values using only the total number of processors available N and the ratio $T_{\mathcal{G}}/T_{\mathcal{F}}$ (which should be straightforward to estimate).

It is important to note that we ignore serial overheads (which include communication between processors and idle time) that inevitably cause discrepancies between numerical results and the theoretical estimates in (2.11)–(2.13). An illustration of the computational task scheduling during the first iteration of Parareal vs. a full serial integration is given in Figure 2.4. This scheduling implementation does not account for communication time and is clearly not optimal as a number of processors remain idle when coarse runs are being carried out. It is therefore worth mentioning that there has been work on optimising the standard scheduling process and redistributing load imbalance in Parareal (Aubanel, 2011; Bolten et al., 2022; Elwasif et al., 2011), leading to significant improvements in numerical speedup (almost double), however, these can often depend highly on the HPC architecture available (Ruprecht, 2017).

2.2.4 Error bound analysis

Deriving rigorous error bounds for PinT methods is important in demonstrating that numerical solutions obtained in parallel are meaningful, accurate, and that they can be compared to one another (Gander et al., 2022). In this section, we will outline the existing results that demonstrate how Parareal iteratively recovers the fine solution (2.3). We will use some of the proof techniques mentioned here to derive error bounds for both SParareal and GParareal in Chapters 4 and 5, respectively.

In the original work, Lions et al. (2001) derived an error bound for Parareal applied to the scalar linear ODE problem, i.e. $\boldsymbol{f}(t, \boldsymbol{u}(t)) := \lambda u$ in (2.1) for $\lambda \in \mathbb{C}$. They fixed the iteration number k, chose $\mathcal{F}_{\Delta T}$ as the exact solver, and $\mathcal{G}_{\Delta T}$ as the backward Euler (implicit RK1) method to find that

$$\max_{1 \leqslant n \leqslant N} |u(t_n) - U_n^k| \leqslant C_k \Delta T^{k+1},$$

where C_k is some constant that grows with k. This result shows that the Parareal error at a fixed iteration k goes to zero as $\Delta T \to 0$ and behaves like an $\mathcal{O}(\Delta T^{k+1})$ method. It does not, however, provide any information about how the algorithm behaves as k increases (due to the fact that C_k increases with k) and sending $\Delta T \to 0$ in Parareal is not really practical as this would require infinitely many time slices/processors N.

Subsequent work generalised this result, showing that if $\mathcal{G}_{\Delta T}$ is a method of order p (i.e. has local truncation error $\mathcal{O}(\Delta T^{p+1})$) then Parareal is a method of order $p(k+1))$ at iteration k (Bal, 2005; Bal and Maday, 2002). Noticing that ΔT should be fixed and k allowed to increase, Gander and Vandewalle (2007) instead derived error bounds for the scalar problem that have linear and superlinear rates of convergence on unbounded and bounded time intervals, respectively (see Appendix A for a brief recap on rates of convergence). With a little extra work, one can also relax the assumption that $\mathcal{F}_{\Delta T}$ is the exact solver, to derive a bound that holds when $\mathcal{F}_{\Delta T}$ is assumed to be a method of order q ($q \geqslant p$)—refer to (Gander and Vandewalle, 2007, Sec. 4.5).

Following the analysis on linear problems, Gander and Hairer (2008) derived the most general result so far. They used the generating function method (see Appendix D.2) to derive a superlinear bound for (autonomous) nonlinear systems of ODEs on bounded intervals. Our interest is in Theorem 2.2 (first derived by Gander et al. (2022)), a tighter bound than the one provided in Gander and Hairer (2008). As before, it is derived under the assumption that $\mathcal{F}_{\Delta T}$ is the exact solver and $\mathcal{G}_{\Delta T}$ is a method of order p which now satisfies a Lipschitz condition—these assumptions will be described in full detail in Section 4.2 when we derive similar error bounds for the SParareal scheme.

Theorem 2.2 (Superlinear error bound for Parareal)**.** *Suppose the Parareal scheme* (2.9) *satisfies Assumptions 4.2, 4.3, and 4.4 (see Section 4.2 for more details). Then, the error of the solution to the nonlinear ODE system* (2.1) *at iteration k and time t_n satisfies*

$$\|\boldsymbol{u}(t_n) - \boldsymbol{U}_n^k\|_\infty \leqslant DA^k \sum_{\ell=0}^{n-(k+1)} \binom{\ell+k}{\ell} B^\ell, \qquad 1 \leqslant k \leqslant n \leqslant N,$$

with constants $A = C_1 \Delta T^{p+1}$, $B = L_{\mathcal{G}}$, and $D = C_2 A$.

Proof. See Appendix B. □

From this result it can be seen that when the constant $A < 1$, the accuracy of Parareal improves with each iteration and that the error goes to zero when $\Delta T \to 0$, encapsulating the results of Lions et al. (2001) and Bal (2005). What is new is the superlinear convergence of the error toward zero as k increases (due to the k in both the binomial term and summation). Notice that, as expected, the error is exactly zero when $k = n$, i.e. after k propagations of $\mathcal{F}_{\Delta T}$, the exact solution is recovered at time step t_k. Convergence results such as Theorem 2.2 are beginning to appear for a number of Parareal variants using the generating function methodology. Examples include, Parareal for systems of ODEs that admit low-rank approximations (Carrel et al., 2022) and Parareal with multiple levels/time averaging (Rosemeier et al., 2022). In the latter case, bounds were also derived in the case where $\mathcal{F}_{\Delta T}$ is no longer assumed to be the exact solver, but instead a method order $q \geqslant p$, see (Rosemeier et al., 2022, Theorem 3.3).

2.2.5 Choice of numerical solvers

As we saw in Section 2.2.3, the choice of solvers $\mathcal{F}_{\Delta T}$ and $\mathcal{G}_{\Delta T}$ has a profound impact on the realisable parallel speedup from Parareal due to the trade-off between cost and accuracy (recall Figure 2.3). What follows is an outline of things to consider when choosing the solvers.

The fine solver $\mathcal{F}_{\Delta T}$

Typically, there is not much freedom in the choice of the fine solver $\mathcal{F}_{\Delta T}$ because the requirements for $\mathcal{F}_{\Delta T}$ (time step δt, order of accuracy q etc.) are determined by properties of the IVP being solved and by the needs of the user. For example, stiff problems may require the use of an adaptive $\mathcal{F}_{\Delta T}$ solver, chaotic problems may require a very small δt, and spatially-dependent PDEs (if using an explicit solver for $\mathcal{F}_{\Delta T}$) may need to satisfy a Courant-Friedrichs-Lewy condition (Courant et al., 1928). Furthermore, the user may instead wish to preserve certain features of the system by using a geometric integrator. The only requirement in any of these cases is

that $\mathcal{F}_{\Delta T}$ be sufficiently computationally expensive, i.e. $T_{\mathcal{F}}$ be sufficiently large, that $\mathcal{F}_{\Delta T}$ cannot be run over $[t_0, T]$ sequentially in feasible time. If it could, then there would be no need to use Parareal in the first place! In our numerical simulations, we will make use of high-order explicit RK methods.

The coarse solver $\mathcal{G}_{\Delta T}$

On the other hand, there is much more freedom in the choice of the coarse solver $\mathcal{G}_{\Delta T}$, even though a number of conditions must be satisfied for Parareal to remain stable and therefore converge to the fine solution. As discussed before, $\mathcal{G}_{\Delta T}$ must be chosen such that it is fast compared to the fine solver (i.e. $T_{\mathcal{G}} \ll T_{\mathcal{F}}$) but also accurate enough that it provides reasonable approximations to the solutions—criteria on how accurate $\mathcal{G}_{\Delta T}$ needs to be can be found in (Maday and Mula, 2020, Sec. 2.2). An additional factor to consider is the numerical stability of Parareal, which has been shown to depend explicitly on the stability of both $\mathcal{F}_{\Delta T}$ and $\mathcal{G}_{\Delta T}$ (Bal, 2005; Farhat and Chandesris, 2003; Ruprecht, 2014; Southworth, 2019; Staff and Rønquist, 2005). The choice of $\mathcal{F}_{\Delta T}$ should automatically be stable for solving the IVP of interest (otherwise an alternative method should be chosen!) and so one must be careful to select a stable coarse solver. It can seem natural to therefore choose an implicit solver for $\mathcal{G}_{\Delta T}$, as they are much more stable than explicit solvers, however, they are much more costly to implement (increasing the ratio $T_{\mathcal{G}}/T_{\mathcal{F}}$).

Without going into the technical details for satisfying the conditions of stability and accuracy[8], we now discuss some possible choices for $\mathcal{G}_{\Delta T}$. This choice will depend heavily on the IVP in question but should not really affect the accuracy of the final solution generated by Parareal as it should still recover the fine solution. The first option is to choose $\mathcal{G}_{\Delta T}$ to be the same solver as $\mathcal{F}_{\Delta T}$ but with a coarser temporal resolution, i.e. larger time steps $\delta T \geqslant \delta t$. It should be noted, however, that this may not be feasible for many IVPs where the size of δT is limited by the numerical stability of $\mathcal{F}_{\Delta T}$—which may depend on a spatial discretisation (Baffico et al., 2002; Baudron et al., 2014a; Gander and Hairer, 2008). Alternatively, $\mathcal{G}_{\Delta T}$ can be a different method than $\mathcal{F}_{\Delta T}$, one that has a lower order of numerical accuracy $p \leqslant q$ (Farhat and Chandesris, 2003; Samaddar et al., 2010; Trindade and Pereira, 2006). For more complex IVPs, work has shown that $\mathcal{G}_{\Delta T}$ can be chosen with coarser time steps and/or lower numerical accuracy to solve simplified model equations that have reduced physics or, perhaps, approximate the dynamics of the IVP (compared to $\mathcal{F}_{\Delta T}$) (Engblom, 2009; Grigori et al., 2021; Legoll et al., 2022; Meng et al., 2020). For spatially-dependent PDEs, $\mathcal{G}_{\Delta T}$ can resolve the IVP using a coarser spatial grid (Clarke et al., 2020; Fischer et al., 2005; Ruprecht, 2014; Samaddar et al., 2010).

[8]It is very difficult to translate the mathematical conditions required for the stability and accuracy of $\mathcal{F}_{\Delta T}$ and $\mathcal{G}_{\Delta T}$ into a practical method for choosing $\mathcal{F}_{\Delta T}$ and $\mathcal{G}_{\Delta T}$ for a given IVP. In most applications of Parareal, choices for the solvers (mostly always $\mathcal{G}_{\Delta T}$) have been guided by intuition or trial and error.

There have also been attempts to construct coarse solvers using different types of machine learning models—these will be discussed in detail in Section 5.1. Some of the aforementioned techniques have also been used in conjunction with one another.

All in all, one must carefully choose $\mathcal{G}_{\Delta T}$, bearing in mind that a more numerically accurate/stable coarse solver may reduce the number of iterations in Parareal but will increase the ratio $T_{\mathcal{G}}/T_{\mathcal{F}}$, leading to possible speedup degradation. For example, an implicit RK method with a large time step will be more numerically stable than its explicit RK counterpart, however, it will incur a much greater numerical cost due to the fact it has to solve a (typically nonlinear) system at each time step. This type of trade-off has led to a demand for accurate but fast coarse solvers for Parareal and has been the subject of much discussion (Nielsen, 2012). In our simulations, we will make use of low-order explicit/implicit methods for $\mathcal{G}_{\Delta T}$ which will be cheap compared to the selected fine solver.

Choosing an appropriate coarse solver is one of the most important aspects of running Parareal. It is the key to realising good numerical speedup and so a lot of effort has been dedicated to testing different approaches. Our goal is not to implement a new type of coarse solver but rather use probabilistic methods to try to harness the existing coarse (and fine) solution information obtained throughout a Parareal simulation. In Chapter 3, we sample from probability distributions with a variance proportional to the residual between coarse solves at different input locations (which is usually large during early iterations). The idea is that SParareal can take samples from these distributions and more efficiently explore the solution space, ideally converging to the exact solution values faster than Parareal can. In Chapter 5, we use a GP emulator to capture variability in the residual between the fine and coarse solvers (i.e. we model $\mathcal{F}_{\Delta T} - \mathcal{G}_{\Delta T}$), training the emulator on known fine and coarse information. We showcase the effectiveness of this approach by demonstrating that GParareal can converge to solutions in cases where the coarse solver is too poor (i.e. of insufficient accuracy) for Parareal. These efforts should help extract further numerical speedup and widen the pool of possible choices for $\mathcal{G}_{\Delta T}$ when running a Parareal-type simulation.

2.2.6 Numerical experiment: Arenstorf Orbit

In this section, we use Parareal to solve a system of ODEs that models a special case of the three-body problem—a similar experiment was presented in Gander and Hairer (2008). Suppose that a large body (e.g. the Earth) is orbited by a smaller, but still large, body (e.g. the Moon) in the two-dimensional plane. We consider the motion of an even smaller object (e.g. a satellite) between the two larger bodies (its mass is negligible compared to these bodies). In Figure 2.5, we can see that the Earth is fixed at the origin whilst the Moon, initially located at $(1, 0)$, orbits the Earth (dashed line). Arenstorf (1963) discovered equations that would allow for a

(single) stable periodic orbit of the small object between the Earth and Moon due to gravity (solid black line) over the period $t \in [0, 17.06521656015796]$. Coupled with initial conditions $\boldsymbol{u}(0) = (0.994, 0, 0, -2.00158510637908)^\intercal$, the system is given by

$$\begin{aligned}
\frac{\mathrm{d}u_1}{\mathrm{d}t} &= u_3, \\
\frac{\mathrm{d}u_2}{\mathrm{d}t} &= u_4, \\
\frac{\mathrm{d}u_3}{\mathrm{d}t} &= u_1 + 2u_4 - \frac{b(u_1+a)}{((u_1+a)^2+u_2^2)^{3/2}} - \frac{a(u_1-b)}{((u_1-b)^2+u_2^2)^{3/2}}, \\
\frac{\mathrm{d}u_4}{\mathrm{d}t} &= u_2 - 2u_3 - \frac{bu_2}{((u_1+a)^2+u_2^2)^{3/2}} - \frac{au_2}{((u_1-b)^2+u_2^2)^{3/2}},
\end{aligned} \tag{2.14}$$

where u_1 and u_2 are the positions of the small object at time t and u_3 and u_4 are the respective velocities. The constant $a = 0.012277471$ is the relative mass of the Moon compared to the combined mass of the Moon and Earth, while $b = 1 - a$ is the same but for Earth. This system is very sensitive to small changes in initial conditions and so accuracy (hence a small time step) is of paramount importance when integrating this system forward in time.

For this experiment, we select solvers $\mathcal{G}_{\Delta T} = \mathrm{RK2}$ and $\mathcal{F}_{\Delta T} = \mathrm{RK8}$ with $N_{\mathcal{G}} = (T - t_0)/\delta T = 1000$ and $N_{\mathcal{F}} = (T - t_0)/\delta t = 2 \times 10^5$ steps, respectively. We integrate over one orbital period using $N = 40$ time slices and a stopping tolerance $\varepsilon = 10^{-6}$.

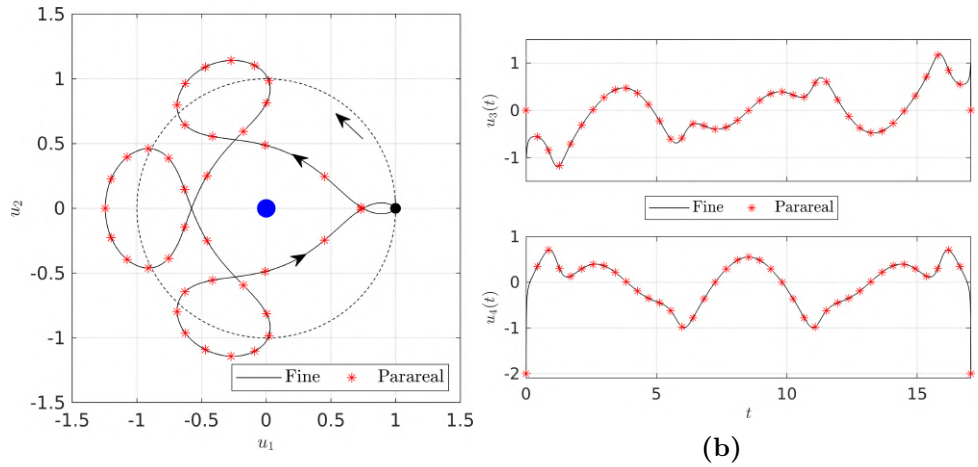


Figure 2.5: Numerical results obtained solving the Arenstorf system (2.14) over one orbital period using Parareal. (a) The position of the satellite, between the Moon (black dot) and Earth (blue dot), is plotted using the fine solver (solid black) and Parareal (red stars) in the (u_1, u_2)-plane (i.e. the (x, y)-plane). Also shown is the orbit of the Moon around the Earth (dashed black) and the direction of motion of both the satellite and the Moon (arrows). (b) The velocity of the satellite in the horizontal (u_3) and vertical (u_4) directions plotted against time using the fine solver (solid black) and Parareal (red stars). Note that Parareal solutions are plotted only at times $\boldsymbol{t}$ for clarity.

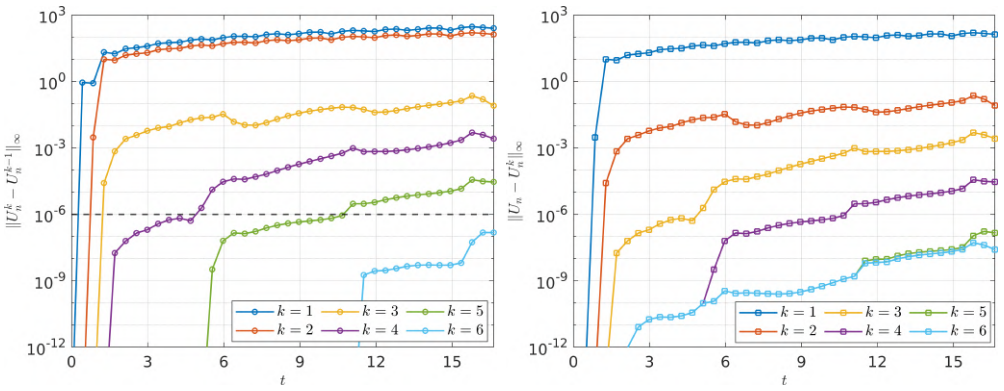


Figure 2.6: (a) Maximum absolute errors (2.10) between successive Parareal solutions ($\boldsymbol{U}_n^k$ and $\boldsymbol{U}_n^{k-1}$) over time, plotted at each iteration k until tolerance $\varepsilon = 10^{-6}$ is met. (b) Maximum absolute errors between the Parareal solution $\boldsymbol{U}_n^k$ and the fine solution $\boldsymbol{U}_n$ at each iteration k, plotted over time.

We observe a good match between the solutions simulated by Parareal (after reaching the stopping tolerance) and the solution obtained by running the fine solver serially. In Figure 2.6(a), we plot how the difference between Parareal solutions iteratively reach the stopping criterion (2.10), stopping once below ε for all time steps. After $k = 3$ iterations, we observe that the solution in the first three time slices have reached the stopping tolerance. Following the fourth iteration, however, seven time slices converge. What this means is that Parareal is avoiding having to run $\mathcal{F}_{\Delta T}$ seven times serially, instead only running $\mathcal{F}_{\Delta T}$ once (during the fourth iteration). The key to observing faster speedup with Parareal is if it can reach the stopping tolerance in more than one time slice each iteration (the more time slices that converge, the better). In a similar vein, we can show how close the Parareal solution is to the (serially obtained) fine solution at each iteration in Figure 2.6(b). As expected, the accuracy of the Parareal solution is poor after just one iteration—of order $\mathcal{O}(10^2)$ from the fine solution. The error reduces drastically within a few iterations, reaching $\mathcal{O}(10^{-7})$ after the stopping tolerance is met at iteration $k = 6$. These results demonstrate Parareal can locate the correct solution, even from a poor initial guess.

Table 2.1: Numerical wallclock time, speedup, and efficiency results obtained solving the Arenstorf system (2.14) using Parareal—refer back to Section 2.2.3 for notation. The results in brackets are the corresponding theoretical results calculated using (2.11)–(2.13). All timings are measured in seconds.

N	k	$T_{\mathcal{G}}$	$T_{\mathcal{F}}$	T_{serial}	T_{para}	S_{para}	E_{para}
40	6	2.92E−2	85.07	3.40E3	522.14 (517.97)	6.52 (6.57)	0.16 (0.16)

We do, however, need to calculate runtimes to truly demonstrate the power of Parareal. To do this, we increase $N_{\mathcal{F}}$ to 2×10^8, thereby increasing the cost of running $\mathcal{F}_{\Delta T}$ relative to $\mathcal{G}_{\Delta T}$ (so that $T_{\mathcal{G}}/T_{\mathcal{F}}$ is small and we can realise speedup)[9]. The numerical results in Table 2.1 display the runtimes of the coarse and fine solvers over a time slice, the runtime of the serial simulation, the Parareal runtime, and the corresponding speedup/efficiency realised. We see that Parareal takes $k = 6$ (out of a maximum $N = 40$) iterations to converge, yielding a wallclock runtime $6.52\times$ faster than what the serial fine solver can achieve. The iterative nature of Parareal, however, leads to very poor parallel efficiency—indicating that Parareal does not utilise its parallel resources very well. Overall, the numerical results match the theoretical estimates calculated in (2.11)–(2.13) nicely, with slight discrepancies coming from some serial overheads/parallel communication ignored by the theory. These brief results have given us an insight into how Parareal works in practice—many more simulations will be carried out using Parareal in later chapters.

2.3 Variants and related work

Now that we have discussed Parareal in detail, we provide a brief overview of some of the different variants that have emerged since its inception to tackle various challenges the standard implementation suffers from when faced with solving particular types of IVP. Note that the following variants are not necessarily related to the probabilistic PinT methods we propose later on and so we save a full exposition of those that are related to sampling- and learning-based methods until Chapters 3 and 5, respectively.

Parareal has been shown to work very well for diffusive (e.g. parabolic-type) IVPs, however, it has been known to repeatedly struggle with non-diffusive (e.g. hyperbolic or advection-dominated) IVPs (Bal, 2005; Gander, 2008; Staff and Rønquist, 2005). Non-diffusive IVPs occur frequently throughout the physical sciences, most notably perhaps in fluid dynamics, where differential operators contain imaginary eigenvalues and therefore solutions exhibit wave-type behaviour. By writing Parareal in iteration matrix form, Ruprecht (2018) investigates its wave propagation characteristics, finding that instability (i.e. slow or non-convergence) arises due to phase errors generated by the coarse solver and amplification factors in higher wave numbers. Iizuka and Ono (2018) report similar findings through numerical investigation and show that using coarse solvers with the same phase accuracy as the fine solver is one way to avoid deteriorating performance, however, one must note that using such solvers is typically more computationally expensive. One variant that has been shown to work well on non-diffusive IVPs is *Krylov-enhanced Parareal* (Gander and Petcu, 2008;

[9]We excessively increase $N_{\mathcal{F}}$ in experiments throughout this thesis to increase $T_{\mathcal{F}}$ and generate meaningful speedup results that are largely absent of communication and other serial costs. Clearly this would not be done in practice but is necessary for some of the low-dimensional systems tested here.

Ruprecht and Krause, 2012), which improves the accuracy of the coarse solver using Krylov subspaces built from Parareal solution information. In Chapter 5 we will discuss this variant at length in the context of learning-based methods. Also worth mentioning is the variant developed by Buvoli and Minion (2023) in which they try to stabilise Parareal by slightly re-formulating the IVP (splitting it into linear and nonlinear parts) and using exponential RK methods as coarse/fine solvers. Ensuring PinT algorithms can handle non-diffusive IVPs is of great importance given that they arise so frequently in mathematical modelling, e.g. numerical weather prediction and plasma physics.

When solving certain IVPs (e.g. Hamiltonian systems) we may also wish to preserve invariant dynamics or conserve geometric/physical quantities of the system (e.g. energy, angular momentum) over long integration times. Bal and Wu (2008) and Dai et al. (2013) show that a blind application of Parareal using symplectic coarse/fine integrators (that serially conserve such properties on their own) does not work. This is because the summation of symplectic functions in the PC does not preserve symplecticity and so a slightly more involved method of composing the symplectic functions in the PC is required. Gander and Hairer (2014) analyse this problem further, deriving long-time error estimates for Parareal applied to Hamiltonian systems.

There are far too many Parareal variants to cover entirely in this section, however, we mention a few more for completeness[10]. Some variants tackle time-scale separation (Haut and Wingate, 2014; Legoll et al., 2020; Rosemeier et al., 2022), time periodicity (Gander et al., 2013b), and adaptive slicing of the time interval (to avoid solution blow up) in longtime simulations of molecular dynamics (Legoll et al., 2022). Others have emerged to to deal with more specific IVPs including: Parareal for low-rank systems (Carrel et al., 2022), PDE constrained optimisation (Gander et al., 2020), and data assimilation (Bhatt et al., 2022).

2.4 Summary

In this chapter, we formulated the problem of interest and stated our objective of finding a high accuracy numerical solution to the IVP (2.1). Typically, such a solution (2.3) is obtained by using an expensive (serial) time stepping method known as the fine solver $\mathcal{F}_{\Delta T}$. We derived Parareal from first principles, showing how it makes use of $\mathcal{F}_{\Delta T}$ as well as a cheaper, lower accuracy coarse solver $\mathcal{G}_{\Delta T}$ to integrate the IVP in parallel using a PC update rule. Its iterative nature means that convergence occurs in k iterations, yielding a maximal speedup of N/k (that we hope to improve upon). Whilst $\mathcal{F}_{\Delta T}$ can be chosen (almost) freely, we caution that $\mathcal{G}_{\Delta T}$ must be chosen to be computationally inexpensive (compared to $\mathcal{F}_{\Delta T}$) but still

[10] A more complete list can be found at http://parallel-in-time.org/references/index.html.

numerically accurate enough to roughly approximate the IVP solution. Theoretical computational complexity estimates were provided and will be repeatedly referred to when we need to analyse and compare our own probabilistic PinT methods with Parareal. In addition, we provided a brief insight into the kind of error bound analysis that we will need to carry out to demonstrate that our algorithms locate accurate solutions. The demonstration of how Parareal works in practice on a small test problem is given to highlight its iterative nature (and therefore show when it stops), how we measure numerical accuracy (against the $\mathcal{F}_{\Delta T}$ solutions), and what kind of numerical speedup it can generate. The framework of this chapter will help set the stage for the probabilistic PinT algorithms we are about to present.

Chapter 3

SParareal I: a sampling-based time-parallel algorithm

Overview

In this chapter we propose SParareal, a sampling-based time-parallel algorithm that can solve IVPs (2.1) using probability distributions constructed from known fine and coarse solution data. The idea is to sample candidate solutions from these distributions in each time slice, propagate each one on its own processor using the fine solver, and select an optimal state that will provide us with a "better" correction in the Parareal PC update. The intuition is that with sufficiently many samples, better corrections will lead to a reduction in the number of iterations taken until convergence, yielding increased parallel speedup.

We begin in Section 3.1 by explaining the intuition behind SParareal and discussing how the first PinT algorithm and a sampling-based PN ODE solver inspired its development. In Section 3.2, we introduce SParareal, explaining how it works, how the probability distributions are constructed and how the sampling and propagation process is carried out. We then elucidate how a variety of different "sampling rules" can be flexibly interchanged to carry out the sampling depending on whether any information is known about the solution to the IVP *a priori* to simulation. This is followed by remarks on computational complexity, where we derive expressions that show the wallclock time of an SParareal iteration is approximately the same as one in Parareal. We also discuss how an increasing number of samples reduces the number of iterations required until convergence. To conclude, we remark on the convergence of the SParareal solutions to the exact fine solution (2.3)—which will be analysed fully in Chapter 4.

In Section 3.3, we conduct numerical experiments that showcase the performance of SParareal using different sampling rules and varying numbers of random samples. Findings are presented for three nonlinear ODE systems (additional experiments are

provided in Appendix C), demonstrating that for sufficiently many samples, SParareal almost certainly converges in fewer iterations than Parareal and generates (stochastic) solutions of comparable numerical accuracy. Results show that performance is improved by generating correlated, as opposed to uncorrelated, random samples. In Section 3.4, we discuss the advantages and disadvantages of SParareal, highlighting what can be improved and how this lead to the development of GParareal in Chapter 5.

3.1 Motivation and background

Throughout this chapter, we seek the same high resolution numerical solutions to (2.1) as given by (2.3). The iteratively improved solutions from SParareal will be denoted using the same notation as Parareal, i.e. as $\boldsymbol{U}_n^k$ (where $\boldsymbol{U}_0^k = \boldsymbol{U}_0 = \boldsymbol{u}^0 \ \forall k \geqslant 0$). The IVP setup, the solvers, and the notation will be the same as Parareal except where otherwise defined.

3.1.1 Our approach

As we saw in Section 2.2.3, the major obstacle preventing further parallel speedup gains in (2.12) is the choice of $\mathcal{G}_{\Delta T}$ and so focus is often directed toward locating faster, more accurate, coarse solvers to achieve reductions in both k and $T_{\mathcal{G}}$. Finding better coarse solvers is notoriously difficult and in many applications there may not be any alternative choices for $\mathcal{G}_{\Delta T}$ due to limitations on step sizes or required numerical accuracy. In the PC (2.9c), the solution states $\boldsymbol{U}_n^k$ are updated deterministically using a correction term based on a single fine and coarse solution from the previous iteration $k-1$. Our aim is to improve the accuracy of this correction, not by modifying $\mathcal{G}_{\Delta T}$, but by making use of the existing fine and coarse solution data to from prior iterations. We do this by using the data to construct probability distributions over regions of state space where we believe the exact states $\boldsymbol{U}_n$ may exist. Sampling from these distributions should allow us to more efficiently explore the solution space and locate the exact states in faster wallclock time.

The main idea is that, instead of using a single deterministically calculated correction term to update the PC solution, we sample M candidate initial values $\boldsymbol{\alpha}_{n,1}^k, \ldots, \boldsymbol{\alpha}_{n,M}^k$, at each unconverged time slice t_n. These samples are drawn from probability distributions defined by a pre-specified sampling rule with given marginal means and standard deviations (see Section 3.2.2). These parameters are defined using the most recently obtained fine and coarse solution information so that we ensure samples are drawn in the neighbourhood of the current PC solution $\boldsymbol{U}_n^k$. All of the sampled initial values are then propagated in parallel using $\mathcal{F}_{\Delta T}$. Given a sufficient number of samples is taken, one sample will be closer (in the Euclidean sense) to the exact root $\boldsymbol{U}_n$ that equation (2.9c) is converging toward. To try and locate the best sample at each t_n, we select an "optimal" sample $\hat{\boldsymbol{\alpha}}_n^k$ by identifying

which sequence of samples generate the most continuous trajectory, at the fine resolution, in state space across $[t_0, t_N]$. These optimal samples are then subsequently propagated (rapidly) forward in time using $\mathcal{G}_{\Delta T}$ and then used directly in the new PC update:

$$\boldsymbol{U}_{n+1}^{k+1} = \underbrace{\mathcal{G}_{\Delta T}(\boldsymbol{U}_n^{k+1})}_{\text{prediction}} + \underbrace{\mathcal{F}_{\Delta T}(\hat{\boldsymbol{\alpha}}_n^k) - \mathcal{G}_{\Delta T}(\hat{\boldsymbol{\alpha}}_n^k)}_{\text{correction}}, \quad 1 \leqslant k \leqslant n < N.$$

The intuition is that for increasing values of M, the stochastically generated set of initial values should be closer to the exact states $\boldsymbol{U}_n$ than those found purely deterministically and therefore converge in fewer iterations.

For example, consider a scenario in which the PC in Parareal is provided initial values $\boldsymbol{V}^{(0)}$, yielding convergence in k iterations and generating the sequence of solutions $\{\boldsymbol{V}^{(0)}, \boldsymbol{V}^{(1)}, \ldots, \boldsymbol{V}^{(k)}\}$. Instead of starting with $\boldsymbol{V}^{(0)}$, suppose we sample initial values from a probability distribution and choose some "better" starting point which is close to, say, $\boldsymbol{V}^{(i)}$ for some $i \in \{1, \ldots, k-1\}$. Then the sequence generated by the PC would instead be approximately $\{\boldsymbol{V}^{(i)}, \boldsymbol{V}^{(i+1)}, \ldots, \boldsymbol{V}^{(k)}\}$, converging in $k - i + 1$ iterations. Therefore, given a fixed number of samples M, SParareal should converge in fewer than k iterations with some non-zero probability, returning a stochastic solution to the IVP. Stochastic methods, that generate different solutions after each independent simulation, are useful in that they are able to explore the solution space more than deterministic methods, potentially revealing unexpected behaviour in the dynamical system. In addition, if we can obtain such solutions in faster wallclock time than Parareal then one can launch additional simulations obtaining a distribution of stochastic trajectories over the solution (something we will show later).

3.1.2 Related work

The idea of propagating multiple initial values in each time slice forward in time (in parallel) is not new. In the first known work proposing a PinT method, Nievergelt (1964) proposed solving IVPs in parallel by choosing the M (sampled) initial values, discussed above, deterministically. For a scalar IVP, he suggests choosing M initial values $U_{n,m}^0$ in each time slice t_n close to the exact (unknown) solution U_n and carrying out $M(N-1)+1$ propagations using $\mathcal{F}_{\Delta T}$ in parallel—see Figure 3.1 for an illustration. The M values are chosen close to some initial guess U_n^0, e.g. a coarse initial guess as in Parareal. The method for determining the solution across $[t_0, T]$ from this *ensemble* of trajectories is to (sequentially) combine two of the samples in each time slice $[t_n, t_{n+1}]$ using an interpolation coefficient, i.e.

$$U_{n+1}^1 = r\mathcal{F}_{\Delta T}(U_{n,m}^0) + (1-r)\mathcal{F}_{\Delta T}(U_{n,m+1}^0), \tag{3.1}$$

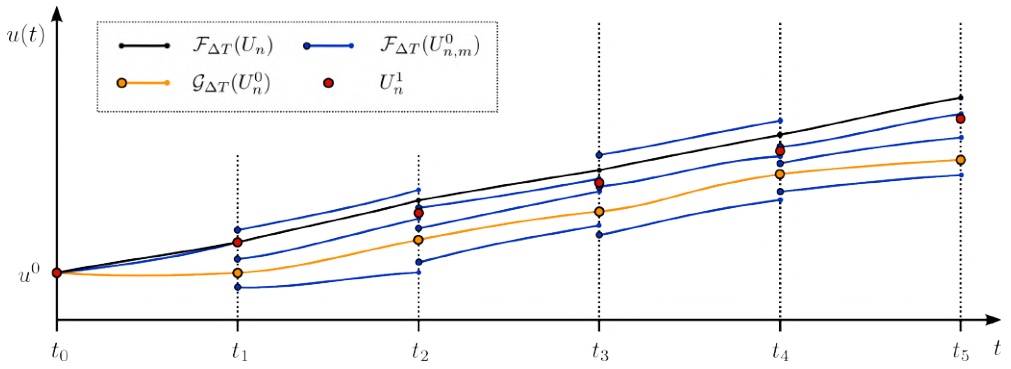


Figure 3.1: Illustration of Nievergelt's method for numerically approximating the exact solution (2.3) of a scalar ODE, obtained via a serial run of the fine solver (black line). A number of candidate initial values $U^0_{n,m}$ (blue dots), selected close to a coarse initial guess U^0_n calculated using $\mathcal{G}_{\Delta T}$ (yellow dots and lines), are propagated forward in time using $\mathcal{F}_{\Delta T}$ in parallel (blue dots and lines). These trajectories are used to directly calculate a solution to the IVP using (3.1) (red dots).

where $r = \left(U^1_n - U^0_{n,m+1}\right) / \left(U^0_{n,m} - U^0_{n,m+1}\right)$ and m is chosen such that $U^1_n \in [U^0_{n,m}, U^0_{n,m+1}]$. This direct (non-iterative) method of solving in parallel works well for scalar linear ODEs, however, suffers from interpolation errors for nonlinear problems, does not generalise for systems of ODEs, and questions remain over how to efficiently (or correctly) choose the M initial values. Nievergelt knew that even though his algorithm was not a practical method for solving IVPs, it would lay the foundations for better methods in the future that have now become known as PinT methods[1]. With SParareal, we address the problem of not being able to solve nonlinear systems of ODEs by using the existing architecture of Parareal and tackle the issue of how to correctly and efficiently choose the M initial values by using the probability distributions described in Section 3.2.2.

The idea of sampling the M initial values from probability distributions stems from the sampling-based ODE solvers (also known as perturbative ODE solvers) developed in the field of PN. Conrad et al. (2017) developed a (sequential) sampling-based ODE solver in which solution states generated by a numerical integrator (such as $\mathcal{F}_{\Delta T}$) are perturbed with Gaussian noise to try to quantify numerical uncertainty in the solution to the ODE. Traditionally, one integrates (2.1) forward in time by applying $\mathcal{F}_{\Delta T}$ sequentially, just as in (2.3). In sampling-based solvers, $\mathcal{F}_{\Delta T}$ is assumed to be a one-step numerical method, i.e. $\delta t = \Delta T$, with local truncation

[1]In his conclusions, Nievergelt summed this up best himself by saying that: "The integration methods introduced in this paper are to be regarded as tentative examples of a much wider class of numerical procedures in which parallelism is introduced at the expense of redundancy of computation. As such, their merits lie not so much in their usefulness as numerical algorithms as in their potential as prototypes of better methods based on the same principles. It is believed that more general and improved versions of these methods will be of great importance when computers capable of executing many computations in parallel become available."

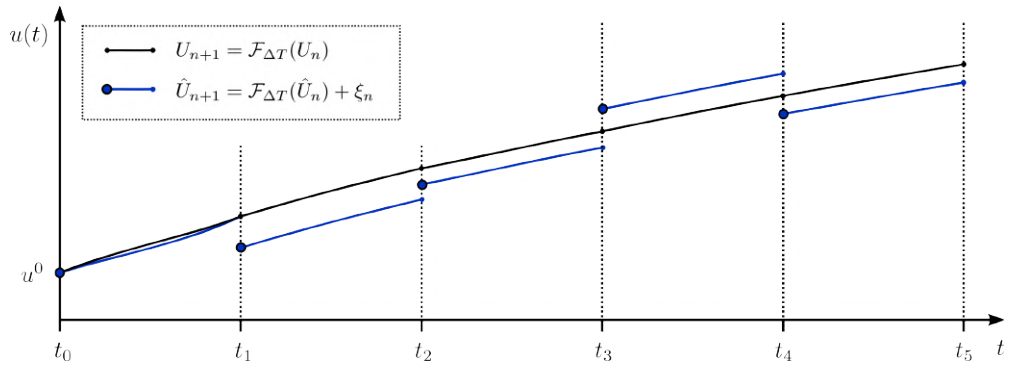


Figure 3.2: An illustration of the sampling-based ODE solver proposed by Conrad et al. (2017). The deterministic solution trajectory (black line) obtained using $\mathcal{F}_{\Delta T}$ is shown alongside one realisation of the sampling-based trajectory (blue lines) obtained using (3.2).

error of order $q+1$ ($q \geqslant 1$). This means that $\boldsymbol{U}_n = \mathcal{F}_{\Delta T}(\boldsymbol{U}_{n-1})$ is accurate with respect to the true solution $\boldsymbol{u}(t_n)$ up to $\mathcal{O}(\Delta T^q)$. Conrad et al. exploit the fact that we are free to select any point within the small "ball" centred on $\boldsymbol{U}_n$, radius $C\Delta T^{q+1}$ (C constant) to propagate forward in time to t_{n+1}. They do this by adding zero-mean, suitably scaled (second moment scaled $\mathcal{O}(\Delta T^{2q+1})$), independent identically distributed (i.i.d.) Gaussian perturbations $\boldsymbol{\xi}_n$ after each propagation using $\mathcal{F}_{\Delta T}$, proposing the numerical scheme

$$\hat{\boldsymbol{U}}_{n+1} = \mathcal{F}_{\Delta T}(\hat{\boldsymbol{U}}_n) + \boldsymbol{\xi}_n, \quad n = 0, \ldots, N-1. \tag{3.2}$$

The idea is that an approximation to the numerical error, i.e. $\boldsymbol{\xi}_n$, is added to the numerical solution after each time step, thus generating a sequence of random variables that captures numerical uncertainty in the solution—see Figure 3.2. Upon multiple simulations of the algorithm, one then obtains a non-Gaussian distribution of solutions rather than a single deterministic trajectory—recall Figure 1.4(a). Whilst this distribution is unstructured and analytically intractable to analyse post-simulation (at present), it does have the power to reveal qualitative features of the system being solved, e.g. stable or unstable manifolds, that "uncertainty-unaware" deterministic solvers cannot. Although we do not directly use random perturbations in SParareal to quantify numerical uncertainty[2], we do use them to explore the solution space and try to accelerate convergence of the deterministic Parareal scheme.

The key takeaway from the work of Conrad et al. (2017) is that the mean-square error of the solutions (3.2) are accurate with respect to the global truncation

[2]In Section 3.3 we will, however, see that upon multiple simulations of SParareal, one can obtain a distribution of solutions to the IVP. We do not claim that such a distribution captures any quantitative numerical uncertainty from the solvers but rather it can provide a more qualitative understanding of how solutions to the IVP behave.

error of $\mathcal{F}_{\Delta T}$ (assuming perturbations have an appropriate bound on their second moments). This means that the solutions (3.2) are of the same order of accuracy as those obtained deterministically using (2.3). Note that the stochastic trajectories illustrated in Figure 3.2 were purposely drawn far away from the deterministic trajectory to emphasise their stochastic nature only. This will be a feature that SParareal mimics with respect to the solutions from Parareal (i.e. they will be of the same order of accuracy). In follow up work, Lie et al. (2019, 2022) extend the theoretical analysis of the solution errors to allow for more general, e.g. non Gaussian, non centred, non i.i.d., perturbations. The underlying assumptions made about the perturbations by Lie et al. will be very useful when we derive mean-square error bounds for the SParareal solutions in Chapter 4. One downside of these solvers is that if $\mathcal{F}_{\Delta T}$ is a solver that preserves some geometric property of the IVP being solved, e.g. energy, then the random perturbations don't preserve these properties. In response to this problem, Abdulle and Garegnani (2020) developed a sampling-based solver in which they randomise the time steps rather than the solutions to preserve such geometric properties—it would be interesting to see if this is compatible with PinT methods. For an extended discussion on sampling-based ODE solvers refer to Hennig et al. (2022).

Finally, for completeness we must note the Parareal variants that have been developed to solve SDEs, with applications mainly focused on molecular dynamics (Baffico et al., 2002; Bal, 2006; Engblom, 2009; Legoll et al., 2020, 2022). These variants typically use a fine SDE solver to integrate the SDE to high resolution and a coarse solver that uses either a larger time step (which is often difficult to do in molecular dynamics simulations) or solves reduced (deterministic) model equations. Either way, the Parareal scheme requires some form of adaptation so that these solutions can be combined in a PC-type scheme. When applying such variants to SDEs, the solutions obtained are inherently stochastic due to the nature of the SDE and, to an extent, the solvers. With SParareal, the IVP is strictly deterministic and the SParareal scheme itself is what introduces randomness that generates stochastic solutions to the (deterministic) IVP. One would need to think carefully about how to apply SParareal to SDEs.

3.2 The algorithm

We are now ready to introduce SParareal and explain how it works.

3.2.1 How it works

Following initialisation, the first iteration ($k = 1$) of SParareal is the same as the first in Parareal—refer to pseudocode in Algorithm 2. This is because the coarse and fine solution information generated up to $k = 1$ is required to construct the probability

distributions for sampling. After the convergence check, we assume (for the purposes of explaining the stochastic iterations) that only the first time slice $[t_0, t_1]$ converges during $k = 1$, leaving $N - 1$ unconverged time slices. At this point we know the most up-to-date PC solutions $\boldsymbol{U}_n^1$ $\forall n \in \{1, \ldots, N\}$ and the stochastic iterations can begin (henceforth $k = 2$).

At any unconverged t_n ($n > 1$), we sample M vectors of initial values, denoted $\boldsymbol{\alpha}_{n,m}^{k-1}$ for $m = 1, \ldots, M$. The first sample is fixed as the PC solution state $\boldsymbol{U}_n^{k-1}$, to ensure that SParareal and Parareal are equivalent when $M = 1$. This helps ensure that the performance of SParareal should be no worse than Parareal (to be discussed further later) and enables us to easily turn the sampling "on" ($M > 1$) and "off" ($M = 1$) in numerical experiments. The other $M - 1$ initial values are sampled from a pre-specified d-dimensional probability distribution Φ_n^{k-1} with finite marginal means $\boldsymbol{\mu}_n^{k-1} = (\mu_{n_1}^{k-1}, \ldots, \mu_{n_d}^{k-1})^\intercal$, marginal standard deviations $\boldsymbol{\sigma}_n^{k-1} = (\sigma_{n_1}^{k-1}, \ldots, \sigma_{n_d}^{k-1})^\intercal$, and correlation structure given by the matrix $\boldsymbol{R}_n^{k-1} \in \mathbb{R}^{d \times d}$. These quantities depend upon the solution information available up to iteration $k - 1$, i.e. a combination of $\boldsymbol{U}_n^{k-1}$, $\mathcal{F}_{\Delta T}(\boldsymbol{U}_n^{k-1})$, $\mathcal{G}_{\Delta T}(\boldsymbol{U}_n^{k-1})$ and $\mathcal{G}_{\Delta T}(\boldsymbol{U}_n^{k-2})$, see Section 3.2.2. The correlation matrix $\boldsymbol{R}_n^{k-1}$ is introduced to take into account any dependence between components of the ODE system (lines 2-7, Algorithm 2). For example, in spatially discretised PDE problems, a system of ODEs govern the dynamics at the spatial locations and so one would expect that solutions of ODEs "next to one another" would be highly correlated and those "further away" would be less correlated. The elements of $\boldsymbol{R}_n^{k-1}$, for $k \geqslant 3$, are defined using the Pearson correlation coefficient

$$\rho_{n_{i,j}}^{k-1} = \frac{\sum_{m=1}^{M}(x_m^{(i)} - \bar{x}^{(i)})(x_m^{(j)} - \bar{x}^{(j)})}{\sqrt{\sum_{m=1}^{M}(x_m^{(i)} - \bar{x}^{(i)})^2}\sqrt{\sum_{m=1}^{M}(x_m^{(j)} - \bar{x}^{(j)})^2}}, \quad i, j \in \{1, \ldots, d\}, \tag{3.3}$$

where

$$x_m^{(i)} = \mathcal{F}_{\Delta T}(\boldsymbol{\alpha}_{n-1,m}^{k-2})(i), \qquad \bar{x}^{(i)} = \frac{1}{M}\sum_{m=1}^{M} x_m^{(i)},$$

and $\mathcal{F}_{\Delta T}(\boldsymbol{\alpha}_{n-1,m}^{k-2})(i)$ denotes the ith element of $\mathcal{F}_{\Delta T}(\boldsymbol{\alpha}_{n-1,m}^{k-2})$. The coefficients $\rho_{n_{i,j}}^{k-1}$ in (3.3) are the estimated pairwise correlation coefficients of the M d-dimensional fine resolution propagations of the sampled initial values at t_n from the previous iteration, i.e. $\mathcal{F}_{\Delta T}(\boldsymbol{\alpha}_{n-1,1}^{k-2}), \ldots, \mathcal{F}_{\Delta T}(\boldsymbol{\alpha}_{n-1,M}^{k-2})$. Note that other types of linear correlation coefficient can be chosen. Since each $\mathcal{F}_{\Delta T}(\boldsymbol{\alpha}_{n-1,m}^{k-2})$ is not available at iteration $k = 2$, we set $\boldsymbol{R}_n^{k-1} = \mathbb{I}_d$ for $k = 2$, i.e. we sample from a multivariate distribution with uncorrelated components.

Following this, the sampling and subsequent propagation using $\mathcal{F}_{\Delta T}$ can begin in parallel (lines 8-19). Given the solution between $[t_0, t_1]$ has converged, $\mathcal{F}_{\Delta T}$ will run from the converged initial value at t_1, with sampling starting from t_2 onward (see

Algorithm 2: SParareal

Initialise: Run Parareal (Algorithm 1) until the end of iteration $k = 1$.

for $k = 2$ **to** N **do**
  %Calculate correlations if $d > 1$, recall (3.3).
  $\boldsymbol{R}_n^{k-1} = \mathbb{I}_d \ \forall n$;
  if $k \geqslant 3$ **then**
    for $n = I + 1$ **to** $N - 1$ **do**
      Calculate $\boldsymbol{R}_n^{k-1}$ using $\mathcal{F}_{\Delta T}(\boldsymbol{\alpha}_{n-1,1}^{k-2}), \ldots, \mathcal{F}_{\Delta T}(\boldsymbol{\alpha}_{n-1,M}^{k-2})$;
    end
  end
  %Sampling and propagation. Lines 8-10 must run in parallel on $P_1, \ldots, P_{M(N-1-I)+1}$.
  $\tilde{\boldsymbol{U}}_{I+1}^{k-1} = \mathcal{F}_{\Delta T}(\boldsymbol{U}_I^{k-1})$; %propagate converged value at t_I on P_1
  for $n = I + 1$ **to** $N - 1$ **do**
    for $m = 1$ **to** M **do**
      if $m = 1$ **then**
        $\boldsymbol{\alpha}_{n,1}^{k-1} = \boldsymbol{U}_n^{k-1}$; %first 'sample' is fixed to PC value
        $\tilde{\boldsymbol{U}}_{n+1,1} = \mathcal{F}_{\Delta T}(\boldsymbol{\alpha}_{n,1}^{k-1})$; %store propagated values
      else
        $\boldsymbol{\alpha}_{n,m}^{k-1} \sim \Phi_n^{k-1}$; %sample initial value randomly
        $\tilde{\boldsymbol{U}}_{n+1,m} = \mathcal{F}_{\Delta T}(\boldsymbol{\alpha}_{n,m}^{k-1})$;
      end
    end
  end
  %Sequentially select most continuous fine trajectory.
  for $n = I + 1$ **to** $N - 1$ **do**
    $J = \underset{j \in \{1,\ldots,M\}}{\operatorname{argmin}} \|\boldsymbol{\alpha}_{n,j}^{k-1} - \tilde{\boldsymbol{U}}_n^{k-1}\|_2$;
    $\hat{\boldsymbol{\alpha}}_n^{k-1} = \boldsymbol{\alpha}_{n,J}^{k-1}$; %store optimal initial value
    $\tilde{\boldsymbol{U}}_{n+1}^{k-1} = \tilde{\boldsymbol{U}}_{n+1,J}$; %store most optimal fine trajectories
  end
  %Run $\mathcal{G}_{\Delta T}$ from the optimal samples (can run in parallel).
  for $n = I + 1$ **to** $N - 1$ **do**
    $\hat{\boldsymbol{U}}_{n+1}^{k-1} = \mathcal{G}_{\Delta T}(\hat{\boldsymbol{\alpha}}_n^{k-1})$;
  end
  %Predict and correct the initial values.
  for $n = I + 1$ **to** N **do**
    $\hat{\boldsymbol{U}}_n^k = \mathcal{G}_{\Delta T}(\boldsymbol{U}_{n-1}^k)$;
    $\boldsymbol{U}_n^k = \hat{\boldsymbol{U}}_n^k + \tilde{\boldsymbol{U}}_n^{k-1} - \hat{\boldsymbol{U}}_n^{k-1}$;
  end
  %Check whether the stopping criterion is met.
  $I = \max_{n \in \{I+1,\ldots,N\}} \|\boldsymbol{U}_i^k - \boldsymbol{U}_i^{k-1}\|_\infty < \varepsilon \ \forall i < n$;
  if $I = N$ **then**
    return k, $\boldsymbol{U}_\cdot^k$; %if tolerance met for all time steps, stop.
  end
end

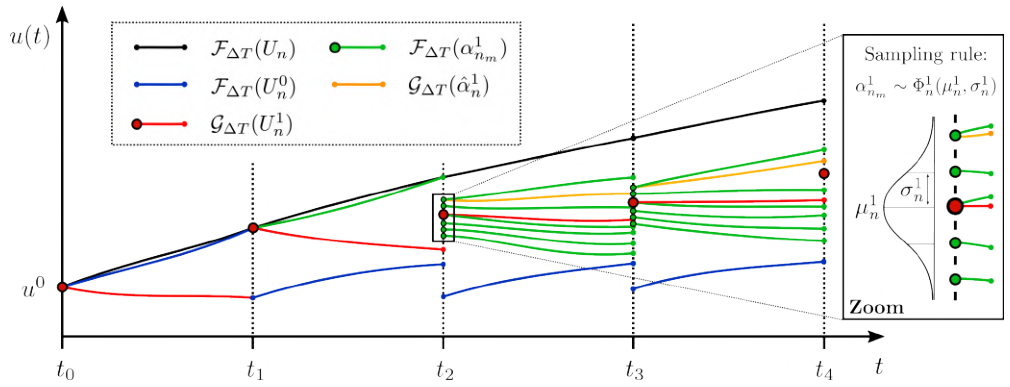


Figure 3.3: An illustration of the sampling and propagation process within SParareal following iteration $k = 1$. The fine solution is given in black, the $k = 0$ fine solutions in blue, the $k = 1$ coarse solutions in red, and the $k = 1$ PC solutions as red dots. With $M = 5$, four samples $\alpha_{n,m}^1$ (green dots) are taken at t_2 and t_3 from distributions with means U_2^1 and U_3^1, and some finite standard deviations respectively. These values, along with U_2^1 and U_3^1 themselves, are propagated in parallel forward in time using $\mathcal{F}_{\Delta T}$ (green lines). The optimally chosen samples $\hat{\alpha}_n^1$ (refer to text for how these are chosen) are then propagated forward in time using $\mathcal{G}_{\Delta T}$ (yellow lines).

Figure 3.3). *All* sampled initial values are then propagated forward in parallel using $\mathcal{F}_{\Delta T}$, requiring at least $M(N-2)+1$ processors (M samples on $N-2$ unconverged time slices plus running $\mathcal{F}_{\Delta T}$ once in $[t_1, t_2]$).

Of the M sampled initial values at each t_n ($n > 1$), only one is retained (denoted by $\hat{\boldsymbol{\alpha}}_n^{k-1}$) chosen such that it minimises the Euclidean distance between the fine solution and the sampled values (lines 20-24). To do this, start from the converged initial values at t_2 given by the fine solver: $\mathcal{F}_{\Delta T}(\boldsymbol{U}_1^{k-1})$. Calculate the Euclidean distance between $\mathcal{F}_{\Delta T}(\boldsymbol{U}_1^{k-1})$ and each of the M samples $\boldsymbol{\alpha}_{2,1}^{k-1}, \ldots, \boldsymbol{\alpha}_{2,M}^{k-1}$. The sample minimising this distance is chosen as $\hat{\boldsymbol{\alpha}}_2^{k-1}$. Repeat for later t_n, minimising the distance between $\mathcal{F}_{\Delta T}(\hat{\boldsymbol{\alpha}}_{n-1}^{k-1})$ and one of the samples $\boldsymbol{\alpha}_{n,1}^{k-1}, \ldots, \boldsymbol{\alpha}_{n,M}^{k-1}$. This process must be run sequentially and relies on the modification to Parareal discussed at the end of Section 2.2.2—that solutions are not altered once converged. Referring again to Figure 3.3, the corresponding coarse trajectories of these optimally chosen samples $\hat{\boldsymbol{\alpha}}_n^{k-1}$ must also be calculated to carry out the PC step (lines 25-27).

At this point, the set of initial values $\{\hat{\boldsymbol{\alpha}}_2^{k-1}, \ldots, \hat{\boldsymbol{\alpha}}_{N-1}^{k-1}\}$ has been selected from the ensemble of random samples, effectively replacing the previously found $\boldsymbol{U}_n^{k-1}$. The coarse and fine propagations of these values are now used in the PC (lines 28-31) such that

$$\boldsymbol{U}_n^k = \mathcal{G}_{\Delta T}(\boldsymbol{U}_{n-1}^k) + \mathcal{F}_{\Delta T}(\hat{\boldsymbol{\alpha}}_{n-1}^{k-1}) - \mathcal{G}_{\Delta T}(\hat{\boldsymbol{\alpha}}_{n-1}^{k-1}), \quad 2 \leqslant k \leqslant n \leqslant N. \tag{3.4}$$

Note that $\hat{\boldsymbol{\alpha}}_{n-1}^{k-1} = \boldsymbol{U}_{n-1}^{k-1}$ for $n = k$, as no sampling takes place in the left-most (already converged) time slice. Using the same stopping criteria (2.10) from Parareal

(lines 32-35), the algorithm either stops or runs another SParareal iteration. For completeness, we define the SParareal scheme as we did for Parareal:

Definition 3.1 (SParareal)**.** For the two numerical flow maps $\mathcal{F}_{\Delta T}$ and $\mathcal{G}_{\Delta T}$, described in Section 2.2, the SParareal scheme is given by

$$\boldsymbol{U}_0^0 = \boldsymbol{u}^0, \tag{3.5a}$$
$$\boldsymbol{U}_{n+1}^0 = \mathcal{G}_{\Delta T}(\boldsymbol{U}_n^0), \qquad 0 \leqslant n \leqslant N-1, \tag{3.5b}$$
$$\boldsymbol{U}_{n+1}^1 = \mathcal{G}_{\Delta T}(\boldsymbol{U}_n^1) + \mathcal{F}_{\Delta T}(\boldsymbol{U}_n^0) - \mathcal{G}_{\Delta T}(\boldsymbol{U}_n^0), \qquad 0 \leqslant n \leqslant N-1, \tag{3.5c}$$
$$\boldsymbol{U}_{n+1}^{k+1} = \mathcal{G}_{\Delta T}(\boldsymbol{U}_n^{k+1}) + \mathcal{F}_{\Delta T}(\hat{\boldsymbol{\alpha}}_n^k) - \mathcal{G}_{\Delta T}(\hat{\boldsymbol{\alpha}}_n^k), \qquad 1 \leqslant k \leqslant n \leqslant N-1, \tag{3.5d}$$

where $\hat{\boldsymbol{\alpha}}_n^k = \boldsymbol{U}_n^k$ when $n = k$.

As a final remark, instead of minimising the distance between $\mathcal{F}_{\Delta T}(\hat{\boldsymbol{\alpha}}_{n-1}^{k-1})$ and one of the samples $\boldsymbol{\alpha}_{n,1}^{k-1}, \ldots, \boldsymbol{\alpha}_{n,M}^{k-1}$, one could think about doing some sort of interpolation (recall the work of Nievergelt (1964)) to choose a more optimal point "between" the M samples. This, however, is not possible in the Parareal setting because we require not just the exact starting condition, which would be the optimally chosen sample, but also its value having been propagated by $\mathcal{F}_{\Delta T}$ (which we only have for the M samples).

3.2.2 Sampling rules

The probability distributions Φ_n^{k-1} incorporate different combinations of available solution information, i.e. the coarse, fine, and PC solution values $\mathcal{G}_{\Delta T}(\boldsymbol{U}_{n-1}^{k-1})$, $\mathcal{G}_{\Delta T}(\boldsymbol{U}_{n-1}^{k-2})$, $\mathcal{F}_{\Delta T}(\boldsymbol{U}_{n-1}^{k-2})$, and $\boldsymbol{U}_n^{k-1}$, respectively. This information is used to define the marginal means and standard deviations in the following "sampling rules". Using Gaussian and copula distributions, we analyse the numerical performance of different sampling rules within SParareal in Section 3.3. This will give us a more comprehensive understanding of whether the choice of distribution family Φ_n^{k-1} or the parameters $\boldsymbol{\mu}_n^{k-1}$, $\boldsymbol{\sigma}_n^{k-1}$, and $\boldsymbol{R}_n^{k-1}$ have the greatest impact on the number of iterations until convergence.

Multivariate Gaussian

First, we consider perturbing the solution states using Gaussian "noise", i.e. considering errors compared to the exact solution states to be normally distributed, a standard method for modelling uncertainty—similar assumptions were made in the work of Conrad et al. (2017). The Gaussian probability density function over $\mathbb{R}^d$ is given by

$$\mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{d/2} \det \boldsymbol{\Sigma}^{1/2}} \exp\left(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^\intercal \boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})\right), \quad \boldsymbol{x} \in \mathbb{R}^d. \tag{3.6}$$

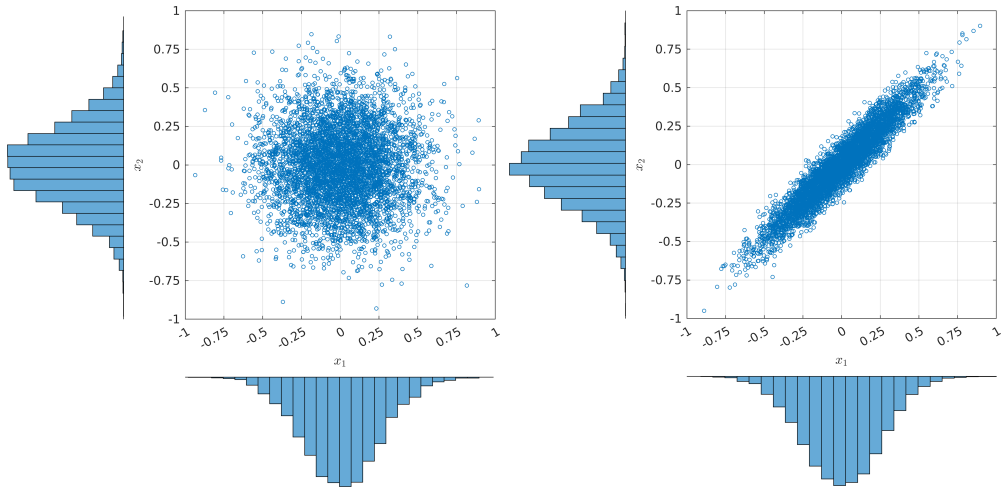


Figure 3.4: Illustration of 5000 samples taken from two different bivariate Gaussian distributions $\boldsymbol{X} = (x_1, x_2)^\intercal \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$. (a) Uncorrelated samples: $\boldsymbol{\mu} = (0,0)^\intercal$ and $\boldsymbol{\Sigma} = [0.625, 0; 0, 0.625]$. (b) Positively correlated samples: $\boldsymbol{\mu} = (0,0)^\intercal$ and $\boldsymbol{\Sigma} = [0.625, 0.594; 0.594, 0.625]$. The histograms depict the marginal distributions of $\boldsymbol{X}$, i.e. samples are approximately distributed as $x_1 \sim \mathcal{N}(0, 0.625)$ and $x_2 \sim \mathcal{N}(0, 0.625)$ in both cases.

The parameters $\boldsymbol{\mu} \in \mathbb{R}^d$ and $\boldsymbol{\Sigma} \in \mathbb{R}^{d \times d}$ are the mean vector and the (symmetric positive semi-definite) covariance matrix, respectively. A real-valued random $\boldsymbol{X}$ is said to be Gaussian (normally) distributed if $\boldsymbol{X} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$. An illustration of samples taken from two bivariate ($d = 2$) Gaussian distributions with different covariance matrices is given in Figure 3.4. Recall that the marginal distributions of a multivariate Gaussian are also Gaussian.

The values $\boldsymbol{\alpha}_{n,m}^{k-1}$ are sampled from $\mathcal{N}(\boldsymbol{\mu}_n^{k-1}, \boldsymbol{\Sigma}_n^{k-1})$ with marginal means denoted $\boldsymbol{\mu}_n^{k-1} = (\mu_{n_1}^{k-1}, \ldots, \mu_{n_d}^{k-1})^\intercal$ and covariance matrix $(\boldsymbol{\Sigma}_n^{k-1})_{i,j} = \rho_{n_{i,j}}^{k-1} \sigma_{n_i}^{k-1} \sigma_{n_j}^{k-1}$. For the mean vector $\boldsymbol{\mu}_n^{k-1}$, we choose either the fine solution states $\mathcal{F}_{\Delta T}(\boldsymbol{U}_{n-1}^{k-2})$ (prior to correction) or the PC states $\boldsymbol{U}_n^{k-1}$. For the marginal standard deviations[3], we choose $\boldsymbol{\sigma}_n^{k-1} = |\mathcal{G}_{\Delta T}(\boldsymbol{U}_{n-1}^{k-1}) - \mathcal{G}_{\Delta T}(\boldsymbol{U}_{n-1}^{k-2})|$ as they are of the order of the corrections made by the PC and each marginal decreases toward zero as the algorithm converges—the importance of this property will be highlighted in Chapter 4. For the correlation coefficients, $\rho_{n_{i,j}}^{k-1}$, we calculate the linear correlation between the $\mathcal{F}_{\Delta T}$ propagated samples using Pearson's method—recall (3.3). The samples $\boldsymbol{\alpha}_{n,m}^{k-1} \sim \mathcal{N}(\boldsymbol{\mu}_n^{k-1}, \boldsymbol{\Sigma}_n^{k-1})$

[3]Testing revealed that alternative marginal standard deviations $|\boldsymbol{U}_n^{k-1} - \boldsymbol{U}_n^{k-2}|$ and $|\mathcal{F}_{\Delta T}(\boldsymbol{U}_{n-1}^{k-2}) - \mathcal{G}_{\Delta T}(\boldsymbol{U}_{n-1}^{k-2})|$ did not span sufficiently large distances around $\boldsymbol{\mu}_n^{k-1}$ in order for sampling to be efficient, i.e they required much higher sampling to perform as well as $\boldsymbol{\sigma}_n^{k-1} = |\mathcal{G}_{\Delta T}(\boldsymbol{U}_{n-1}^{k-1}) - \mathcal{G}_{\Delta T}(\boldsymbol{U}_{n-1}^{k-2})|$ (results not shown). Also, note that $|\cdot|$ denotes the component-wise absolute value.

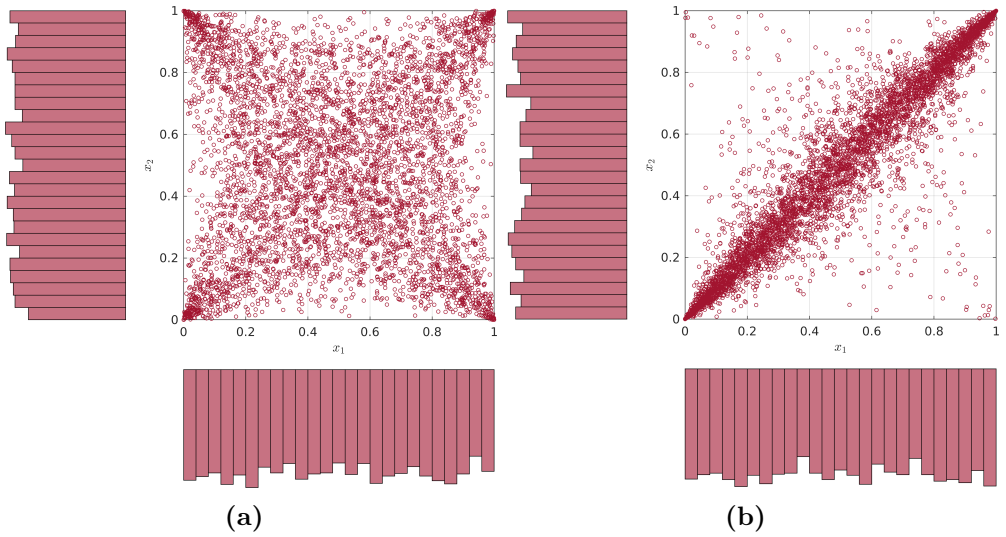


Figure 3.5: Illustration of 5000 samples taken from two different bivariate t-copula distributions $\boldsymbol{X} = (x_1, x_2)^\intercal \sim \mathcal{C}^t$ with $\nu = 1$. (a) Uncorrelated samples: $\boldsymbol{R} = [1, 0; 0, 1]$. (b) Correlated samples: $\boldsymbol{R} = [1, 0.95; 0.95, 1]$. The histograms depict the marginal distributions of $\boldsymbol{X}$, i.e. samples are approximately distributed as $x_1 \sim \mathcal{U}(0, 1)$ and $x_2 \sim \mathcal{U}(0, 1)$ in both cases.

are taken according to the following sampling rules:

$$\textbf{Rule 1}: \ \boldsymbol{\mu}_n^{k-1} = \mathcal{F}_{\Delta T}(\boldsymbol{U}_{n-1}^{k-2}) \text{ and } \boldsymbol{\sigma}_n^{k-1} = |\mathcal{G}_{\Delta T}(\boldsymbol{U}_{n-1}^{k-1}) - \mathcal{G}_{\Delta T}(\boldsymbol{U}_{n-1}^{k-2})|.$$
$$\textbf{Rule 2}: \ \boldsymbol{\mu}_n^{k-1} = \boldsymbol{U}_n^{k-1} \text{ and } \boldsymbol{\sigma}_n^{k-1} = |\mathcal{G}_{\Delta T}(\boldsymbol{U}_{n-1}^{k-1}) - \mathcal{G}_{\Delta T}(\boldsymbol{U}_{n-1}^{k-2})|.$$

Note that a linear combination of both rules (or taking half the samples from each) also works well, with performance similar to the individual rules themselves (results not shown).

Multivariate copula

Now we wish to consider the case where samples are drawn from a multivariate uniform distribution, allowing us to compare the performance of using perturbations with (uniform) and without (Gaussian) closed support. The multivariate uniform distribution, however, does not contain a dependency structure, i.e. random samples are assumed to be uncorrelated. To be able to include such a dependency structure, we consider using copulas.

A copula $\mathcal{C} : [0, 1]^d \to [0, 1]$ is a joint cumulative distribution function with uniform marginal distributions (Nelsen, 2006). Sklar's theorem states that any multivariate cumulative distribution function with continuous marginal distributions can be written in terms of d uniform marginal distributions and a copula that describes the correlation structure between them (Sklar, 1959). This will allow

us to sample from a multivariate distribution with uniform marginal distributions and a given correlation structure (that we can turn on and off). We will build these copulas in such a way that they have the same marginal means and standard deviations as sampling rules 1 and 2. While there are numerous families of copula to choose from, we consider the symmetric t-copula $\mathcal{C}^t$ that underlies the multivariate t-distribution. It depends on a parameter ν (representing the degrees of freedom) and the correlation matrix $\boldsymbol{R}$ which will encode the dependency structure (this will be $\boldsymbol{R}_n^{k-1}$ in SParareal). In Figure 3.5 we plot samples from two bivariate ($d = 2$) t-copulas with parameter $\nu = 1$ and different correlation matrices—we use this value of ν in our experiments. One can see that the effect of $\nu = 1$ means samples have a higher probability of being drawn toward the edges of the box $[0, 1]^2$, i.e. the "tails" of the distribution, than a Gaussian distribution in both cases. In addition, notice how the marginal distributions are both uniformly distributed. If we were to send $\nu \to \infty$, we would obtain the Gaussian copula that has uniform marginals and a Gaussian dependency structure—see Nelsen (2006) for more details on copulas.

To compare with sampling rules 1 and 2, the correlated samples $\boldsymbol{X} \sim \mathcal{C}^t$ generated in $[0, 1]^d$ need to be re-scaled such that each marginal is uniformly distributed in an interval $[a_i, b_i] \subset \mathbb{R}$, with mean μ_i and standard deviation σ_i for $i \in \{1, \ldots, d\}$. By definition, a marginal uniform distribution on $[a_i, b_i]$ has mean $(a_i+b_i)/2$ and variance $(b_i - a_i)^2/12$ which we set equal to μ_i and σ_i^2, respectively. Solving these equations, we find that the desired marginals are uniform distributions on $[\mu_i - \sqrt{3}\sigma_i, \mu_i + \sqrt{3}\sigma_i]$. Therefore, scaling by $2\sqrt{3}\sigma_i X_i + \mu_i - \sqrt{3}\sigma_i$ guarantees that the generated samples $\boldsymbol{X} \sim \mathcal{C}^t$ have the same marginal means μ_i and standard deviations σ_i as the Gaussian sampling rules. This will allow us to compare the performance of both distribution families in Section 3.3. The t-copula sampling rules (**Rules 3** and **4**) are therefore defined component-wise as $\boldsymbol{\alpha}_{n,m}^{k-1}(i) = 2\sqrt{3}\sigma_i X_i + \mu_i - \sqrt{3}\sigma_i$, for $i \in \{1, \ldots, d\}$, with $\boldsymbol{X} \sim \mathcal{C}^t$ and parameters μ_i and σ_i chosen to be identical to Rule 1 and Rule 2, respectively.

3.2.3 Computational complexity

During each iteration, SParareal uses the fine solver more frequently than Parareal, albeit still in parallel, and therefore requires a larger number of processors. The first iteration of SParareal requires N processors, however, once sampling begins ($k \geqslant 2$), it requires at most $M(N - I - 1) + 1$ processors—assuming I time slices converge during $k = 1$. This number scales directly with M and so sampling may be limited if a small number of processors are available.

As the stochastic iterations progress, the number of processors required, i.e. $M(N - I - 1) + 1$, decreases as the number of converged time slices I increases. Each additional time slice that converges leaves M processors idle meaning that we can re-assign them to do additional sampling and propagation. We assign each set

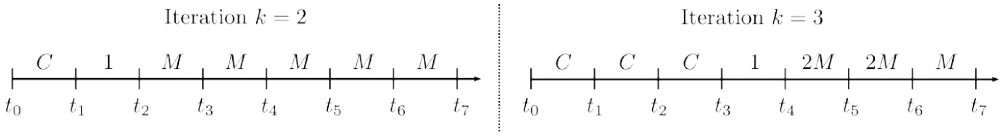


Figure 3.6: Illustration of a possible processor configuration if two time slices were to converge between iterations two and three of SPararealal. The letter C denotes a converged time slice, the number 1 denotes a time slice where we propagate the converged initial value from the preceding time slice using $\mathcal{F}_{\Delta T}$, and the letter M denotes the number of samples taken in an unconverged time slice.

of M idle processors to the earliest unconverged time slice with the least number of samples, ensuring all processors are working at all times to explore the solution space for the exact solution states (see Figure 3.6 for an illustration). We do not explicitly write the pseudocode for re-assigning the idle processors in Algorithm 2 to avoid additional complexity—the process is, however, implemented in the numerical experiments in Section 3.3 [4].

As we did with Parareal, we can estimate the wallclock time, speedup, and parallel efficiency of SPararealal—refer back to Section 2.2.3 for notation. Assuming that the additional serial costs in SPararealal, e.g. correlation estimation and selecting optimal samples, take negligible wallclock time compared to $T_{\mathcal{F}}$, we can estimate the total wallclock time for SPararealal as

$$\begin{aligned} T_{\text{SPara}} \approx\ & \underbrace{NT_{\mathcal{G}}}_{\text{Iteration 0}} + \underbrace{T_{\mathcal{F}} + (N-1)T_{\mathcal{G}}}_{\text{Iteration 1}} + \sum_{i=2}^{k} \underbrace{\big(T_{\mathcal{F}} + 2(N-i)T_{\mathcal{G}}\big)}_{\text{Iterations 2 to } k} \\ =\ & kT_{\mathcal{F}} + \big(2kN - k(k+1) + 1\big)T_{\mathcal{G}}. \end{aligned} \tag{3.7}$$

Note that the summation term includes the additional cost of running $\mathcal{G}_{\Delta T}$ for the optimal samples (as well as the runs of $\mathcal{G}_{\Delta T}$ carried out in the PC step). The parallel speedup of SPararealal can then be written as

$$S_{\text{SPara}} \approx \frac{T_{\text{serial}}}{T_{\text{SPara}}} = \left[\frac{k}{N} + \left(2k - \frac{k}{N}(k+1) + \frac{1}{N}\right)\frac{T_{\mathcal{G}}}{T_{\mathcal{F}}}\right]^{-1}, \tag{3.8}$$

and the parallel efficiency as

$$E_{\text{SPara}} \approx \frac{S_{\text{SPara}}}{NM} = \frac{1}{M}\left[k + (2kN - k(k+1) + 1)\frac{T_{\mathcal{G}}}{T_{\mathcal{F}}}\right]^{-1}. \tag{3.9}$$

Assuming the ratio $T_{\mathcal{G}}/T_{\mathcal{F}}$ is negligible, we expect that iterations in SPararealal and

[4] To increase efficiency further we also attempted to store previously sampled and propagated fine trajectories to use in future iterations of the algorithm, however, they did not improve performance. This was because only the most recently obtained samples were ever chosen in each iteration (results not shown).

Parareal will have very similar runtimes. Therefore, if SParareal converges in fewer iterations than Parareal, the wallclock time for SParareal will be lower than for Parareal (as we will see in Section 3.3). This does, however, come at a cost of requiring $\mathcal{O}(MN)$ processors rather than N to solve the IVP, which in turn degrades the parallel efficiency (3.9) of the algorithm quite significantly. Although, it should be highlighted that if SParareal converges in even one less iteration than Parareal, we avoid an extra run of $\mathcal{F}_{\Delta T}$ which will save at least $T_{\mathcal{F}}$ seconds of wallclock time.

3.2.4 Convergence

It should be clear that given the stochastic nature of SParareal, it will return a numerical solution $\boldsymbol{U}_n^k$ which varies stochastically between independent simulations of the algorithm. Given the inherent randomness encoded in the numerical solutions, we can discuss the convergence of SParareal in two ways.

Firstly, we consider convergence in terms of minimising k_s, the number of iterations taken for SParareal to converge (noting that this is a random variable). We can do this by studying $\mathbb{P}(k_s < k)$, the probability of SParareal converging in fewer iterations than Parareal. Proving that $\mathbb{P}(k_s < k) = 1$, or at least $\mathbb{E}(k_s) = \sum_{i=1}^{N} i\mathbb{P}(k_s = i) < k$, will be challenging given that there are no analytical results for Parareal guaranteeing that $k < N$ for any given problem. We can, however, qualitatively discuss $\mathbb{P}(k_s < k)$ and $\mathbb{E}(k_s)$ with respect to the number of samples M. Consider the following cases:

(i) $M = 1$
Running SParareal is equivalent to running Parareal, hence the convergence of SParareal follows from that of Parareal and therefore $\mathbb{E}(k_s) = k_s = k$.

(ii) $1 < M < \infty$
When using finitely many samples, we compute the discrete probability distributions $\mathbb{P}(k_s = k)$ and observe, in all numerical experiments (see Section 3.3), that $\mathbb{P}(k_s < k) \to 1$ as M increases. Moreover, we observe $\mathbb{E}(k_s)$ decreases for increasing M, with $\mathbb{E}(k_s) < k$ for all values of M tested.

(iii) $M \to \infty$
If we were able to take infinitely many samples, SParareal effectively samples every possible value in the support of Φ, i.e. every solution state that has a non-zero probability of being sampled from Φ. Therefore, if Φ has infinite support, e.g. the Gaussian distribution, all possible solution states in $\mathbb{R}^d$ are sampled and propagated, hence the fine solution will be recovered almost surely in $\mathbb{E}(k_s) = 2$ iterations. Note this is the smallest value k_s can take to converge assuming convergence does not occur following the first iteration. In Section 3.3.1, we try to illustrate this property numerically by taking a large number of samples for a single realisation of SParareal.

In the scenario that SParareal converges in $k_s = N$ iterations (irrespective of the value of M), it will return the fine solution just as Parareal does when $k = N$ (having propagated the exact initial value at t_0 sequentially N times using $\mathcal{F}_{\Delta T}$).

Secondly, we can consider convergence in terms of the stochastic solution states $\boldsymbol{U}_n^k$ (3.4) approaching (in the mean-square sense) the exact solution states $\boldsymbol{U}_n$ as k increases. Deriving rigorous mean-square error bounds for SParareal, however, requires a lot of set up and so we postpone this analysis until Chapter 4. There we will give a full exposition of the convergence of SParareal, deriving explicit mean-square errors bounds under a number of different assumptions on the solvers and for the sampling rules. In the numerical experiments in Section 3.3, we do not observe a case where SParareal fails to converge to the exact solution. In fact, we observe tight confidence intervals on the numerical errors between $\boldsymbol{U}_n^k$ and $\boldsymbol{U}_n$ upon multiple realisations of SParareal (see Figure 3.10 and Figure 3.14). The only situation in which SParareal may fail to converge (i.e. solutions blow up) is in cases in which Parareal also fails—typically this means that a more accurate coarse solver is required for both algorithms.

3.3 Numerical experiments: nonlinear ODEs

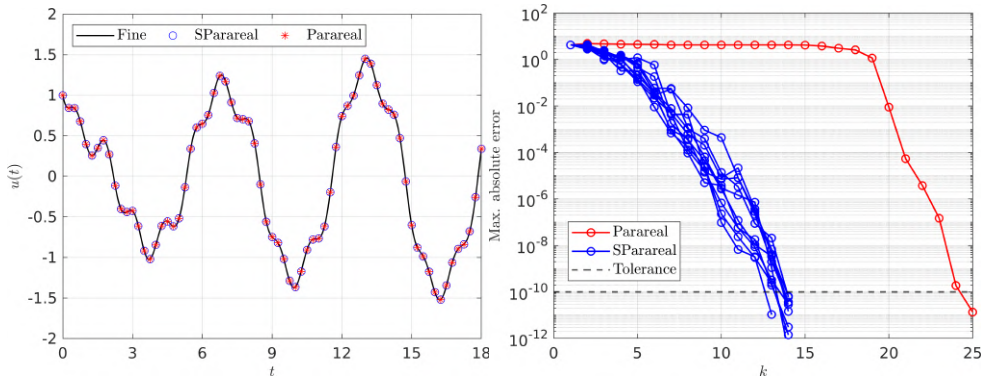
In this section, we compare the numerical performance of Parareal and SParareal on nonlinear ODE systems of increasing complexity. We fix $\mathcal{F}_{\Delta T} = \mathcal{G}_{\Delta T} = \text{RK4}$ for all experiments and let $N_{\mathcal{F}}$ and $N_{\mathcal{G}}$ denote the number of time steps each solver uses over $[t_0, T]$, respectively. Due to the stochastic nature of solutions from SParareal, we will quantify performance by estimating the distributions of k_s for each sampling rule over a number of independent simulations, comparing these results to the deterministic value k obtained by Parareal. We will also measure the accuracy of the stochastic solutions against those obtained serially with $\mathcal{F}_{\Delta T}$. At the time of writing only a limited number of processors were available for these experiments, hence the majority of the results in this section are based not on calculating wallclock runtimes but on comparing the iteration counts k and k_s—which are independent of the number of processors used[5]. Additional results for two further test problems can be found in Appendix C.

3.3.1 Scalar nonlinear equation

First, we consider the nonlinear nonautonomous scalar ODE

$$\frac{\mathrm{d}u}{\mathrm{d}t} = \sin(u)\cos(u) - 2u + e^{-t/100}\sin(5t) + \ln(1+t)\cos(t), \tag{3.10}$$

[5]We obtained access to a larger number of processors following the publication of this work and so we have retrospectively run some experiments to examine the wallclock time and speedup generated by SParareal in Section 3.3.1.



(a) (b)

Figure 3.7: (a) Numerical solution of (3.10) over $[0, 18]$ using $\mathcal{F}_{\Delta T}$ serially (black), a single realisation of SParareal (blue), and Parareal (red). Note that only a subset of the fine times steps of the SParareal and Parareal solutions are shown for clarity. (b) Errors at successive iterations of Parareal (red line) and ten independent realisations of SParareal (blue lines). Horizontal dashed black line represents the stopping tolerance $\varepsilon = 10^{-10}$. Note that both panels use SParareal with sampling rule 1 and $M = 3$.

with initial value $u(0) = 1$ (Chartier and Philippe, 1993). We discretise the time interval $t \in [0, 100]$ using $N = 40$ time slices and $N_{\mathcal{G}} = 100$ coarse and $N_{\mathcal{F}} = 8000$ fine time steps, respectively. Numerical solutions to (3.10) are shown on the interval $[0, 18]$ in Figure 3.7(a) where (deterministically) Parareal locates a solution in $k = 25$ iterations using error tolerance $\varepsilon = 10^{-10}$. SParareal converges in a varying number of iterations k_s, with $\mathbb{P}(k_s < k) = 1$, see Figure 3.7(b) for the convergence of ten independent simulations using $M = 3$. From this plot, we can see that by taking just three samples, SParareal reduces the number of iterations by almost a factor of two—from 25 to approximately 13 or 14. In Figure 3.8, we can see how this directly translates to increased speedup[6], where we observe SParareal locating a solution between 2.25–3× faster (than the serial $\mathcal{F}_{\Delta T}$ solver) and Parareal only 1.4× faster. Also shown are the speedup results when taking two or four samples and the corresponding theoretical bounds on speedup derived in Section 3.2.3. In Table 3.1, we provide a breakdown of the SParareal runtimes for a few simulations in the $M = 4$ experiment. They show that over 90% of the runtime is dedicated to carrying out fine solves whilst the coarse solves and optimal sample selection account for less than 0.03% of total runtime. Notice that the overheads are quite large because we carry out a lot of solution storage and diagnostic tracking (which are unnecessary in an optimised simulation) that are all serial computations. These results highlight that speedup continually increases as more samples are taken, at a cost of using significantly more processors (which is detrimental to parallel efficiency).

[6]Note that to generate the speedup results, $N_{\mathcal{F}}$ was increased by a factor of 10^4 so that the ratio $T_{\mathcal{G}}/T_{\mathcal{F}}$ was sufficiently small as to return meaningful speedup results.

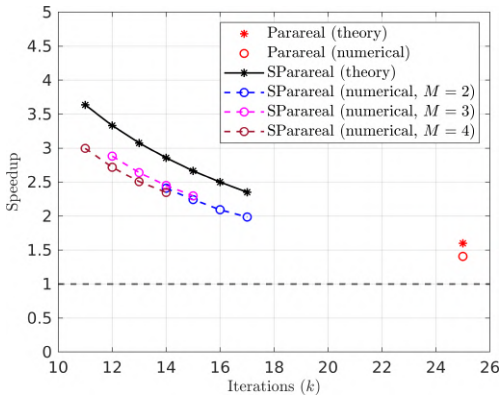
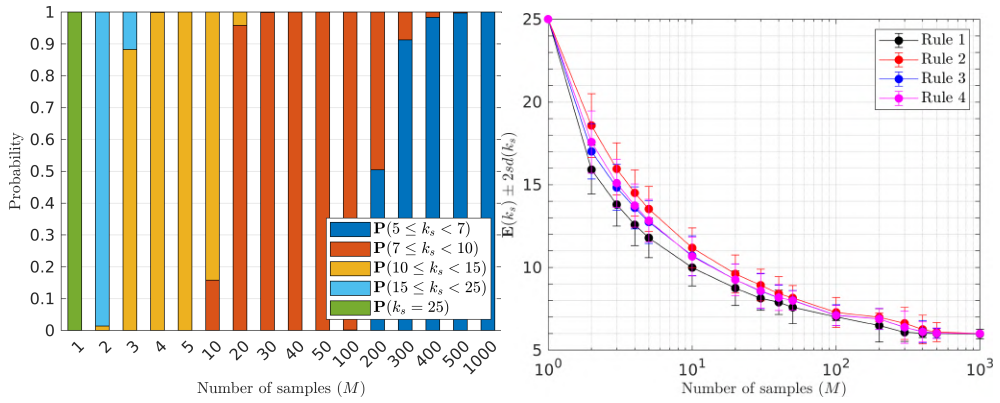


Figure 3.8: Speedup against iterations k, obtained when solving (3.10). Experiments (with sampling rule 1) are run over 50 independent simulations showing mean speedup with $M = 2$ (blue), $M = 3$ (purple), and $M = 4$ (brown) samples. Speedup for Parareal is also shown (red circle), calculated by averaging over 10 independent simulations. Corresponding theoretical results (red star and solid black line) were calculated using (2.12) and (3.8) for Parareal and SParareal, respectively.

Comparing the performance of SParareal directly with Parareal may be slightly unfair due to the fact that SParareal makes use of $\mathcal{O}(MN)$ processors while Parareal has access only to N processors. In the $M = 2$ experiment in Figure 3.8, SParareal uses up to $NM = 80$ processors to calculate a solution and so we therefore run an additional experiment in which Parareal has access to $N = 80$ processors to see how performance changes. In this experiment, Parareal took $k = 64$ iterations to locate a solution resulting in a numerical speedup of only $1.09\times$ (compared to $1.4\times$ in the $N = 40$ simulation). This dramatic increase in k is most likely due to the fact that the coarse solver is now taking one time step per time slice rather than the two steps per slice it took in the $N = 40$ simulation (recall the number of time slices doubled to 80). In this particular experiment, using more processors in Parareal actually has a detrimental impact on speedup (compared to SParareal as well), however, this is

Table 3.1: Wallclock time breakdown of four ($k = 11, 12, 13, 14$) SParareal simulations from the $M = 4$ experiment presented in Figure 3.8. For each, we show the total time taken doing coarse solves, fine solves, optimal sample selection, and overheads. All timings are measured in seconds and the percentage of the total SParareal runtime T_{SPara} is shown in brackets for each quantity.

k	Coarse solves	Fine solves	Optimal sample selection	Overheads	T_{SPara}
11	5.20E−3 (0.02%)	24.79 (92.54%)	3.00E3 (0.01%)	1.99 (7.43%)	26.79
12	5.30E−3 (0.02%)	27.08 (95.22%)	3.10E3 (0.01%)	1.35 (4.76%)	28.45
13	5.30E−3 (0.02%)	29.28 (95.90%)	3.10E3 (0.01%)	1.30 (4.07%)	30.59
14	5.60E−3 (0.02%)	31.62 (90.09%)	3.30E3 (0.01%)	3.47 (9.88%)	35.09



(a) **(b)**

Figure 3.9: (a) Estimated discrete distributions of k_s as a function of M for sampling rule 1. (b) Estimated expectation of k_s as a function of M, calculated using estimated distributions of k_s for each sampling rule with error bars representing $\pm$ two standard deviations $sd(k_s)$. Distributions in both panels are estimated by simulating 2000 independent realisations of SParareal for each M.

certainly not generic behaviour and would require further investigation for alternative systems and time step configurations.

We know that for $M > 1$, SParareal generates stochastic solutions that converge in a varying number of iterations k_s. In order to accurately compare k with the discrete random variable k_s, we run 2000 independent simulations of SParareal to estimate the distribution of k_s for a given M. Upon estimating these distributions, it was found that $\mathbb{P}(k_s < 25) = 1$ for each of the four sampling rules (for all $M > 1$), meaning that

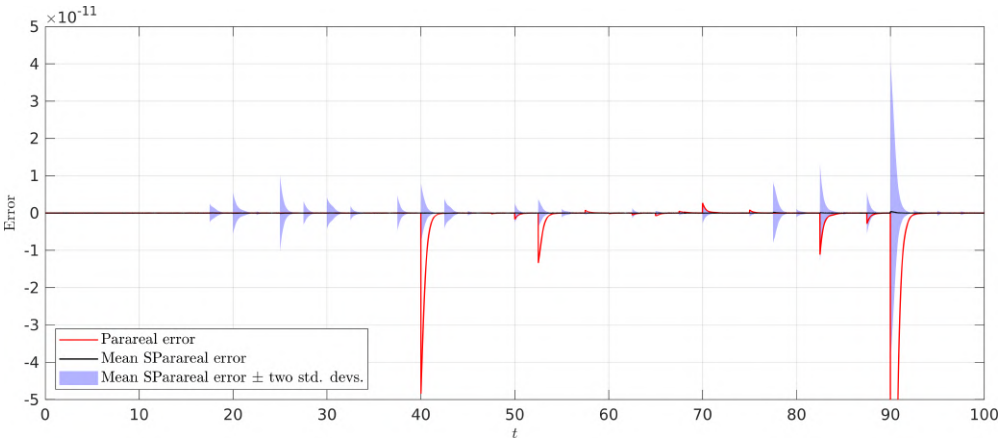


Figure 3.10: Errors of Parareal (red) and mean SParareal (black) solutions against the serial $\mathcal{F}_{\Delta T}$ solution over time. The mean error is obtained by running 2000 independent realisations of SParareal with sampling rule 2 and $M = 4$—the confidence interval representing the mean $\pm$ two standard deviations is shown in light blue.

we can beat Parareal with probability approaching one. The estimated distributions of k_s, using sampling rule 1 (the other rules perform similarly), as a function of M are given in Figure 3.9(a). The stacked bars represent the estimated discrete probability of a simulation converging in a given number of iterations. The results show SParareal converging in just five iterations in the best case—demonstrating SParareal has the potential to yield significant parallel speedup, although this would require $\mathcal{O}(10^3)$ processors to achieve! Clearly using this many processors to achieve (up to) $8\times$ speedup would be overkill for such a simple IVP but it demonstrates that SParareal works as expected. Figure 3.9(b) emphasises the power of the stochastic method, showing that the estimated expected value $E(k_s)$ decreases as M increases, with the estimated standard deviation $sd(k_s) = \sqrt{\sum_{k=1}^{N}(k - E(k_s))^2 \mathbb{P}(k_s = k)}$ decreasing too. The improved performance of SParareal as M increases reflects what was discussed in Section 3.2.4. We also ran a single realisation of SParareal with $M = 10^6$, observing that SParareal converged in four iterations (result not shown), confirming that k_s continues to decrease for increasing M. By looking at Figure 3.9(b), we see that sampling rule 1 yields the lowest expected values of k_s for small values of M, with all sampling rules performing similarly for large M.

To verify the accuracy of the stochastic solutions, we plot the difference between the mean of 2000 independent realisations of SParareal and the serially calculated $\mathcal{F}_{\Delta T}$ solution in Figure 3.10. Also shown is the confidence interval given by two standard deviations of the stochastic solutions (which is at most $\mathcal{O}(10^{-11})$) and the error generated by Parareal. Accuracy is maintained with respect to the fine solution across the time interval, even more so than the Parareal solution. See Appendix C.1 for numerical results of SParareal applied to a *stiff* scalar nonlinear ODE. In that case, the stiffness of the equation demands a higher value of M to improve k_s—something we observe for the Brusselator in the next section.

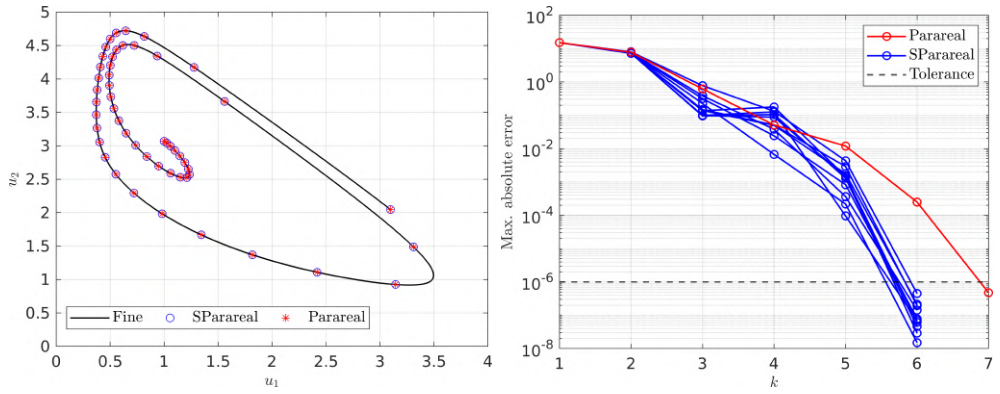
3.3.2 The Brusselator system

Next, consider the Brusselator system

$$\frac{\mathrm{d}u_1}{\mathrm{d}t} = A + u_1^2 u_2 - (B+1)u_1, \tag{3.11a}$$

$$\frac{\mathrm{d}u_2}{\mathrm{d}t} = Bu_1 - u_1^2 u_2, \tag{3.11b}$$

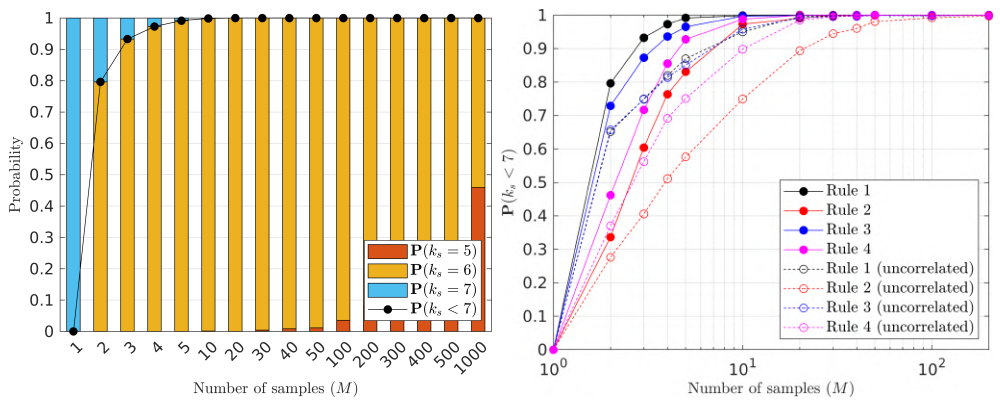
a pair of stiff nonlinear ODEs that model an auto-catalytic chemical reaction (Lefever and Nicolis, 1971). Using parameters $(A, B) = (1, 3)$, trajectories of the system exhibit oscillatory behaviour in phase space, approaching a limit cycle (as $t \to \infty$) that contains the unstable fixed point $(1, 3)^\intercal$. Now that $d > 1$, we use bivariate distributions to sample the initial values—meaning we can compare the effects of including or excluding the correlations between variables. System (3.11) is solved



(a) **(b)**

Figure 3.11: (a) Numerical solution of system (3.11) in the phase plane using $\mathcal{F}_{\Delta T}$ (black), Parareal (red), and SParareal (blue). Note that solutions from Parareal and SParareal are shown at times $\boldsymbol{t}$ for clarity. (b) Errors at successive iterations of Parareal (red line) and ten independent realisations of SParareal (blue lines). Horizontal dashed black line represents the tolerance $\varepsilon = 10^{-6}$. Both panels run SParareal with sampling rule 1 and $M = 10$.

using initial value $\boldsymbol{u}(0) = (1, 3.07)^\intercal$ over time interval $t \in [0, 15.3]$ with $N = 25$, $N_{\mathcal{G}} = 25$, and $N_{\mathcal{F}} = 2500$ (Trefethen et al., 2017). In Figure 3.11(a), we plot numerical solutions to system (3.11) in the phase plane and the errors at successive iterations of ten SParareal simulations in Figure 3.11(b). With these parameters and a tolerance of $\varepsilon = 10^{-6}$, Parareal takes $k = 7$ iterations to stop and return a numerical solution whilst SParareal takes $k_s = 6$.



(a) **(b)**

Figure 3.12: (a) Estimated discrete probabilities of k_s as a function of M for sampling rule 1. (b) Estimated probability that the convergence rate k_s is smaller than $k = 7$ as a function of M for the sampling rules with (solid lines) and without (dashed lines) correlations. Distributions were estimated by simulating 2000 independent realisations of SParareal for each M.

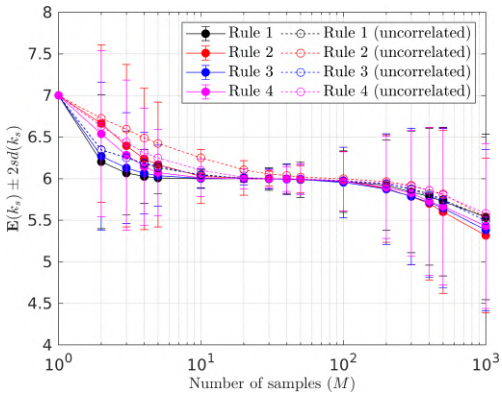


Figure 3.13: Estimated expectation of k_s as a function of M, calculated using estimated distributions of k_s for each sampling rule with error bars representing $\pm$ two standard deviations $sd(k_s)$ shown for correlated (solid lines) sampling rules. Uncorrelated sampling rules are shown with dashed lines without error bars. Distributions are estimated by simulating 2000 independent realisations of SParareal for each M.

The estimated distributions of k_s for sampling rule 1 are given in Figure 3.12(a). Even though Parareal takes just $k = 7$ iterations to stop, we observe that SParareal can still reach the desired tolerance in 5 or 6 iterations—albeit requiring larger values of M. We believe this is due to the stiffness of the system and poor accuracy of the $\mathcal{G}_{\Delta T}$ solver—results presented for the stiff ODE in Appendix C.1 appear to confirm this. The solid lines in Figure 3.12(b) show that, using sampling rules 1 or 3, SParareal only requires $M \approx 10$ to beat Parareal almost certainly, i.e. to guarantee

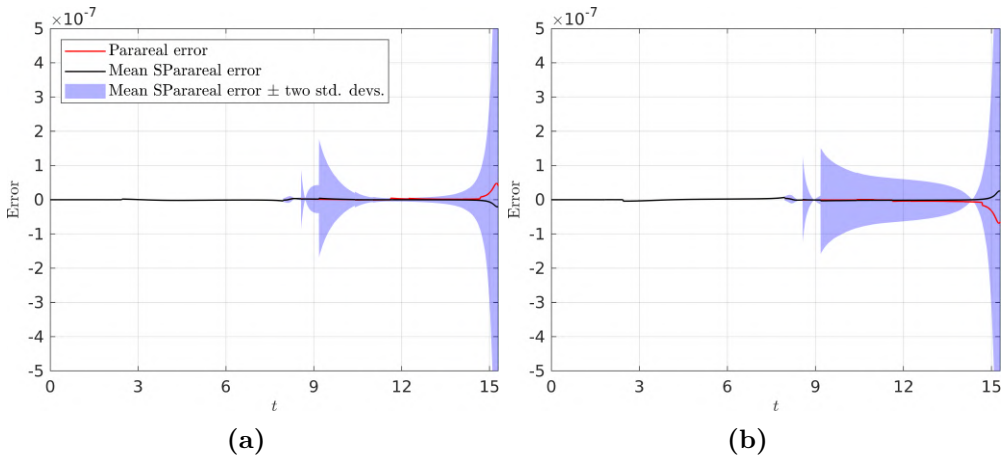


Figure 3.14: Errors of Parareal (red) and mean SParareal (black) solutions against the $\mathcal{F}_{\Delta T}$ solution. The mean error is obtained by running 2000 independent realisations of SParareal with sampling rule 4 and $M = 200$—the confidence interval representing the mean $\pm$ two standard deviations is shown in light blue. Panel (a) displays errors for the u_1 component of the solution whilst (b) displays the u_2 component.

that $\mathbb{P}(k_s < 7) \to 1$. Sampling rules 1 and 3 outperform 2 and 4 in this particular system. Note, however, the stark decrease in performance if instead uncorrelated samples are generated within SParareal (dashed lines). This demonstrates the importance of accounting for the dependence between variables in nonlinear systems such as (3.11). In Figure 3.13 we report the expected value of k_s as a function of M for each of the sampling rules. These results further suggest (in addition to Figure 3.12(a)) that larger values of M are required to reduce $E(k_s)$ even further for this stiff system. Observe, in Figure 3.14, how the mean SParareal solutions still maintain equivalent or better accuracy than the Parareal solutions, with standard deviations at most $\mathcal{O}(10^{-6})$.

In Appendix C.2, we apply SParareal to a non-stiff two-dimensional nonlinear system and observe that less sampling is required to accelerate convergence compared to the Brusselator system.

3.3.3 The Lorenz63 system

Finally, we consider the Lorenz63 system

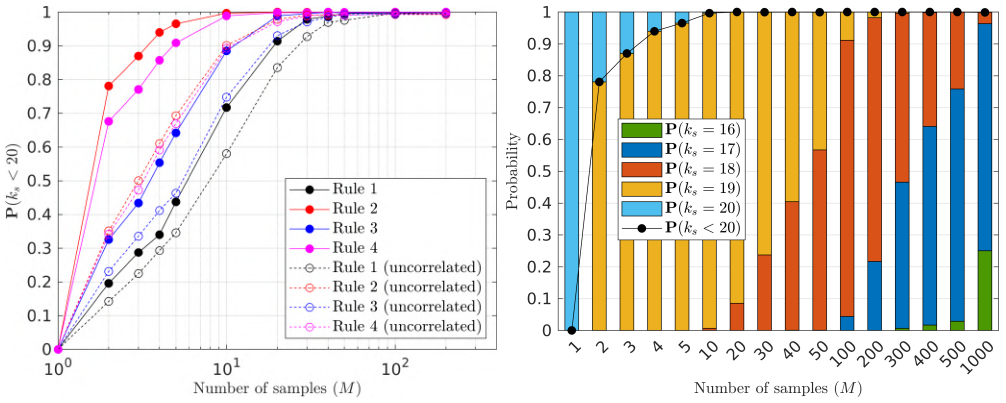
$$\frac{\mathrm{d}u_1}{\mathrm{d}t} = \gamma_1(u_2 - u_1), \tag{3.12a}$$

$$\frac{\mathrm{d}u_2}{\mathrm{d}t} = \gamma_2 u_1 - u_1 u_3 - u_2, \tag{3.12b}$$

$$\frac{\mathrm{d}u_3}{\mathrm{d}t} = u_1 u_2 - \gamma_3 u_3, \tag{3.12c}$$

a simplified model for weather prediction developed by Lorenz (1963). With the parameters $(\gamma_1, \gamma_2, \gamma_3) = (10, 28, 8/3)$, (3.12) exhibits chaotic behaviour where trajectories with initial values close to one another diverge exponentially. This will test the robustness of SParareal, as small numerical differences between initial values will mean that errors can grow rapidly as time progresses. We solve (3.12) using initial value $\boldsymbol{u}(0) = (-15, -15, 20)^\intercal$ over the interval $[0, 18]$, discretised using $N = 50$ time slices and time steps $N_{\mathcal{G}} = 250$ and $N_{\mathcal{F}} = 18,750$. With a tolerance of $\varepsilon = 10^{-8}$, Parareal takes $k = 20$ iterations to converge.

Running SParareal to compare the performance of the sampling rules, we see again in Figure 3.15(a) that taking correlated samples is much more efficient than not and that only $M \approx 10$ samples are required to beat Parareal with probability one. For the chaotic trajectories generated by (3.12), sampling close to the PC, rules 2 and 4, yields superior performance compared to rules 1 and 3 for small values of M. Figure 3.15(b) displays estimated distributions for varying M using sampling rule 2—yielding a best $k_s = 16$ for approximately 25% of runs with $M = 1000$. Figure 3.16 displays $E(k_s)$ against M, confirming that generating correlated samples close to the PC solutions (sampling rules 2 and 4) yield the lowest expected values of k_s—although it takes a large number of samples to reduce the iteration count by just



(a) **(b)**

Figure 3.15: (a) Estimated probability that the convergence rate k_s is smaller than $k = 20$ as a function of M for the sampling rules with (solid lines) and without (dashed lines) correlations. Distributions were estimated by simulating 2000 independent realisations of SParareal for each M. (b) Estimated discrete probabilities of k_s as a function of M for sampling rule 2.

a few. As before, we plot the absolute errors between the mean SParareal solution and the fine solution for completeness—see Figure 3.17. Accuracy is again maintained with respect to the Parareal solution, even as the errors grow with increasing time (which is expected in a chaotic system). Keeping the errors (relatively) small is challenging for PinT methods when solving chaotic systems and so these results demonstrate the robustness of SParareal and that the sampling and propagation process is not impeded by the exponential divergence of trajectories.

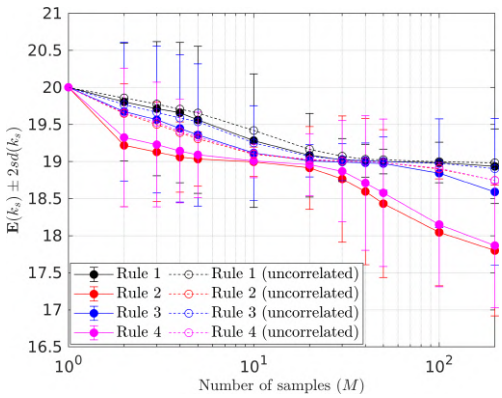


Figure 3.16: Estimated expectation of k_s as a function of M, calculated using estimated distributions of k_s for each sampling rule with error bars representing $\pm$ two standard deviations $sd(k_s)$ shown for correlated (solid lines) sampling rules. Uncorrelated sampling rules are shown with dashed lines without error bars. Distributions are estimated by simulating 2000 independent realisations of SParareal for each M.

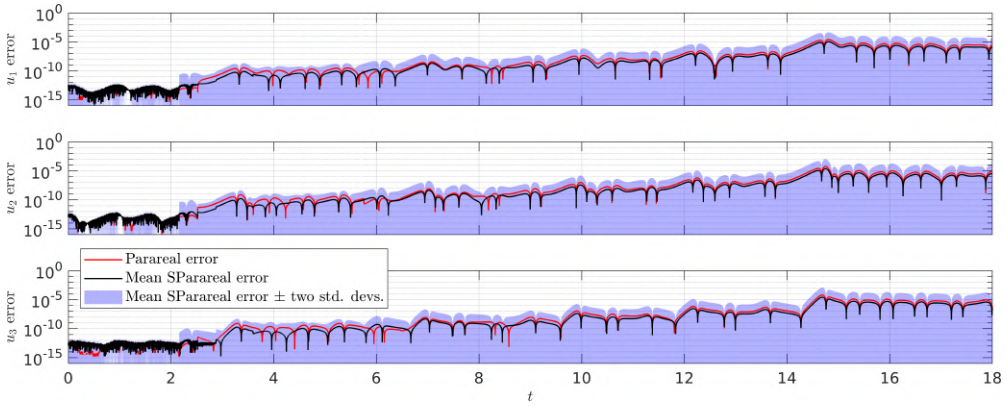


Figure 3.17: Absolute errors of Parareal (red) and mean SParareal (black) solutions against the $\mathcal{F}_{\Delta T}$ solution in each component: u_1 (top), u_2 (middle), and u_3 (bottom). The mean error is obtained by running 2000 independent realisations of SParareal with sampling rule 2 with $M = 500$—the confidence interval representing the mean ± two standard deviations is shown in light blue.

3.4 Discussion and further work

In this chapter, we have extended the Parareal algorithm using probabilistic methods to develop a (sampling-based) stochastic Parareal algorithm for solving systems of ODEs in a time-parallel manner. Instead of passing deterministically calculated solution states into Parareal's PC (2.9c), SParareal selects "more accurate" values from a randomly sampled set, in each time slice, to converge in fewer iterations. In Section 3.3, we compared performance against the deterministic Parareal algorithm on several low-dimensional ODE systems of increasing complexity by calculating the distributions of the iteration count (upon multiple independent realisations of SParareal) with increasing numbers of random samples M. By taking just $M \approx 10$ (correlated) samples, the estimated probability of converging sooner than Parareal approached one in all test cases. Similarly, we observed numerical convergence toward the fine (exact) solution with accuracy of similar order to Parareal and obtained a distribution over the ODE solution upon multiple realisations of the algorithm.

The probability that SParareal converges faster than Parareal depends on a number of factors: the complexity/size of the problem being solved, the number of time time slices (N), the accuracy of the coarse integrator ($\mathcal{G}_{\Delta T}$), the number of random samples (M), and the type of sampling rule in use. Sampling rules 1 and 3 (sampling close to the fine solution states) outperformed rules 2 and 4 (sampling close to the PC states) for the ODE systems in Section 3.3.1, Section 3.3.2, and Appendix C.1. The reverse was true, however, for the systems in Section 3.3.3 and Appendix C.2, making it difficult to determine an optimal rule for a general ODE system. To overcome having to choose a particular sampling rule, one could linearly combine different rules or even sample from multiple rules simultaneously.

We would suggest sampling from probability distributions with infinite support, i.e. the Gaussians (rules 1 and 2), so that samples can be taken anywhere in $\mathbb{R}^d$ with non-zero probability. Having finite support may have created difficulty for the uniform marginal t-copulas (rules 3 and 4) because samples could only be taken in a finite hyperrectangle in $\mathbb{R}^d$—problematic if the exact solution state were to lay outside of this space.

When solving stiff ODEs (see Section 3.3.2 and Appendix C.1), results indicated that SParareal demanded increasingly high sampling to converge sooner than Parareal than for non-stiff systems. For example, we observe that when taking $M = 100$ samples in the non-stiff scalar ODE in Figure 3.9, the expected number of iterations decreases from 25 to 7 whereas for the stiff scalar ODE in Appendix C.1, the number only drops from 8 to 6. A similar observation can be made in the two-dimensional test cases in Section 3.3.2 (stiff) and Appendix C.2 (non-stiff). These results exemplify the role that system complexity, e.g. stiffness or chaos, plays in the performance of both algorithms. In Appendix C.1, SParareal was also shown to perform more efficiently for problems that Parareal itself struggles with, i.e. cases in which the accuracy of the coarse integrator $\mathcal{G}_{\Delta T}$ is poor. In Appendix C.2 it was also observed that, for low sample numbers ($M = 2$), SParareal actually converged in one more iteration than Parareal in less than 2.5% of cases. This suggests there may be minimum number of samples required to beat Parareal in some situations—something to be investigated with further experimentation.

The curse of dimensionality also plays a stark role in the performance of SParareal. It should be obvious that as the dimension of the system d increases, the effectiveness of the sampling (for fixed M) will decrease. In other words, SParareal will require exponentially increasing numbers of samples as d increases—this can be seen in Section 3.3.3 where a large M is required to reduce k_s by even a few iterations (where d is only equal to three). This is problematic as the number of processors scales directly with M and so for very high dimensional systems (think about PDEs discretised with a large number of spatial points), the number of processors required to solve in parallel (faster than Parareal) will be exceedingly large. We are unsure how to avoid this issue but assessing performance on problems with larger d certainly warrants further investigation.

In addition to accelerated convergence, we were able to generate a distribution of stochastic solutions to the IVPs tested using SParareal. While the distributions generated by such ensembles may not have an explicit mathematical interpretation (i.e. the uncertainty does not represent numerical uncertainty generated by the fine or coarse solvers), the individual stochastic trajectories are indeed accurate with respect to the fine solution. Therefore, each one can be interpreted as a solution to the IVP and their stochastic nature is useful in that they may reveal additional information about the system dynamics we may not see with a single deterministic

trajectory. In Chapter 4, we derive explicit error bounds for the solutions obtained from SParareal, showing that perturbations of small enough "size" are required to obtain accurate solutions—a condition that the sampling rules automatically satisfy.

In summary, we have demonstrated that sampling-based methods and additional processors can, for low-dimensional ODEs at least, be used to accelerate the convergence of Parareal. There exist a few avenues for possible improvement and generalisation. Firstly, if information is known about the behaviour of the solution prior to simulation (e.g. if it is bounded or perhaps non-negative) then one could construct a sampling rule to satisfy these requirements. Secondly, one could use an alternative stopping criterion that halts SParareal when the largest standard deviation of the distribution (in each time slice) is below ε. This would indicate that solutions are no longer being improved significantly and so SParareal should stop, potentially saving a costly iteration or two. In terms of the solvers $\mathcal{F}_{\Delta T}$ and $\mathcal{G}_{\Delta T}$, one could test whether the sampling-based solvers developed by Conrad et al. (2017) could be used within Parareal—although some adaptation would be required to ensure that perturbations are not amplified by the PC scheme, leading to non-convergence. Finally, it would advantageous to avoid wasting the valuable information about the system gained from the ensemble of fine propagated trajectories and see whether it can be harnessed to extract further numerical speedup. This is something we will explore in Chapter 5 when we develop GParareal, adopting a more Bayesian approach to the problem by using *all* of the available solution data (at all iterations and time steps) to inform the Parareal PC update.

Chapter 4

SParareal II: error bound analysis

Overview

Deriving rigorous error bounds for SParareal (and other PinT methods in general) is important in demonstrating that numerical solutions obtained in parallel are meaningful, accurate, and that they can be compared to one another (Gander et al., 2022). In this chapter, we extend the qualitative discussion of numerical convergence from Section 3.2.4 by making use of existing convergence analysis on Parareal (recall Section 2.2.4) and the sampling-based ODE solver proposed by Lie et al. (2019). We derive explicit mean-square error bounds for SParareal applied to nonlinear systems of ODEs (over a finite time interval), using two different types of perturbation: *state-independent* and *state-dependent*.

We begin in Section 4.1 by re-defining the SParareal scheme in such a way that allows us to carry out our convergence analysis. The reason for this is that definition (3.1) is very difficult to manipulate and analyse, therefore we need to make a simplification to derive rigorous error bounds. We then recall the sampling rules first introduced in Section 3.2.2. In Section 4.2, we outline the assumptions on the fine and coarse integrators required to derive the error bounds. We first consider the *state-independent* setting, in which the random perturbations do not depend on solution states at any time step or iteration and are assumed to have bounded absolute moments. In this setting, we derive our main result (Theorem 4.7), a superlinear bound on the mean-square error that depends on both the time step and iteration of SParareal. Using this result, we can maximise the error over time to derive a linear error bound (Corollary 4.8). In the *state-dependent* setting, we allow the perturbations to depend on solution states up to the current time step and iteration, i.e. the known coarse and fine solution information. This will allow us to analyse the convergence of the sampling rules proposed in Chapter 3. We derive

linear bounds (Corollaries 4.14 and 4.15) in this setting. Following this, we verify all of the theoretical bounds by comparing them to numerical errors generated when solving a linear system of ODEs and a nonlinear scalar ODE in Section 4.3. We conclude with some brief remarks on the significance of these results and discuss the limitations which could warrant further study in Section 4.4.

Throughout this chapter, we denote variables $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{R}^d$ as d-dimensional real-valued vectors, the component-wise absolute value of a vector as $|\boldsymbol{u}| = (|u_1|, \dots, |u_d|)^\intercal$, and the Hadamard (component-wise) product as $\boldsymbol{u} \circ \boldsymbol{v} = (u_1 v_1, \dots, u_d v_d)^\intercal$. We let $\boldsymbol{u}^2$ correspond to component-wise squaring, i.e. $\boldsymbol{u}^2 = \boldsymbol{u} \circ \boldsymbol{u}$, and $\|\boldsymbol{u}\|$ the infinity (or uniform) norm, i.e. $\|\boldsymbol{u}\| = \max_{i=1,\dots,d} |u_i|$. The d-dimensional vector of ones and the identity matrix will be written as $\mathbf{1}$ and $\mathbb{I}_d$, respectively, with non-negative constants denoted throughout by C_1, C_2,.... In Appendix D, we provide some technical results used to derive the aforementioned error bounds.

4.1 Re-defining SParareal

In this section, we provide an alternative (but equivalent) definition of the SParareal scheme needed to carry out the error bound analysis in Section 4.2. Note that throughout this chapter we will be considering autonomous IVPs, i.e. $\boldsymbol{f}(t, \boldsymbol{u}(t)) := \boldsymbol{f}(\boldsymbol{u}(t))$ in (2.1), where everything that follows should extend naturally to the nonautonomous case.

4.1.1 The alternative scheme

The intuition behind SParareal is to perturb the solution states $\boldsymbol{U}_n^k$ in the classic Parareal scheme, i.e. the PC (2.9c), with some additive noise to reduce the number of iterations k taken until the stopping tolerance (2.10) is met. The original definition (3.1) was difficult to analyse because it allowed for the taking of M random samples and it was unclear how to bound the error of this scheme. We propose the following alternative definition of SParareal, inspired by the form of the sampling-based ODE solver proposed by Conrad et al. (2017), whereby only *one* random sample is now taken.

Definition 4.1 (Alternative SParareal)**.** For the two numerical flow maps $\mathcal{F}_{\Delta T}$ and $\mathcal{G}_{\Delta T}$, described in Section 2.2, the (alternative) SParareal scheme is given by

$$\boldsymbol{U}_0^0 = \boldsymbol{u}^0, \tag{4.1a}$$

$$\boldsymbol{U}_{n+1}^0 = \mathcal{G}_{\Delta T}(\boldsymbol{U}_n^0), \qquad 0 \leqslant n \leqslant N-1, \tag{4.1b}$$

$$\boldsymbol{U}_{n+1}^1 = \mathcal{G}_{\Delta T}(\boldsymbol{U}_n^1) + \mathcal{F}_{\Delta T}(\boldsymbol{U}_n^0) - \mathcal{G}_{\Delta T}(\boldsymbol{U}_n^0), \qquad 0 \leqslant n \leqslant N-1, \tag{4.1c}$$

$$\boldsymbol{U}_{n+1}^{k+1} = \mathcal{G}_{\Delta T}(\boldsymbol{U}_n^{k+1}) + \mathcal{F}_{\Delta T}(\boldsymbol{U}_n^k) - \mathcal{G}_{\Delta T}(\boldsymbol{U}_n^k) + \boldsymbol{\xi}_n^k(\boldsymbol{U}_n^k), \qquad 1 \leqslant k \leqslant n \leqslant N-1, \tag{4.1d}$$

where $\boldsymbol{\xi}_n^k(\boldsymbol{U}_n^k)$ are (possibly state-dependent) random variables. Note that we define $\boldsymbol{\xi}_n^k(\boldsymbol{U}_n^k) \equiv \mathbf{0}$ when $n = k$.

The first three stages of the scheme (4.1a)–(4.1c) are identical to the 'zeroth' and first iteration of Parareal. The exact initial condition (4.1a) is propagated forward in time using the coarse solver (4.1b), then there is a first pass of the PC (4.1c). Following this, the stochastic iterations begin (4.1d), whereby a random perturbation, i.e. a single draw from the random variable $\boldsymbol{\xi}_n^k(\boldsymbol{U}_n^k)$, is added to the PC solution. Notice that no random perturbation is added when $n = k$ to ensure that SParareal returns the exact solution up to time t_k after k iterations, just as Parareal does (recall Section 2.2.3). The reason the random perturbations are only included from iteration $k \geqslant 1$ onward is because $\boldsymbol{\xi}_n^k(\boldsymbol{U}_n^k)$ may depend on solution information from iteration $k-1$. Note that the Parareal scheme (2.9) can be recovered by setting $\boldsymbol{\xi}_n^k(\boldsymbol{U}_n^k) \equiv \mathbf{0}$ for $n \geqslant k$ in (4.1d).

The scheme in (4.1) looks slightly different to the one presented in (3.5), where random perturbations were instead incorporated via random variables (denoted by the optimal sample $\hat{\boldsymbol{\alpha}}_n^k$) in the correction term. The different state-dependent forms that $\boldsymbol{\alpha}_n^k$ can explicitly take are defined through the sampling rules in Section 4.1.2. Also note that $\boldsymbol{\alpha}_n^k = \boldsymbol{U}_n^k$ when $n = k$, which is equivalent to the condition that $\boldsymbol{\xi}_n^k(\boldsymbol{U}_n^k) \equiv \mathbf{0}$ when $n = k$ in (4.1d). The scheme defined by (3.5d) was designed so that $M \geqslant 1$ samples could be drawn from each of the random variables $\boldsymbol{\alpha}_n^k$ to increase the probability of locating the exact solution state $\boldsymbol{U}_n$ in fewer iterations. From the sets of samples generated at each t_n, all having been propagated in parallel using $\mathcal{F}_{\Delta T}$ and $\mathcal{G}_{\Delta T}$, those generating the most continuous $\mathcal{F}_{\Delta T}$ trajectory across $[t_0, t_N]$ were then chosen as the "best" perturbations $\hat{\boldsymbol{\alpha}}_n^k$. Numerical experiments illustrated that increasing M led to further and further reductions in k, albeit at the cost of requiring more processors, specifically $\mathcal{O}(MN)$ in SParareal vs $\mathcal{O}(N)$ in Parareal.

To enable us to carry out the convergence analysis, we move the random perturbations in (3.5d) outside the correction term, and express them using $\boldsymbol{\xi}_n^k(\boldsymbol{U}_n^k)$ in (4.1d). This new scheme is equivalent to the old scheme in the case where one sample ($M = 1$) is drawn at each t_n[1]. The following error bounds are derived assuming one sample is drawn from each $\boldsymbol{\xi}_n^k(\boldsymbol{U}_n^k)$—the M sample case from Chapter 3 is much more complex and out of the scope of the present work.

4.1.2 Sampling rules

The sampling rules presented in Section 3.2.2 describe the probability distributions that $\boldsymbol{\alpha}_n^k$ follow in the SParareal algorithm, see Table 4.1 for a summary. These distributions were designed to vary with both iteration k and time step n, so that as

[1]Note that in the new scheme (4.1), $M = 1$ corresponds to drawing a random sample, whereas $M = 1$ in the old scheme (3.5) corresponded to taking the (deterministic) PC state and simply running Parareal.

Table 4.1: Sampling rules that the random variables $\boldsymbol{\alpha}_n^k$ follow. The quantities $\boldsymbol{z}_n^k \sim \mathcal{N}(\mathbf{0}, \mathbb{I}_d)$ and $\boldsymbol{w}_n^k \sim \mathcal{U}([0,1]^d)$ are d-dimensional Gaussian and uniform random vectors, respectively, whilst $\boldsymbol{\sigma}_n^k = |\mathcal{G}_{\Delta T}(\boldsymbol{U}_{n-1}^k) - \mathcal{G}_{\Delta T}(\boldsymbol{U}_{n-1}^{k-1})|$.

Sampling rule	Distribution	$\boldsymbol{\alpha}_n^k$
1	Gaussian	$\mathcal{F}_{\Delta T}(\boldsymbol{U}_{n-1}^{k-1}) + (\boldsymbol{\sigma}_n^k \circ \boldsymbol{z}_n^k)$
2		$\boldsymbol{U}_n^k + (\boldsymbol{\sigma}_n^k \circ \boldsymbol{z}_n^k)$
3	Uniform	$\mathcal{F}_{\Delta T}(\boldsymbol{U}_{n-1}^{k-1}) + \big(\sqrt{3}\boldsymbol{\sigma}_n^k \circ (2\boldsymbol{w}_n^k - \mathbf{1})\big)$
4		$\boldsymbol{U}_n^k + \big(\sqrt{3}\boldsymbol{\sigma}_n^k \circ (2\boldsymbol{w}_n^k - \mathbf{1})\big)$

the solution states $\boldsymbol{U}_n^k$ get closer to $\boldsymbol{U}_n$, their variances would decrease—the benefit of this property will be highlighted in Section 4.3. These (state-dependent) rules were constructed to assess the performance of SPararеal when the perturbations had different distribution families, marginal means, or correlations. To derive error bounds for the sampling rules, we need to define $\boldsymbol{\xi}_n^k(\boldsymbol{U}_n^k)$ in terms of $\boldsymbol{\alpha}_n^k$. To do this, we simply equate (4.1d) and (3.5d) to find

$$\boldsymbol{\xi}_n^k(\boldsymbol{U}_n^k) = \big(\mathcal{F}_{\Delta T}(\boldsymbol{\alpha}_n^k) - \mathcal{G}_{\Delta T}(\boldsymbol{\alpha}_n^k)\big) - \big(\mathcal{F}_{\Delta T}(\boldsymbol{U}_n^k) - \mathcal{G}_{\Delta T}(\boldsymbol{U}_n^k)\big). \tag{4.2}$$

Recall that sampling rules 1 and 2 correspond to multivariate Gaussian perturbations with marginal means $\mathcal{F}_{\Delta T}(\boldsymbol{U}_{n-1}^{k-1})$ and $\boldsymbol{U}_n^k$, respectively, and marginal standard deviations $\boldsymbol{\sigma}_n^k = |\mathcal{G}_{\Delta T}(\boldsymbol{U}_{n-1}^k) - \mathcal{G}_{\Delta T}(\boldsymbol{U}_{n-1}^{k-1})|$. Sampling rules 3 and 4 correspond to perturbations following a multivariate uniform distribution with the same marginal means and standard deviations as rules 1 and 2, respectively. Note that in Section 3.2.2, we considered both correlated and uncorrelated random variables $\boldsymbol{\alpha}_n^k$ in our experiments, whereas here we carry out analysis only for the uncorrelated case.

4.2 Error bound analysis

In this section, we will be analysing the mean-square error

$$e_n^k := \mathbb{E}\big[\|\boldsymbol{u}(t_n) - \boldsymbol{U}_n^k\|^2\big], \tag{4.3}$$

between the exact solutions $\boldsymbol{u}(t_n)$ (equivalently $\boldsymbol{U}_n$) and the stochastic numerical solutions $\boldsymbol{U}_n^k$ located by SPararеal (4.1). We also define the maximal mean-square error (over time) at iteration k to be

$$\hat{e}^k := \max_{1 \leqslant n \leqslant N} \{e_n^k\}. \tag{4.4}$$

Specifically, we analyse the mean-square error e_n^k for the nonlinear (autonomous) system of ODEs in (2.1), first deriving superlinear (Theorem 4.7) and linear (Corollary 4.8) bounds using state-independent perturbations in SPararеal. Then, using these results, we derive linear bounds for the state-dependent sampling rules 2 and 4 (Corollary 4.14) and 1 and 3 (Corollary 4.15). In the following, we introduce some assumptions on the flow maps (Gander and Hairer, 2008) and perturbations (Lie et al., 2019) required to derive the error bounds.

Assumption 4.2 (Exact flow map $\mathcal{F}_{\Delta T}$)**.** The flow map $\mathcal{F}_{\Delta T}$ solves (2.1) exactly such that

$$\boldsymbol{u}(t_{n+1}) = \mathcal{F}_{\Delta T}(\boldsymbol{u}(t_n)). \tag{4.5}$$

This assumption is made for simplicity, since SPararеal is trying to locate the solution that would be obtained by running the fine solver serially, i.e. (2.3), in parallel. If instead we were to consider $\mathcal{F}_{\Delta T}$ to be a numerical method with some (very small) numerical error with respect to the exact solution, then the accuracy of $\mathcal{F}_{\Delta T}$ would provide a lower bound on the accuracy of the SPararеal scheme as a whole.

Assumption 4.3 (One-step coarse flow map $\mathcal{G}_{\Delta T}$)**.** The flow map $\mathcal{G}_{\Delta T}$ is a one-step numerical method with uniform local truncation error $\mathcal{O}(\Delta T^{p+1})$, for $p \geqslant 1$, such that

$$\mathcal{F}_{\Delta T}(\boldsymbol{u}) - \mathcal{G}_{\Delta T}(\boldsymbol{u}) = c_1(\boldsymbol{u})\Delta T^{p+1} + c_2(\boldsymbol{u})\Delta T^{p+2} + \dots, \tag{4.6}$$

for $\boldsymbol{u} \in \mathbb{R}^d$ and continuously differentiable functions $c_i(\boldsymbol{u})$. Taking the difference of (4.6) evaluated at states $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{R}^d$, then applying norms and the triangle inequality, we can write

$$\| \left(\mathcal{F}_{\Delta T}(\boldsymbol{u}) - \mathcal{G}_{\Delta T}(\boldsymbol{u})\right) - \left(\mathcal{F}_{\Delta T}(\boldsymbol{v}) - \mathcal{G}_{\Delta T}(\boldsymbol{v})\right) \| \leqslant C_1 \Delta T^{p+1} \|\boldsymbol{u} - \boldsymbol{v}\|, \tag{4.7}$$

where $C_1 > 0$ is the Lipschitz constant for c_1 and we absorb terms $\mathcal{O}(\Delta T^{p+2})$ into C_1.

Assumption 4.4 (Lipschitz coarse flow $\mathcal{G}_{\Delta T}$)**.** The flow map $\mathcal{G}_{\Delta T}$ satisfies the Lipschitz condition

$$\|\mathcal{G}_{\Delta T}(\boldsymbol{u}) - \mathcal{G}_{\Delta T}(\boldsymbol{v})\| \leqslant L_{\mathcal{G}} \|\boldsymbol{u} - \boldsymbol{v}\|, \tag{4.8}$$

for $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{R}^d$ and Lipschitz constant $L_{\mathcal{G}} > 0$.

Note that these assumptions do not restrict the choice of $\mathcal{G}_{\Delta T}$, as they are met when choosing any Runge-Kutta or Taylor method (Hairer et al., 1993).

In addition to assumptions on the flow maps, we require an assumption on the absolute moments of the (state-independent) random variables, which will be needed to prove Theorem 4.7.

Assumption 4.5 (Bounded absolute moments of $\boldsymbol{\xi}_n^k$)**.** For $q \geqslant 0$, $\tilde{r} \in \mathbb{N} \cup \{\infty\}$, and $C_2 > 0$ independent of n, k, and ΔT, the r-th absolute moments of $\boldsymbol{\xi}_n^k$ satisfy

$$\mathbb{E}\big[\|\boldsymbol{\xi}_n^k\|^r\big] \leqslant \big(C_2 \Delta T^{q+\frac{1}{2}}\big)^r, \quad 1 \leqslant r \leqslant \tilde{r}. \tag{4.9}$$

This assumption enables flexibility in defining the state-independent perturbations, in the sense that it does not require that the random variables be centred or i.i.d. (Lie et al., 2019). It also means that each $\boldsymbol{\xi}_n^k$ could follow a different probability distribution, with the only requirement being that they share a common maximal bound on their absolute moments with respect to the norm. Note that we assume $\Delta T < 1$ without loss of generality here, so that for increasing q, the perturbations get smaller and smaller. For $\Delta T > 1$, we can simply take q to be negative.

These assumptions will enable us to derive error bounds in the state-independent and state-dependent cases (using sampling rules 2 and 4). The sampling rule 1 and 3 cases require an additional assumption.

Assumption 4.6 (Lipschitz exact flow $\mathcal{F}_{\Delta T}$)**.** The flow map $\mathcal{F}_{\Delta T}$ satisfies the Lipschitz condition

$$\|\mathcal{F}_{\Delta T}(\boldsymbol{u}) - \mathcal{F}_{\Delta T}(\boldsymbol{v})\| \leqslant L_{\mathcal{F}} \|\boldsymbol{u} - \boldsymbol{v}\|, \tag{4.10}$$

for $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{R}^d$ and constant $L_{\mathcal{F}} > 0$.

4.2.1 State-independent perturbations

In this section, we derive error bounds for SPararеal when using the state-independent perturbations $\boldsymbol{\xi}_n^k(\boldsymbol{U}_n^k) = \boldsymbol{\xi}_n^k$.

Theorem 4.7 (Superlinear error bound for state-independent perturbations)**.** *Suppose the SPararеal scheme* (4.1) *with* $\boldsymbol{\xi}_n^k(\boldsymbol{U}_n^k) = \boldsymbol{\xi}_n^k$ *satisfies Assumptions 4.2, 4.3, 4.4, and 4.5. Then, the mean-square error* (4.3) *of the solution to the nonlinear ODE system* (2.1) *at iteration* k *and time* t_n *satisfies*

$$e_n^k \leqslant D A^{k-1} \sum_{\ell=0}^{n-k} \binom{\ell + k - 1}{\ell} B^\ell + \Lambda \sum_{j=0}^{k-2} \sum_{\ell=0}^{n-(j+1)} \binom{\ell + j}{\ell} A^j B^\ell,$$

for $2 \leqslant k < n \leqslant N$ *and constants* $A = C_1^2 \Delta T^{2p+2}(2 + \Delta T^{-1})$, $B = L_{\mathcal{G}}^2(1 + 2\Delta T)$, $\Lambda = C_2^2 \Delta T^{2q+1}(2 + \Delta T^{-1})$, *and* $D = A\hat{e}^0$.

Proof. Using (4.1d), that $\mathcal{F}_{\Delta T}$ is the exact solver (4.5), and adding and subtracting $\mathcal{G}_{\Delta T}(\boldsymbol{u}(t_n))$, we see that

$$\begin{aligned} e_{n+1}^{k+1} &= \mathbb{E}\big[\|\mathcal{F}_{\Delta T}(\boldsymbol{u}(t_n)) - \big(\mathcal{G}_{\Delta T}(\boldsymbol{U}_n^{k+1}) + \mathcal{F}_{\Delta T}(\boldsymbol{U}_n^k) - \mathcal{G}_{\Delta T}(\boldsymbol{U}_n^k) + \boldsymbol{\xi}_n^k(\boldsymbol{U}_n^k)\big) \\ &\qquad + \mathcal{G}_{\Delta T}(\boldsymbol{u}(t_n)) - \mathcal{G}_{\Delta T}(\boldsymbol{u}(t_n))\|^2\big] \\ &= \mathbb{E}[\|W_1 + W_2 + W_3\|^2], \end{aligned}$$

where W_1, W_2, and W_3 are given by

$$\begin{aligned} W_1 &= \mathcal{F}_{\Delta T}(\boldsymbol{u}(t_n)) - \mathcal{G}_{\Delta T}(\boldsymbol{u}(t_n)) - \big(\mathcal{F}_{\Delta T}(\boldsymbol{U}_n^k) - \mathcal{G}_{\Delta T}(\boldsymbol{U}_n^k)\big), \\ W_2 &= \mathcal{G}_{\Delta T}(\boldsymbol{u}(t_n)) - \mathcal{G}_{\Delta T}(\boldsymbol{U}_n^{k+1}), \\ W_3 &= -\boldsymbol{\xi}_n^k. \end{aligned}$$

Then, using the triangle inequality and (D.1) for the cross terms (Engblom, 2009, Sec. 4.2), we obtain

$$e_{n+1}^{k+1} \leqslant (1 + \delta_1^{-1} + \delta_2^{-1})\mathbb{E}[\|W_1\|^2] + (1 + \delta_1 + \delta_3^{-1})\mathbb{E}[\|W_2\|^2] + (1 + \delta_2 + \delta_3)\mathbb{E}[\|W_3\|^2]. \quad (4.11)$$

Using (4.7), we can bound

$$\mathbb{E}[\|W_1\|^2] \leqslant C_1^2 \Delta T^{2(p+1)} e_n^k. \quad (4.12)$$

Applying the Lipschitz condition (4.8), we obtain

$$\mathbb{E}[\|W_2\|^2] \leqslant L_{\mathcal{G}}^2 e_n^{k+1}. \quad (4.13)$$

Using (4.9) with $r = 2$, we obtain

$$\mathbb{E}[\|W_3\|^2] \leqslant C_2^2 \Delta T^{2q+1}. \quad (4.14)$$

Plugging (4.12)–(4.14) into (4.11) and choosing $\delta_1 = \Delta T$, $\delta_2 = 1$, and $\delta_3 = \Delta T^{-1}$, we obtain the double recursion

$$e_{n+1}^{k+1} \leqslant A e_n^k + B e_n^{k+1} + \Lambda, \quad e_{n+1}^1 \leqslant D + B e_n^1, \quad (4.15)$$

where $A = C_1^2 \Delta T^{2p+2}(2 + \Delta T^{-1})$, $B = L_{\mathcal{G}}^2(1 + 2\Delta T)$, $\Lambda = C_2^2 \Delta T^{2q+1}(2 + \Delta T^{-1})$, and $D = A\hat{e}^0$. Solving (4.15) using the generating function method in Lemma D.3, we obtain the result. □

One can make alternative choices for δ_1, δ_2, and δ_3, however, the choices given in the proof above seem to yield the tightest error bounds. If we were to maximise (4.15) over n, we obtain the following linear error bound in the case that $B < 1$, i.e. $L_{\mathcal{G}} < (1 + 2\Delta T)^{-1/2}$.

Corollary 4.8 (Linear error bound for state-independent perturbations)**.** *Suppose the SPararеal scheme* (4.1) *with* $\boldsymbol{\xi}_n^k(\boldsymbol{U}_n^k) = \boldsymbol{\xi}_n^k$ *satisfies Assumptions 4.2, 4.3, 4.4, and 4.5. Then, the maximal mean-square error* (4.4) *of the solution to the nonlinear ODE system* (2.1) *at iteration* k *satisfies*

$$\hat{e}^k \leqslant \hat{e}^1 \left(\frac{A}{1-B}\right)^{k-1} + \frac{\Lambda}{1-B} \sum_{j=0}^{k-2} \left(\frac{A}{1-B}\right)^j, \quad \textit{if} \quad B < 1,$$

for $2 \leqslant k \leqslant N$ *and constants* $A = C_1^2 \Delta T^{2p+2}(2 + \Delta T^{-1})$, $B = L_{\mathcal{G}}^2(1 + 2\Delta T)$, *and* $\Lambda = C_2^2 \Delta T^{2q+1}(2 + \Delta T^{-1})$.

Proof. Following the proof of Theorem 4.7, we maximise (4.15) over n to obtain

$$\hat{e}^{k+1} \leqslant \tilde{A}\hat{e}^k + \tilde{\Lambda}, \tag{4.16}$$

where

$$\tilde{A} = \frac{A}{1-B} \quad \text{and} \quad \tilde{\Lambda} = \frac{\Lambda}{1-B}.$$

Solving recursion (4.16) with initial condition $\hat{e}^1$, we obtain the desired result. □

Remark 4.9. The bounds in Theorem 4.7 and Corollary 4.8 hold for $2 \leqslant k < n \leqslant N$ due to the design of the SPararеal scheme. We can recover the bound for iteration $k = 1$ (which is deterministic) by solving the second recursion in (4.15) with initial value $e_1^1 = 0$ such that

$$e_n^1 \leqslant \hat{e}^0 A \sum_{i=0}^{n-2} B^i, \quad 1 \leqslant n \leqslant N. \tag{4.17}$$

For the case when $k = n$, the numerical error is zero, as $\mathcal{F}_{\Delta T}$ will have propagated the exact initial condition at t_0 forward k times without any perturbations, just like Parareal.

Remark 4.10. The bound in Theorem 4.7 (similarly for Corollary 4.8) can be written as

$$e_n^k \leqslant C_{k,n} \max\{\Delta T^{(2p+1)k}, \Delta T^{2q}\}, \tag{4.18}$$

where $C_{k,n}$ is a function of n and k. Assuming $\Delta T < 1$ and that $C_{k,n}$ is non-increasing in k, the accuracy of SPararеal should increase with each iteration proportional to the local truncation error of $\mathcal{G}$ (i.e. the term $\Delta T^{(2p+1)k}$) up until the errors induced by the perturbations (i.e. ΔT^{2q}) become dominant. We illustrate this property numerically in Section 4.3.

Remark 4.11. As $\Delta T \to 0$, both error bounds go to zero as expected, as can be seen clearly in (4.18). The intuition being that as $\Delta T \to 0$, the local truncation error of $\mathcal{G}_{\Delta T}$ goes to zero, i.e. it 'becomes' the exact flow map $\mathcal{F}_{\Delta T}$, see (4.7).

Remark 4.12. If we send $q \to \infty$, the second moments of the random variables (in both Theorem 4.7 and Corollary 4.8) vanish and we recover the classic Parareal scheme. These bounds are not identical to those calculated by Gander and Vandewalle (2007), Gander and Hairer (2008), and Gander et al. (2022) because we are working with the mean-square error (4.3), not the (mean) absolute error. To recover Theorem 2.2, we can easily bound the mean absolute error $\tilde{e}_n^k \coloneqq \mathbb{E}\big[\|\boldsymbol{u}(t_n) - \boldsymbol{U}_n^k\|\big]$ using the same result as the Theorem 4.7 which yields constants $A = C_1 \Delta T^{p+1}$, $B = L_{\mathcal{G}}$, $\Lambda = C_2 \Delta T^{q+\frac{1}{2}}$, and $D = A \max_n\{\tilde{e}_n^k\}$. If we now send $q \to \infty$, we recover the bound in Theorem 2.2 without the expectation of course (noting that the SParareal result holds for $k \geqslant 2$ whereas the Parareal result holds for $k \geqslant 1$).

Remark 4.13. If we additionally assume that the random variables $\boldsymbol{\xi}_n^k$ are centred, i.e. $\mathbb{E}[\boldsymbol{\xi}_n^k] = \mathbf{0}$, and work in the 2-norm, i.e. $\|\boldsymbol{u}\|_2^2 = \langle \boldsymbol{u}, \boldsymbol{u} \rangle = \sum_{i=1}^d u_i^2$, in the proof of Theorem 4.7, we can write (4.11) as

$$e_{n+1}^{k+1} \leqslant (1 + \delta_1^{-1})\mathbb{E}[\|W_1\|^2] + (1 + \delta_1)\mathbb{E}[\|W_2\|^2] + \mathbb{E}[\|W_3\|^2] \\ + 2\mathbb{E}[\langle W_1, W_3 \rangle] + 2\mathbb{E}[\langle W_2, W_3 \rangle],$$

where the final two terms are equal to zero by independence of W_1 and W_2 with W_3 and using the fact that each $\boldsymbol{\xi}_n^k$ is centred. Continuing the proof, we obtain the same bounds for Theorem 4.7 and Corollary 4.8 with slightly altered constants $A = C_1^2 \Delta T^{2p+2}(1 + \Delta T^{-1})$, $B = L_{\mathcal{G}}^2(1 + \Delta T)$, $\Lambda = C_2^2 \Delta T^{2q+1}$, and $D = A\hat{e}^0$.

4.2.2 State-dependent perturbations (sampling rules)

We now use the previous results to derive the corresponding error bounds for the state-dependent sampling rules defined in Table 4.1.

Corollary 4.14 (Linear error bound for state-dependent sampling rules 2 and 4)**.** *Suppose the SParareal scheme* (4.1) *satisfies Assumptions 4.2, 4.3, and 4.4, with* $\boldsymbol{\xi}_n^k(\boldsymbol{U}_n^k)$ *defined using sampling rule 2 or 4. Then, the maximal mean-square error* (4.4) *of the solution to the nonlinear ODE system* (2.1) *at iteration* k *satisfies*

$$\hat{e}^k \leqslant \hat{e}^0 \left[\frac{A + \Lambda_1 + \sqrt{(A + \Lambda_1)^2 + 4\Lambda_2(1 - B)}}{2(1 - B)} \right]^k, \quad \text{if} \quad B < 1,$$

for $2 \leqslant k \leqslant N$ *and constants* $A = C_1^2 \Delta T^{2p+2}(2 + \Delta T^{-1})$, $B = L_{\mathcal{G}}^2(1 + 2\Delta T)$, $\Lambda_1 = C_1^2 \Delta T^{2p+2} L_{\mathcal{G}}^2(1 + \Delta T^{-1})$, *and* $\Lambda_2 = C_1^2 \Delta T^{2p+2} L_{\mathcal{G}}^2(1 + \Delta T)$.

Proof for sampling rule 2. The proof follows in the same fashion as Theorem 4.7. Instead of using the bound (4.14), we obtain, using (4.2) and applying (4.7),

$$\mathbb{E}[\|W_3\|^2] \leqslant C_1^2 \Delta T^{2(p+1)} \mathbb{E}[\|\boldsymbol{\alpha}_n^k - \boldsymbol{U}_n^k\|^2]. \tag{4.19}$$

Substituting in $\boldsymbol{\alpha}_n^k$ for sampling rule 2 (Table 4.1) we get

$$\begin{aligned}\mathbb{E}[\|W_3\|^2] &\leqslant C_1^2 \Delta T^{2(p+1)} \mathbb{E}[\|\boldsymbol{\sigma}_n^k \circ \boldsymbol{z}_n^k\|^2] \\ &\leqslant C_1^2 \Delta T^{2(p+1)} \mathbb{E}[\|\boldsymbol{\sigma}_n^k\|^2] \mathbb{E}[\|\boldsymbol{z}_n^k\|^2] \\ &\leqslant C_1^2 \Delta T^{2(p+1)} L_{\mathcal{G}}^2 \mathbb{E}[\|\boldsymbol{U}_{n-1}^k - \boldsymbol{U}_{n-1}^{k-1}\|^2].\end{aligned}$$

The second inequality follows by Cauchy-Schwarz and independence of $\boldsymbol{\sigma}_n^k$ and $\boldsymbol{z}_n^k$. The third follows by plugging in $\boldsymbol{\sigma}_n^k$, applying (4.8), and noting that $\mathbb{E}[\|\boldsymbol{z}_n^k\|^2] = 1$. Next, we add and subtract $\boldsymbol{u}(t_{n-1})$ inside the expectation and then apply (D.1), with $\delta = \Delta T$, to get

$$\mathbb{E}[\|W_3\|^2] \leqslant C_1^2 \Delta T^{2(p+1)} L_{\mathcal{G}}^2 \big((1 + \Delta T^{-1}) e_{n-1}^k + (1 + \Delta T) e_{n-1}^{k-1}\big). \tag{4.20}$$

Using the new bound for $\mathbb{E}[\|W_3\|^2]$ in (4.11), we obtain the double recurrence

$$e_{n+1}^{k+1} \leqslant A e_n^k + B e_n^{k+1} + \Lambda_1 e_{n-1}^k + \Lambda_2 e_{n-1}^{k-1}, \tag{4.21}$$

where $A = C_1^2 \Delta T^{2p+2}(2 + \Delta T^{-1})$, $B = L_{\mathcal{G}}^2(1 + 2\Delta T)$, $\Lambda_1 = C_1^2 \Delta T^{2p+2} L_{\mathcal{G}}^2 (1 + \Delta T^{-1})$, and $\Lambda_2 = C_1^2 \Delta T^{2p+2} L_{\mathcal{G}}^2 (1 + \Delta T)$. Maximising over n, we obtain

$$\hat{e}^{k+1} \leqslant \tilde{A} \hat{e}^k + \tilde{B} \hat{e}^{k-1}, \tag{4.22}$$

where

$$\tilde{A} = \frac{A + \Lambda_1}{1 - B} \quad \text{and} \quad \tilde{B} = \frac{\Lambda_2}{1 - B}.$$

Recursion (4.22) can be solved using Lemma D.4, resulting in the desired bound. □

Proof for sampling rule 4. The proof follows in the same way as the proof for sampling rule 2, except that $\mathbb{E}[\|\sqrt{3}(2\boldsymbol{w}_n^k - \mathbf{1})\|^2] = 1$ is used in place of $\mathbb{E}[\|\boldsymbol{z}_n^k\|^2] = 1$. □

Corollary 4.15 (Linear error bound for state-dependent sampling rules 1 and 3)**.** *Suppose the SPararеal scheme* (4.1) *satisfies Assumptions 4.2, 4.3, 4.4, and 4.6, with* $\boldsymbol{\xi}_n^k(\boldsymbol{U}_n^k)$ *defined using sampling rule 1 or 3. Then, the maximal mean-square error*

(4.4) *of the solution to the nonlinear ODE system* (2.1) *at iteration* k *satisfies*

$$\hat{e}^k \leqslant \hat{e}^0 \left[\frac{A + \Lambda_1 + \Lambda_3 + \sqrt{(A + \Lambda_1 + \Lambda_3)^2 + 4\Lambda_2(1 - B)}}{2(1 - B)} \right]^k, \quad \textit{if} \quad B < 1,$$

for $2 \leqslant k \leqslant N$ *and constants* $A = C_1^2 \Delta T^{2p+2}(2 + \Delta T^{-1})$, $B = L_{\mathcal{G}}^2(1 + 2\Delta T)$, $\Lambda_1 = 2C_1^2 \Delta T^{2p+2} L_{\mathcal{G}}^2(1 + \Delta T^{-1})$, $\Lambda_2 = 2C_1^2 \Delta T^{2p+2}(L_{\mathcal{G}}^2(1 + \Delta T) + 2L_{\mathcal{F}}^2)$, *and* $\Lambda_3 = 4C_1^2 \Delta T^{2p+2}$.

Proof for sampling rule 1. The proof follows in the same fashion as Corollary 4.14. Substituting $\boldsymbol{\alpha}_n^k$ for sampling rule 1 (Table 4.1) in (4.19), we get

$$\begin{aligned} \mathbb{E}[\|W_3\|^2] &\leqslant C_1^2 \Delta T^{2(p+1)} \mathbb{E}[\|\mathcal{F}_{\Delta T}(\boldsymbol{U}_{n-1}^{k-1}) - \boldsymbol{U}_n^k + \boldsymbol{\sigma}_n^k \circ \boldsymbol{z}_n^k\|^2] \\ &\leqslant 2C_1^2 \Delta T^{2(p+1)} \big(\underbrace{\mathbb{E}[\|\mathcal{F}_{\Delta T}(\boldsymbol{U}_{n-1}^{k-1}) - \boldsymbol{U}_n^k\|^2]}_{\text{1st Term}} + \underbrace{\mathbb{E}[\|\boldsymbol{\sigma}_n^k \circ \boldsymbol{z}_n^k\|^2]}_{\text{2nd Term}} \big). \end{aligned} \tag{4.23}$$

The second inequality follows by applying (D.1) with $\delta = 1$. To bound the first term in (4.23), we add and subtract $\mathcal{F}_{\Delta T}(\boldsymbol{u}(t_{n-1}))$ inside the expectation and apply (D.1) again with $\delta = 1$, obtaining

$$\begin{aligned} \text{1st Term} &\leqslant 2\big(\mathbb{E}[\|\mathcal{F}_{\Delta T}(\boldsymbol{U}_{n-1}^{k-1}) - \mathcal{F}_{\Delta T}(\boldsymbol{u}(t_{n-1}))\|^2] + \mathbb{E}[\|\mathcal{F}_{\Delta T}(\boldsymbol{u}(t_{n-1})) - \boldsymbol{U}_n^k\|^2]\big) \\ &\leqslant 2\big(L_{\mathcal{F}}^2 e_{n-1}^{k-1} + e_n^k\big). \end{aligned}$$

The second inequality follows by applying the Lipschitz condition (4.10) and recalling that $\mathcal{F}_{\Delta T}$ is the exact solver (4.5). The second term in (4.23) can be bounded as in (4.20) in Corollary 4.14,

$$\text{2nd Term} \leqslant L_{\mathcal{G}}^2\big((1 + \Delta T^{-1}) e_{n-1}^k + (1 + \Delta T) e_{n-1}^{k-1}\big).$$

Combining both terms in (4.23), we obtain

$$\mathbb{E}[\|W_3\|^2] \leqslant \Lambda_1 e_{n-1}^k + \Lambda_2 e_{n-1}^{k-1} + \Lambda_3 e_n^k,$$

where $\Lambda_1 = 2C_1^2 \Delta T^{2p+2} L_{\mathcal{G}}^2(1 + \Delta T^{-1})$, $\Lambda_2 = 2C_1^2 \Delta T^{2p+2}(L_{\mathcal{G}}^2(1 + \Delta T) + 2L_{\mathcal{F}}^2)$, and $\Lambda_3 = 4C_1^2 \Delta T^{2p+2}$. Using the new bound for $\mathbb{E}[\|W_3\|^2]$ in (4.11), we obtain the following recurrence

$$e_{n+1}^{k+1} \leqslant (A + \Lambda_3) e_n^k + B e_n^{k+1} + \Lambda_1 e_{n-1}^k + \Lambda_2 e_{n-1}^{k-1}, \tag{4.24}$$

where $A = C_1^2 \Delta T^{2p+2}(2 + \Delta T^{-1})$ and $B = L_{\mathcal{G}}^2(1 + 2\Delta T)$. Maximising over n, we obtain

$$\hat{e}^{k+1} \leqslant \tilde{A} \hat{e}^k + \tilde{B} \hat{e}^{k-1}, \tag{4.25}$$

where

$$\tilde{A} = \frac{A + \Lambda_1 + \Lambda_3}{1 - B} \quad \text{and} \quad \tilde{B} = \frac{\Lambda_2}{1 - B}.$$

Recursion (4.25) can be solved using Lemma D.4, resulting in the desired bound. □

Proof for sampling rule 3. The proof follows in a similar fashion to the proof for sampling rule 1, with $\mathbb{E}[\|\sqrt{3}(2\boldsymbol{w}_n^k - \mathbf{1})\|^2] = 1$ being used in place of $\mathbb{E}[\|\boldsymbol{z}_n^k\|^2] = 1$. □

Remark 4.16. In Section 4.3, we can observe the behaviour of e_n^k (not just $\hat{e}^k$) for each of the sampling rules by solving the recursions (4.21) and (4.24) numerically. We do this by replacing the inequality with an equality, i.e. upper bounding the error estimate.

4.3 Numerical experiments

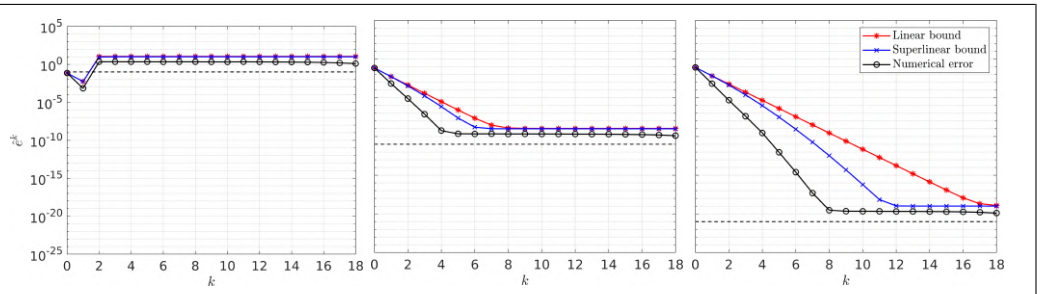
Here, we present some experiments to compare the theoretical bounds derived in Section 4.2 with the errors generated by running SParareal numerically.

4.3.1 System of linear ODEs

In the following experiments, we solve the linear system

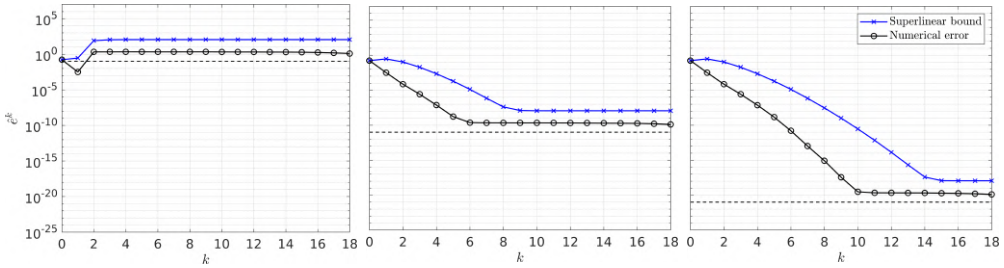
$$\frac{\mathrm{d}\boldsymbol{u}}{\mathrm{d}t} = Q\boldsymbol{u} \quad \text{over} \quad t \in [0, T], \quad \text{with} \quad \boldsymbol{u}(0) = \boldsymbol{u}^0, \tag{4.26}$$

where $Q \in \mathbb{R}^{d \times d}$. This system has the exact solution $\boldsymbol{u}(t) = \boldsymbol{u}^0 e^{Qt}$, where $e^{Qt} = \sum_{i=0}^{\infty} (Qt)^i / i!$ is the matrix exponential.



(a) $q = 0$ **(b)** $q = 5$ **(c)** $q = 10$

Figure 4.1: Theoretical bounds vs. numerical errors for SParareal applied to the linear system of ODEs (4.26) (with $B < 1$) using state-independent Gaussian perturbations (4.27). The superlinear bound (Theorem 4.7) is given in blue, the linear bound (Corollary 4.8) in red, the numerical error in black, and ΔT^{2q+1} in dashed black. Each plot corresponds to a different level of Gaussian noise: (a) $q = 0$, (b) $q = 5$, and (c) $q = 10$. Numerical errors were calculated by averaging over 500 independent realisations of SParareal.



(a) $q = 0$ **(b)** $q = 5$ **(c)** $q = 10$

Figure 4.2: Theoretical bounds vs. numerical errors for SParareal applied to the linear system of ODEs (4.26) (with $B \geqslant 1$) using state-independent Gaussian perturbations (4.27). The superlinear bound (Theorem 4.7) is given in blue, the numerical error in black, and ΔT^{2q+1} in dashed black. Each plot corresponds to a different level of Gaussian noise: (a) $q = 0$, (b) $q = 5$, and (c) $q = 10$. Numerical errors were calculated by averaging over 500 independent realisations of SParareal.

First, we examine the superlinear and linear bounds derived in Theorem 4.7 and Corollary 4.8, respectively, by running SParareal numerically with state-independent Gaussian perturbations

$$\boldsymbol{\xi}_n^k \sim \mathcal{N}(\mathbf{0}, \Delta T^{2q+1}\mathbb{I}_d), \quad q \geqslant 0. \tag{4.27}$$

We solve (4.26) with $d = 100$ and $T = 2$, discretising the time interval into $N = 20$ time slices so that $\Delta T = 0.1$. We construct the matrix of coefficients Q such that $B < 1$ and also select a fixed initial condition $\boldsymbol{u}^0 \in [-5, 5]^d$. We use the exact solver $\mathcal{F}_{\Delta T}(\boldsymbol{u}) = \boldsymbol{u}e^{Q\Delta T}$ and the forward Euler method $\mathcal{G}_{\Delta T}(\boldsymbol{u}) = (\mathbb{I}_d + Q\Delta T)\boldsymbol{u}$. In Figure 4.1, we plot the maximal theoretical bounds $\hat{e}^k$ and numerical errors of

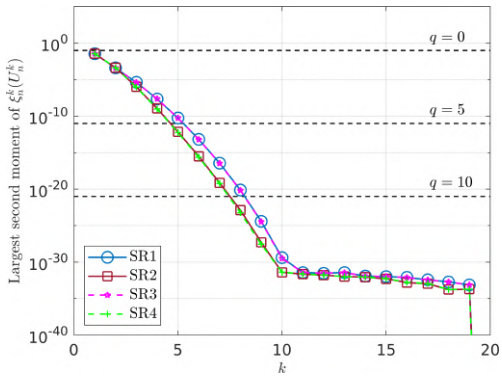
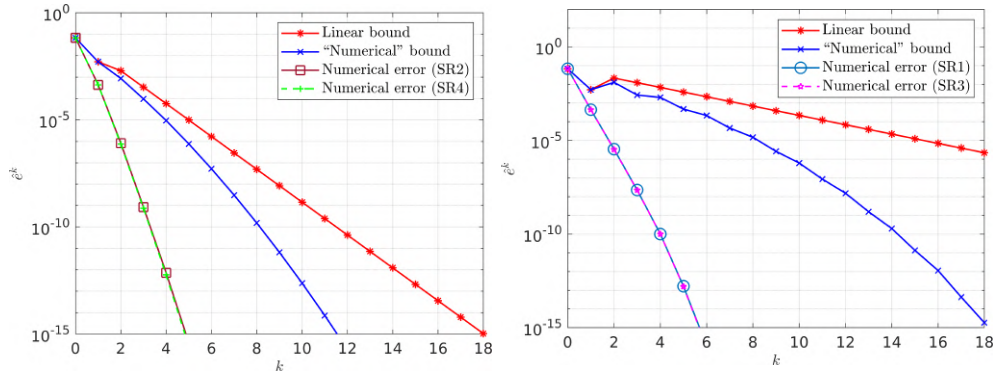


Figure 4.3: Largest second moments (over n) of $\boldsymbol{\xi}_n^k(\boldsymbol{U}_n^k)$ for the sampling rules 1 to 4 (light blue, brown, purple, and green respectively) and the Gaussian perturbations (4.27) for $q \in \{0, 5, 10\}$ (dashed black), plotted against iteration number k. Second moments for the sampling rules were calculated by averaging over 500 independent realisations of SParareal.



(a) Sampling rules 2 and 4 **(b)** Sampling rules 1 and 3

Figure 4.4: Theoretical bounds vs. numerical errors for SPararеal applied to the linear system of ODEs (4.26) (with $B < 1$) using the state-dependent sampling rules. (a) The linear bound in Corollary 4.14 is shown in red, the numerically solved recursion (4.21) in blue, and the numerical errors for sampling rules 2 and 4 in brown and green, respectively. (b) The linear bound in Corollary 4.15 is shown in red, the numerically solved recursion (4.24) in blue, and the numerical errors for sampling rules 1 and 3 in light blue and purple, respectively. Numerical errors were calculated by averaging over 500 independent realisations of SPararеal.

SPararеal as a function of k for different values of q when $B < 1$. These results illustrate how the errors decrease as k increases (except when $q = 0$), up until the error induced by the perturbations become dominant—exactly the effect described in Remark 4.10. For all considered values of q, the error for $k \geqslant 2$ has a hard lower bound of $\mathcal{O}(\Delta T^{2q+1})$, i.e. the error cannot go below the second moments of the perturbations (indicated by the dashed black line in each case). By altering Q and

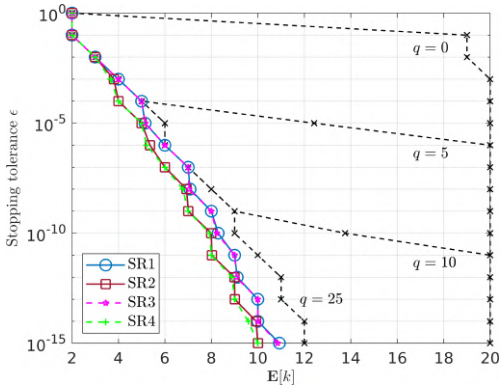


Figure 4.5: Expected number of iterations k taken to reach stopping tolerance ε (2.10) for SPararеal applied to the linear system (4.26) (with $B < 1$). Results plotted using SPararеal with each sampling rule (see legend) and the Gaussian perturbations (4.27) for $q \in \{0, 5, 10, 25\}$ (dashed black lines). $\mathbb{E}[k]$ calculated by averaging k over 500 independent realisations of SPararеal.

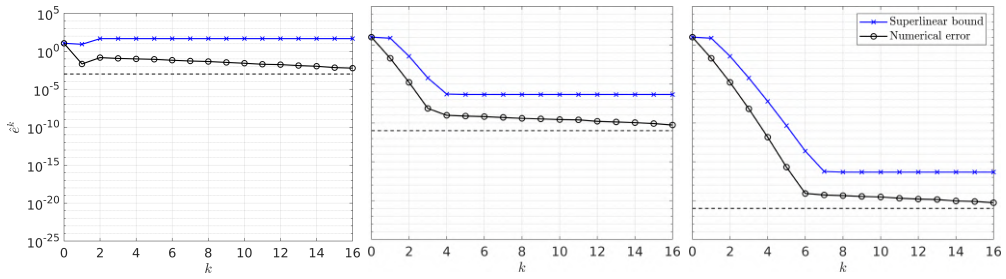
running the same experiment, we see similar effects in the $B \geqslant 1$ case, see Figure 4.2.

It can be seen that, regardless of B, using the state-independent perturbations may not be optimal because of the lower bound forced upon the errors. If they are to be used, then they would need to be chosen such that the second moments are smaller than the accuracy of the solutions sought. This approach, however, may not yield accelerated convergence over the classic Parareal scheme. To avoid this (and the lower bound on accuracy), the perturbations need to be state-dependent and therefore able to adapt, i.e. the second moments need to decrease with k and scale with n. In Figure 4.3, we illustrate how the second moments of the perturbations used in the state-dependent sampling rules decrease with k throughout the SParareal simulation, comparing these to the fixed second moments of the Gaussians (4.27) for each $q \in \{0, 5, 10\}$ (dashed lines). Using the sampling rules enables SParareal to sample from probability distributions that begin to "contract" around the exact solution states as the simulation progresses, i.e. as k increases. This results in high solution accuracy in very few iterations, as will be shown in Figure 4.4.

It should be noted that we could have also chosen a different distribution other than the Gaussian from which to sample each state-independent $\boldsymbol{\xi}_n^k$, as long as Assumption 4.5 is satisfied. For example, choosing uniformly distributed perturbations $\boldsymbol{\xi}_n^k \sim \mathcal{U}[-\sqrt{3}\Delta T^{q+\frac{1}{2}}, \sqrt{3}\Delta T^{q+\frac{1}{2}}]^d$ yielded almost identical results (not shown).

Next, we plot the linear bounds for perturbations defined by the sampling rules, i.e. Corollary 4.14 and Corollary 4.15, against the corresponding numerical errors in Figure 4.4 (for the $B < 1$ problem). We observe that the linear bounds are not that tight due to the maximisation over n required to derive them. However, by solving recursions (4.21) and (4.24) numerically (recall Remark 4.16), we observe a tighter bound on the error. All that is required to calculate these numerical bounds are the errors at the 'zeroth' iteration (obtained from SParareal itself by just running $\mathcal{G}$), errors at the first iteration (recall (4.17)) and the constants C_1 and $L_{\mathcal{G}}$. Note that the numerical errors for sampling rules 2/4 and 1/3 overlap because the perturbations used in each scheme have almost identical second moments (recall Figure 4.3).

In Figure 4.5, we compare the performance of the state-independent and -dependent perturbations by plotting the expected number of iterations $\mathbb{E}[k]$ taken to reach a pre-defined stopping tolerance ε, recall (2.10). We observe that, on average, the sampling rules reach tolerance in fewer iterations than the state-independent perturbations. The sampling rules also outperform classic Parareal, which can be observed by comparing them to the state-independent perturbations for $q = 25$, for which perturbations are so small that SParareal is practically deterministic (i.e. Parareal). Recall that reducing k by even a few iterations can significantly increase parallel speedup.



(a) $q = 1$ **(b)** $q = 5$ **(c)** $q = 10$

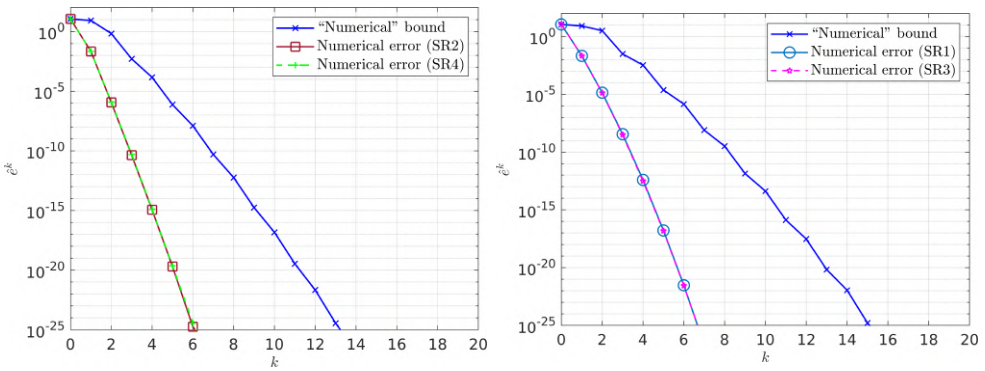
Figure 4.6: Theoretical bounds vs. numerical errors for SParareal applied to the nonlinear scalar ODE (4.28) (with $B \geqslant 1$) using state-independent Gaussian perturbations (4.27). The superlinear bound (Theorem 4.7) is given in blue, the numerical error in black, and ΔT^{2q+1} in dashed black. Each plot corresponds to a different level of Gaussian noise: (a) $q = 1$, (b) $q = 5$, and (c) $q = 10$. Numerical errors were calculated by averaging over 500 independent realisations of SParareal.

4.3.2 Scalar nonlinear ODE

In the following experiments, we solve the scalar nonlinear equation

$$\frac{\mathrm{d}u}{\mathrm{d}t} = \sqrt{u^2 + 2} \quad \text{over} \quad t \in [-1, 1], \quad \text{with} \quad u(-1) = 5. \tag{4.28}$$

This equation has exact solution $u(t) = \sqrt{2}\sinh(t + 1 + \sinh^{-1}(5/2))$. We solve (4.28) using SParareal with $N = 20$ time slices (thus $\Delta T = 0.1$), exact solver $\mathcal{F}_{\Delta T}(u) = \sqrt{2}\sinh(\Delta T + \sinh^{-1}(u/\sqrt{2}))$, and forward Euler $\mathcal{G}_{\Delta T} = u + \Delta T\sqrt{u^2 + 2}$.



(a) Sampling rules 2 and 4 **(b)** Sampling rules 1 and 3

Figure 4.7: "Numerical" bounds vs. numerical errors for SParareal applied to the nonlinear scalar ODE (4.28) (with $B \geqslant 1$) using the state-dependent sampling rules. (a) The numerically solved recursion (4.21) is shown in blue and the numerical errors for sampling rules 2 and 4 in brown and green, respectively. (b) The numerically solved recursion (4.24) is shown in blue and the numerical errors for sampling rules 1 and 3 in light blue and purple, respectively. Numerical errors were calculated by averaging over 500 independent realisations of SParareal.

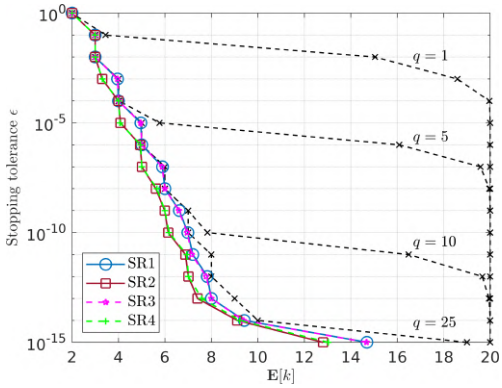


Figure 4.8: Expected number of iterations k taken to reach stopping tolerance ε (2.10) for SParareal applied to the nonlinear scalar ODE (4.28) (with $B \geqslant 1$). Results plotted using SParareal with each sampling rule (see legend) and the Gaussian perturbations (4.27) for $q \in \{0, 5, 10, 25\}$ (dashed black lines). $\mathbb{E}[k]$ calculated by averaging k over 500 independent realisations of SParareal.

Figure 4.6 illustrates a good match between the superlinear bound (Theorem 4.7, $B \geqslant 1$) and the numerical errors when using SParareal and the Gaussian perturbations (4.27). One can see that at $k = 0$ the error is quite large, $\mathcal{O}(10^1)$, and so even when using the forward Euler method for $\mathcal{G}_{\Delta T}$, the SParareal error decreases rapidly (for sufficiently large q). Given the bounds in Corollary 4.14 and Corollary 4.15 only hold when $B < 1$, we again solve the respective recursions (4.21) and (4.24) numerically, obtaining a good match between theory and numerics when using the sampling rules (see Figure 4.7). Figure 4.8 illustrates the performance of the state-independent perturbations and the sampling rules for varying stopping tolerances. As it did for the system of linear ODEs, using SParareal with state-dependent perturbations is more effective than with the state-independent perturbations, regardless of the chosen value of q for the Gaussian perturbations (recall that Parareal can be recovered when choosing $q \geqslant 25$).

4.4 Discussion and further work

The SParareal algorithm solves IVPs by perturbing solutions from the classic (deterministic) Parareal scheme using (in this version) a single sample drawn from pre-specified probability distributions. This sampling-based time-parallel scheme generates stochastic solutions to the IVP. In this chapter, we analysed the error of these stochastic numerical solutions by deriving mean-square error bounds for SParareal, equipped with different types of perturbations.

In Section 4.2, we make assumptions about the fine and coarse numerical integrators used by SParareal, namely that $\mathcal{F}_{\Delta T}$ returns the exact solution to the ODE and that $\mathcal{G}_{\Delta T}$ has uniform local truncation error and satisfies a Lipschitz condition. Error

bounds were then derived for two types of random perturbation, one in which the random variables do not depend on the current state of the system (state-independent) and one in which they do (state-dependent). In the state-independent case, where specific upper bounds were assumed on the second moments of the random variables, we derived both superlinear (Theorem 4.7) and linear (Corollary 4.8) bounds on the mean-square error. In the state-dependent case, where a number different perturbations were defined according to sampling rules (Section 4.1.2), we derived linear bounds on the errors (see Corollaries 4.14 and 4.15).

In Section 4.3, we illustrate these bounds, comparing them to the errors generated by running SPararеal numerically. We demonstrate a good match between the theoretical bounds and numerical errors for systems of linear ODEs and a scalar nonlinear ODE. Using the state-independent perturbations, we observed tight bounds with respect to the numerical errors. However, because these perturbations do not adapt with iteration k and time step n, their practical usage faces limitations. They encode a hard lower bound on solution accuracy (of the order of the size of the second moments, see Remark 4.10) and more iterations were typically required to reach stopping tolerance for larger perturbations. Instead, the sampling rules, shown to adapt with both k and n, did not suffer from these issues, as was previously discussed in Chapter 3. The derivation of the linear bounds for the sampling rules did, however, require multiple applications of the Peter-Paul inequality (Appendix D.1), resulting in less tight bounds compared to those found in the state-independent case. Tighter bounds were observed by solving the double recursions (4.21) and (4.24) numerically. In addition, these linear bounds required the constant B to be less than one (restricting its use to problems where the Lipschitz constant for $\mathcal{G}_{\Delta T}$ is smaller than one) and that $\mathcal{F}_{\Delta T}$ be Lipschitz continuous for sampling rules 1 and 3 (an additional restriction). In the future, it would be interesting to see if these restrictions can be avoided or whether one can derive bounds by relaxing the Lipschitz assumptions.

As HPC technology advances, the demand for faster and more accurate time-parallel integration methods will increase. With SPararеal, we have seen that introducing local perturbations into an existing time-parallel scheme can enable convergence in fewer iterations (using the sampling rules) and can, on average, result in higher accuracy solutions (refer back to numerical experiments in Sections 3.3 and 4.3). Following multiple realisations of SPararеal, these solutions can form a distribution over the exact solution, the accuracy of which can be estimated using the error bounds in this chapter. Further work is required to investigate whether similar bounds can be derived for the original SPararеal scheme (where M samples drawn instead of just one) which is able to locate solutions with increasing accuracy and numerical speedup when increasing numbers of samples are taken. In addition, in most practical applications, the exact flow map $\mathcal{F}_{\Delta T}$ is unknown and so it would be

advantageous to investigate what happens when one relaxes this assumption, taking $\mathcal{F}_{\Delta T}$ to be a numerical flow map.

Having now shown rigorously that solutions from SParareal are accurate, we conclude our work on SParareal. A discussion on its significance and impact will be made in Chapter 7. We now move on to discuss GParareal, a learning-based time parallel algorithm whereby we seek to make use of *all* solution data generated by Parareal in a Bayesian manner.

Chapter 5

GParareal I: a learning-based time-parallel algorithm

Overview

In this chapter we propose GParareal, a learning-based time-parallel algorithm that solves IVPs by inferring the (expensive) multi-fidelity correction term in Parareal, i.e. the difference between fine and coarse solutions, using a Gaussian process (GP) emulator. In SParareal, we showed that we could use the fine and coarse solution data at a given iteration to construct probability distributions for sampling, obtaining better corrections through increased sampling and propagation. We now approach the problem from a Bayesian viewpoint with the aim of using the *acquisition data*, i.e. the coarse and fine solutions from *all* prior iterations, to *infer* better corrections. We use these corrections in a slightly modified PC update, whereby the coarse solver makes rapid low-accuracy predictions (just like Parareal/SParareal) that are subsequently refined using a correction obtained by querying the "trained" GP emulator. As with SParareal, the hope is that the corrections provided by the GP emulator (trained using the acquisition data) are more accurate than those that would come from the standard Parareal correction.

We start in Section 5.1 by giving an overview of what a GP emulator is, how it works, and why they are extremely useful for emulating functions when one only has access to limited (expensive) data. We then provide a high level overview of the idea behind GParareal and motivate why using learning-based methods is beneficial for PinT simulations. Following this, we briefly review similar PinT algorithms that have made use of learning-based methods both inside and outside of the Parareal framework. In Section 5.2, we derive GParareal and explain, in detail, how the GP is conditioned on the acquisition data, how the GP hyperparameters are calculated, and provide expressions for the computational complexity. Using a result on GP posterior consistency and the assumptions defined in Chapter 4, we will also derive

an error bound for the GParareal solutions at a given iteration, showing that errors are proportional to the accuracy of the emulator.

In Section 5.3, we perform numerical experiments on HPC facilities to compare and contrast the performance of GParareal and Parareal. We demonstrate good performance in terms of convergence, wallclock time, and solution accuracy on a number of low-dimensional nonlinear ODE problems using just acquisition data. Furthermore, we demonstrate how the GP emulator captures variability in the correction term, enabling convergence in cases where the coarse solver is too inaccurate for Parareal. We also show that GParareal has the advantage of being able to use archives of "legacy data", e.g. solutions from prior runs of GParareal using different initial conditions, to further accelerate convergence of the method. Strong scaling experiments confirm that our theoretical wallclock time and speedup estimates for GParareal (and Parareal) are valid. In addition, these experiments highlight that the cost of training the emulator must be small with respect to the fine solver in order for full speedup potential to be realised.

In Section 5.4, we discuss a modification to GParareal that can help with slow convergence in situations where the emulator is insufficiently well-trained to locate a solution in a small number of iterations. We show how one can introduce a "Parareal fallback correction" by placing a switching tolerance on the GP posterior standard deviations. This allows GParareal to automatically switch between taking corrections from the GP emulator (when the variance is below tolerance) and the standard Parareal correction (when variance is high and therefore the posterior mean of the emulator is poor). We examine the feasibility of using the switching condition with numerical experiments on the Lorenz96 system. We conclude in Section 5.5, discussing the benefits, drawbacks, and open questions surrounding GParareal—some of which we explore in Chapter 6.

5.1 Motivation and background

As before, we seek the same high resolution numerical solutions to (2.1) as expressed in (2.3), except now we solve for an autonomous scalar ODE, i.e. $\boldsymbol{f}(t, \boldsymbol{u}(t)) \coloneqq f(u(t))$. This is just to simplify the explanation of GParareal—we will describe the extension to the full nonautonomous multivariate case in Section 5.2.5. As with Parareal and SParareal, we will denote the iteratively improved approximations from GParareal as U_n^k (as before, $U_0^k = U_0 = u^0 \ \forall k \geqslant 0$).

5.1.1 Gaussian process emulation

Emulators, also known as *surrogate models*, are statistical models that can approximate the output of deterministic black-box simulators (O'Hagan, 2006). A *simulator* can be regarded as an expensive-to-evaluate function $g\colon \mathbb{R} \to \mathbb{R}$ that takes an input

$x \in \mathbb{R}$ and produces an output $y = g(x)$, often taking seconds, minutes, or hours to generate such an output. Emulators are designed to be able to rapidly infer (i.e. generate a posterior probability distribution over) the value of $g(x')$ at any input location $x' \in \mathbb{R}$ without evaluating $g(x')$ itself. They do so by conditioning a prior distribution over g on a limited set of simulator runs $\{(x_j, y_j)\}_{j=1}^{D}$, where $y_j = g(x_j)$. This set is often referred to as the set of *design points* or the *training set* and can often be generated in parallel.

GP emulators are a particular type of surrogate model that use multivariate Gaussian distributions to infer $g(x')$ given a set of training data $\{(x_j, y_j)\}_{j=1}^{D}$ (O'Hagan, 1978; Rasmussen and Williams, 2006). They are used almost ubiquitously in a number of different settings including Bayesian optimisation (Murphy, 2023, Sec. 6.8), Tsunami modelling (Beck and Guillas, 2016; Liu and Guillas, 2017), and galaxy dynamics (Gration and Wilkinson, 2019), to name but a few. Suppose we wish to use a GP emulator to infer the (*a priori* unknown) function $g(x) = \cos(x)$ with a small training set. Firstly, we define a GP as a collection of random variables, any finite subset of which has a joint Gaussian distribution (Rasmussen, 2004). Therefore, we can define a GP prior over g as

$$g \sim \mathcal{GP}(m, \kappa), \tag{5.1}$$

meaning that g is distributed as a GP with mean function $m\colon \mathbb{R} \to \mathbb{R}$ and covariance kernel $\kappa\colon \mathbb{R} \times \mathbb{R} \to \mathbb{R}$. To visualise what this means, consider a set of input values $\boldsymbol{x} \in \mathbb{R}^J$, which yield the corresponding vector of means $\mu(\boldsymbol{x}) = (m(x_j))_{j=1,\ldots,J}^{\intercal}$ and the covariance matrix $K(\boldsymbol{x}, \boldsymbol{x}) = (\kappa(x_i, x_j))_{i,j=1,\ldots,J}$. Suppose we choose $m(x) \equiv 0$ and use the square exponential (SE) kernel

$$\kappa(x_i, x_j) = \sigma^2 \exp\left(-\frac{(x_i - x_j)^2}{2\ell^2}\right), \quad \text{for some} \quad x_i, x_j \in \mathbb{R}, \tag{5.2}$$

with input and output length scales $\ell^2 = 1$ and $\sigma^2 = 0.75$, respectively[1]. The prior from (5.1) can now be written in finite dimensional form

$$g(\boldsymbol{x}) \sim \mathcal{N}\big(\mu(\boldsymbol{x}), K(\boldsymbol{x}, \boldsymbol{x})\big), \tag{5.3}$$

which we illustrate in Figure 5.1(a) and from which we can draw samples, see Figure 5.1(b).

Clearly, the prior (and the samples drawn) do not resemble $g(x) = \cos(x)$. Suppose we now have access to $D = 5$ evaluations of g—see Figure 5.1(c). We can

[1]These parameters, often referred to as *hyperparameters*, are specified *a priori* here, however, they are usually optimised with respect to the training dataset to obtain a more accurate GP posterior distribution—something we will discuss how to do in Section 5.2.2.

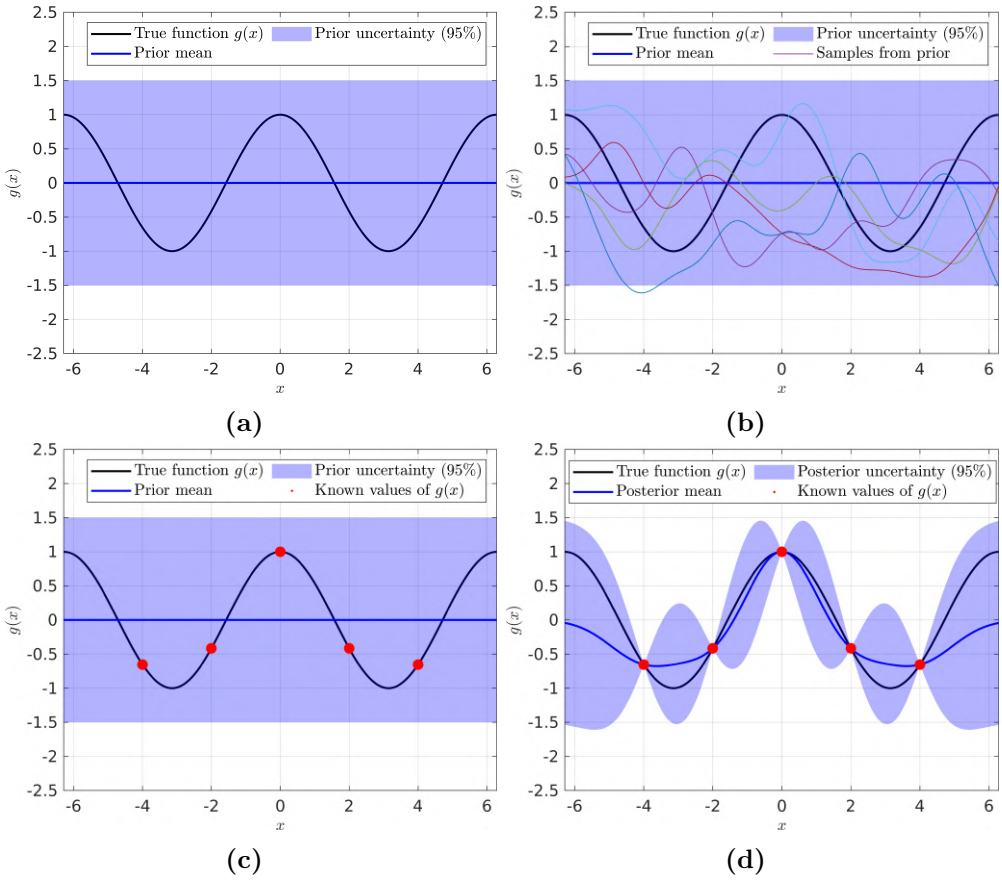


Figure 5.1: Modelling the unknown function $g(x) = \cos(x)$ using a GP emulator. (a) True unknown function $g(x)$ (solid black) and the GP prior over g (5.3) with mean (solid blue) and 95% confidence interval (light purple). (b) The same plot as (a) with five samples (multicoloured) drawn from the GP prior. (c) The GP prior, now plotted with $D = 5$ evaluations $\{(x_j, y_j)\}_{j=1}^{D}$ of g. (d) The GP posterior over g (5.4).

condition the prior (5.3) on these evaluations analytically[2] to obtain a Gaussian posterior distribution over $g(x')$ at any input location $x' \in \mathbb{R}$:

$$g(x') \mid \{\boldsymbol{x}, \boldsymbol{y}\} \sim \mathcal{N}\big(\hat{\mu}(x'), \hat{K}(x', x')\big), \tag{5.4}$$

with mean

$$\hat{\mu}(x') = \underbrace{\mu(x')}_{=0} + K(x', \boldsymbol{x})[K(\boldsymbol{x}, \boldsymbol{x})]^{-1}\big(\boldsymbol{y} - \underbrace{\mu(\boldsymbol{x})}_{=\mathbf{0}}\big) \tag{5.5}$$

and variance

$$\hat{K}(x', x') = K(x', x') - K(x', \boldsymbol{x})[K(\boldsymbol{x}, \boldsymbol{x})]^{-1} K(\boldsymbol{x}, x'). \tag{5.6}$$

[2]To see how the expressions of the posterior mean and covariance are calculated, see Murphy (2022, Sec. 3.2.3).

In Figure 5.1(d) we can see that the posterior mean now interpolates the training data exactly and the posterior variance is zero at these locations. As we move away from an observed value of $g(x)$, the mean deviates from the true $g(x)$ and the posterior variance increases, telling us that the emulator is more uncertain about the value of $g(x)$ at unobserved locations.

The key reason for using an emulator to model g is that we can rapidly query the posterior (5.4) at any $x' \in \mathbb{R}$ without evaluating $g(x')$ itself. The idea is that the cost of inferring $g(x')$ (5.4) (which is proportional to the cost of inverting the covariance matrix $K(\boldsymbol{x}, \boldsymbol{x})$) should be much smaller than the cost of evaluating $g(x')$. In addition, GP emulators are flexible enough that one can choose any prior mean and covariance functions based on known structure of the function being emulated. For example, if we know the function is periodic (as we know $g(x) = \cos(x)$ is) then we can prescribe this property within the mean/covariance functions—see Rasmussen and Williams (2006, Chp. 4). One can of course emulate multivariate functions as well (which we will need to do in GParareal) and for this one can use vector-valued GP emulators (Álvarez et al., 2011). Next, we describe how we utilise a GP emulator within the Parareal framework.

5.1.2 Our approach

In Parareal, we know that the PC (2.9c) updates the solutions U_n^k using a correction term based on information (from the solvers $\mathcal{F}_{\Delta T}$ and $\mathcal{G}_{\Delta T}$) calculated during the *previous* iteration $k-1$. In a Markovian-like manner, all fine/coarse information about the solution obtained prior to iteration $k-1$ is ignored by the PC, a feature present in most Parareal-type algorithms and variants (Ait-Ameur et al., 2020; Dai et al., 2013; Elwasif et al., 2011; Maday and Mula, 2020), including SParareal. As mentioned before, the goal is to demonstrate that *all* of the fine and coarse solution information accumulated up to iteration k, i.e. the acquisition data, can be exploited to determine a solution in faster wallclock time than Parareal.

In GParareal, we propose the following update rule, again based on a coarse prediction and multi-fidelity correction, that instead refines solutions using information from the *current* iteration k, rather than $k-1$:

$$\begin{aligned} U_n^k &= \mathcal{F}_{\Delta T}(U_{n-1}^k) \\ &= (\mathcal{F}_{\Delta T} - \mathcal{G}_{\Delta T} + \mathcal{G}_{\Delta T})(U_{n-1}^k) \\ &= \underbrace{\mathcal{G}_{\Delta T}(U_{n-1}^k)}_{\text{prediction}} + \underbrace{(\mathcal{F}_{\Delta T} - \mathcal{G}_{\Delta T})(U_{n-1}^k)}_{\text{correction}} \quad 1 \leqslant k < n \leqslant N. \end{aligned} \tag{5.7}$$

If U_{n-1}^k is known, the prediction term is rapidly calculable, however the correction is not known explicitly without running $\mathcal{F}_{\Delta T}$ at expensive cost. We propose using a GP emulator to infer the correction term, which we know is computationally expensive

to simulate due to the presence of the $\mathcal{F}_{\Delta T}$ term. We can condition a GP prior over $\mathcal{F}_{\Delta T} - \mathcal{G}_{\Delta T}$ on *all* previously obtained evaluations of $\mathcal{F}_{\Delta T}$ and $\mathcal{G}_{\Delta T}$ to obtain a Gaussian posterior distribution from which we can extract an explicit value and carry out the refinement in (5.7).

The solutions generated by GPparareal should converge to the exact solution assuming the accuracy of the emulator increases as more acquisition data is obtained each iteration. The hope is that the GP predictions should be more accurate than those provided by Parareal. This approach could be particularly beneficial if one wishes to fully understand and evaluate the dynamics of (2.1) by simulating solutions for a range of initial values $\boldsymbol{u}^0$ or over different time intervals. Firstly, if one can obtain additional parallel speedup, generating such a sequence of independent simulations becomes more computationally tractable in feasible time. Secondly, the legacy data, i.e. batches of acquisition data accumulated between independent simulations, can be used to inform future simulations by increasing the size of the dataset available to the GP emulator prior to simulation. Being able to re-use (expensive) acquisition or legacy data to integrate IVPs in parallel is not something, to the best of our knowledge, that existing time-parallel algorithms can do.

5.1.3 Related work

As briefly mentioned in Chapter 1, learning-based methods have been previously developed to solve IVPs in PN (Hennig et al., 2022). The first ODE filters used GP regression[3] techniques to calculate a posterior probability distribution over the solution to an IVP at any $t \in [t_0, T]$—recall Figure 1.4(b). They achieved this by conditioning the GP on observation data, i.e. inexact solution and derivative evaluations from the IVP, obtained sequentially in time. Since then, modern ODE filters have moved away from GP regression methods, instead using Gauss-Markov processes (Øksendal, 2013) that can make use of more computationally efficient Kalman filters and Rauch-Tung-Striebel smoothers (Bosch et al., 2021; Särkkä, 2013; Schober et al., 2019; Tronarp et al., 2019; Wenger et al., 2021). Even though such methods are becoming computationally competitive (compared to classical methods) (Kersting et al., 2020; Krämer et al., 2022), running them sequentially over large time intervals or expensive vector fields is still a computationally intractable process.

While there has been some work on parallelising the implementation of Kalman filters and smoothers (Särkkä and Garciá-Fernández, 2021) as well as particle-based smoothers (Corenflos et al., 2022), i.e. de-sequentialised Monte Carlo, it does not seem as though these methods have yet been deployed within the ODE filter framework. Although this would be of interest, we instead harness ideas from the early PN ODE

[3]GP regression models condition a GP prior using observations that contain uncertainty, i.e. are corrupted by statistical noise (not necessarily Gaussian). GP emulators, on the other hand, condition on observations that are assumed to have no uncertainty, i.e. are noise-free.

filters in a slightly different way by modelling the *residual* between solutions provided by the deterministic fine and coarse solvers in Parareal rather than modelling the solution to the IVP itself. This is mainly because the filters and smoothers used in the PN ODE solvers handle time series data that arrives sequentially whereas the acquisition data in Parareal, which arrives in batches each iteration, does not depend on time. While the method proposed in this chapter *will not* return a probabilistic solution to (2.1) like the sequential PN ODE filters do, we believe that it constitutes a positive step in this direction.

Within the PinT field, the first work investigating whether acquisition data could be used to improve solutions generated by the PC was *Krylov-enhanced Parareal*, introduced by Gander and Petcu (2008). The idea was to construct a projection operator, using a Krylov-subspace spanned by the PC solution set, to replace the coarse solver—of which the accuracy should increase as more data is accumulated. This variant was designed to deal with the slow convergence observed when solving hyperbolic (non-diffusive) IVPs (Gander and Vandewalle, 2007; Ruprecht, 2018). The Krylov subspace approach was shown to work well for a linear acoustic-advection system by Ruprecht and Krause (2012) and was extended to work for nonlinear problems (slightly outside the Parareal framework) by Cortial and Farhat (2009). Rather than replacing the coarse solver and using the PC solution dataset, as the aforementioned works do, we will make use of the fine and coarse solution dataset to replace the correction term, which should hopefully be easier to model. In addition, the (serial) cost incurred by constructing a new coarse solver grows as more acquisition data is accumulated and is something we need to bear in mind when analysing the computational complexity of GParareal.

To further resolve issues created by the coarse solver in Parareal (e.g. numerical instabilities, accuracy, and slow convergence) work has begun on replacing the coarse solver with neural networks (NNs). Yalla and Engquist (2018) use a NN to emulate $\mathcal{F}_{\Delta T}$ using (legacy) data obtained by propagating M training points (either randomly selected or guided by an initial coarse solve) with $\mathcal{F}_{\Delta T}$ (in parallel). Numerical experiments show that linear systems can be simulated in one iteration (something we will discuss further in Chapter 6) and nonlinear systems in fewer iterations than Parareal (if M is sufficiently large) when they use the NN in place of $\mathcal{G}_{\Delta T}$. These results, however, make no mention of the computational costs (and the number of processors) required to generate the training data or train the NN. Similar findings are reported in Agboh et al. (2020) when applying the same methodology to IVPs in robotic manipulation. In Nguyen and Tsai (2022), a NN is constructed to learn a mapping from the coarse to the fine solution space and then used in place of $\mathcal{G}_{\Delta T}$. With the aim of solving wave equations, numerical results suggest that the NNs trained using acquisition data (obtained throughout the simulation) were more beneficial for faster convergence than pre-generated (legacy) data sampled randomly

throughout the state space. On a similar but different track, it is worth mentioning the work of Lee et al. (2022), in which Parareal is used to speedup the training process of a NN, something that could be worth considering in the aforementioned works where the (almost certainly significant) serial training costs were ignored. These findings demonstrate that learning-based methods can indeed be used to accelerate the convergence of Parareal, however, much work still needs to be done to assess the cost of training, optimising, and tuning the NNs and determine whether they are actually viable in large-scale PinT simulations.

Also worth briefly mentioning are the use of physics-informed neural networks (PINNs) within the PinT framework. PINNs approximate the solution to IVPs by minimising a loss function comprised of the equations being solved, the initial/boundary conditions, and any other known information of the system (e.g. conservation laws). Training data is sampled randomly throughout the domain, however, the choice of these data points, the NN architecture, and optimisation process has a large impact on performance and accuracy—see Cuomo et al. (2022) for an overview. So far, PINNs have been used to solve a number of time-dependent PDE problems and are known to become prohibitively expensive if one tries to solve over a sufficiently long time interval using large amounts of training data. Meng et al. (2020) develop a *Parareal PINN* that parallelises a PINN solver, using smaller PINNs as the coarse and fine solvers within the classic Parareal algorithm. The results are encouraging, however, the test problems presented are very simple, converging in one or two Parareal iterations and it is assumed that simplified ODEs/PDEs are available for the coarse solver—which is not always the case in practical applications. Whereas the Parareal PINN scheme uses Parareal to parallelise a PINN solver, we are more interested in how PINNs could be used to accelerate convergence of the Parareal scheme itself. In Ibrahim et al. (2023), a PINN is used in place of the coarse solver, with the idea being that a pre-trained PINN is faster to evaluate (over short time intervals) and provides more accurate solutions compared to a standard coarse solver. Whilst reporting runtimes faster than Parareal, on the order of milliseconds, the (offline) pre-training time of the PINN (which took 30 minutes using $\mathcal{O}(10^5)$ training points) was not accounted for (as in the previous studies). This is, however, encouraging as the expensive pre-training could be justified in cases where one wishes to solve a given IVP for many different initial conditions, saving overall computational time—though this warrants further numerical investigation.

Most of the approaches discussed have replaced the coarse solver with a learning-based method trained using the fine or PC solution data, with the aim of making the coarse predictions in Parareal more accurate. This amounts to trying to learn the nonlinear vector field generated by the fine solver $\mathcal{F}_{\Delta T}$, which we expect to be more difficult (and perhaps requires more solution data) than trying to emulate the correction $\mathcal{F}_{\Delta T} - \mathcal{G}_{\Delta T}$. This is because we expect the output length scale of

$\mathcal{F}_{\Delta T} - \mathcal{G}_{\Delta T}$ to be small, on the order of the difference between the local truncation error of each solver, and therefore to vary smoothly with the input states. As reported by Nguyen and Tsai (2022), we expect acquisition data to be most effective in training the GP emulator, however, we will also investigate the effect of using legacy data from prior simulations and random sampling on convergence and speedup. We will also account for the serial costs that training and optimising the GP emulator will have on the wallclock time and parallel speedup estimates of GPararealI. Using the GP emulator will also allow us to derive an error bound for GPararealI solutions, something more difficult to do when working with NNs which depend on the choices of the numbers of neurons, layers, activation functions, and so on.

5.2 The algorithm

We are now ready to explain how GPararealI works.

5.2.1 How it works

Before solving (2.1), we define a GP prior over the unknown "correction function" $\mathcal{F}_{\Delta T} - \mathcal{G}_{\Delta T}$ just like we did in Section 5.1.1. This function maps an initial value $x_n \in \mathcal{U}$ at time t_n to the residual difference between $\mathcal{F}_{\Delta T}(x_n)$ and $\mathcal{G}_{\Delta T}(x_n)$ at time t_{n+1}. More formally, we write

$$\mathcal{F}_{\Delta T} - \mathcal{G}_{\Delta T} \sim \mathcal{GP}(m, \kappa), \tag{5.8}$$

with mean function $m \colon \mathcal{U} \to \mathbb{R}$ and covariance kernel $\kappa \colon \mathcal{U} \times \mathcal{U} \to \mathbb{R}$. Given the vectors $\boldsymbol{x}, \boldsymbol{x}' \in \mathcal{U}^J$, the corresponding mean vector is denoted $\mu(\boldsymbol{x}) = (m(x_j))_{j=0,\dots,J-1}$ and the covariance matrix $K(\boldsymbol{x}, \boldsymbol{x}') = (\kappa(x_i, x'_j))_{i,j=0,\dots,J-1}$. The correction term is expected to be small, on the order of the difference between the accuracy of $\mathcal{F}_{\Delta T}$ and $\mathcal{G}_{\Delta T}$, hence we define a zero-mean process, i.e. $m(x) = 0 \; \forall x \in \mathcal{U}$. We are free to select any appropriate covariance kernel based on any prior knowledge of the solution to (2.1), e.g. regularity/periodicity. However, assuming we have no information *a priori* to simulation, we will select the SE kernel (5.2) [4]. The kernel hyperparameters denoting the length scales ℓ^2 and σ^2 are referred to collectively in the vector $\boldsymbol{\theta}$ and need to be optimised during the simulation. Having initialised the GP emulator, the algorithm proceeds as follows (see Algorithm 3 for pseudocode).

[4] Note that we did analyse the use of alternate Matern kernels for the IVPs tested, however, they yielded significantly poorer performance, see Murphy (2022, Sec. 17.1.2).

Algorithm 3: GParareal

Initialise: Set counters $k = I = 0$ and define U_n^k, $\hat{U}_n^k$ and $\tilde{U}_n^k$ as the refined, coarse, and fine solutions at the nth time step and kth iteration respectively (note $U_0^k = \hat{U}_0^k = \tilde{U}_0^k = u^0 \ \forall k$). If known, initialise any legacy $\mathcal{F}_{\Delta T} - \mathcal{G}_{\Delta T}$ input data $\boldsymbol{x}$, output data $\boldsymbol{y}$, and hyperparameters $\boldsymbol{\theta}$.

%Calculate initial values at each t_n by running $\mathcal{G}_{\Delta T}$ serially.

for $n = 1$ **to** N **do**
  $\hat{U}_n^0 = \mathcal{G}_{\Delta T}(\hat{U}_{n-1}^0)$;
  $U_n^0 = \hat{U}_n^0$;
end
for $k = 1$ **to** N **do**
  %Propagate refined solutions (from iteration $k-1$) on unconverged time slices by running $\mathcal{F}_{\Delta T}$ in parallel.
  for $n = I + 1$ **to** N **do**
    $\tilde{U}_n^{k-1} = \mathcal{F}_{\Delta T}(U_{n-1}^{k-1})$;
  end
  $I = I + 1$;
  $U_I^k = \tilde{U}_I^{k-1}$ for all k ; %copy converged solution at t_I to future k.
  $\boldsymbol{x} = \text{append}(\boldsymbol{x}, (U_I^{k-1}, \ldots, U_{N-1}^{k-1})^{\mathrm{T}})$; %collect new input data.
  $\boldsymbol{y} = \text{append}(\boldsymbol{y}, (\tilde{U}_{I+1}^{k-1} - \hat{U}_{I+1}^{k-1}, \ldots, \tilde{U}_N^{k-1} - \hat{U}_N^{k-1})^{\mathrm{T}})$; %collect new output data.
  $\boldsymbol{\theta} = \text{GPoptimise}(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\theta})$; %optimise hyperparameters.
  %Propagate refined solution (at iteration k) with $\mathcal{G}_{\Delta T}$, then correct using the expected value of the GP prediction (5.12) (this step *cannot* be carried out in parallel).
  for $n = I + 1$ **to** N **do**
    $x^\star = U_{n-1}^k$;
    $\hat{U}_n^k = \mathcal{G}_{\Delta T}(x^\star)$;
    $y^\star = \text{GPpredict}(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\theta}, x^\star)$; %returns Gaussian random variable
    $U_n^k = \hat{U}_n^k + \mathbb{E}[y^\star]$;
  end
  %Evaluate stopping criterion, saving all solutions up to t_I.
  $I = \max_{n \in \{I, \ldots, N\}} |U_i^k - U_i^{k-1}| < \varepsilon \quad \forall i < n$;
  if $I = N$ **then**
    return k, $\boldsymbol{U}^k$, $\boldsymbol{x}$, $\boldsymbol{y}$, $\boldsymbol{\theta}$; %if tolerance met, stop.
  end
end

Iteration $k = 0$

Firstly, run $\mathcal{G}_{\Delta T}$ sequentially from the exact initial value U_0^0, on a single processor, to locate the coarse solutions

$$U_n^0 = \mathcal{G}_{\Delta T}(U_{n-1}^0) \quad \text{for} \quad n = 1, \ldots, N. \tag{5.9}$$

Store these solutions in the vector $\boldsymbol{x} \coloneqq (U_0^0, \ldots, U_{N-1}^0)^\intercal$ for use in the GP emulator.

Iteration $k = 1$

Use $\mathcal{F}_{\Delta T}$ to propagate the values in (5.9) on each time slice in *parallel* (using the N processors) to obtain the following values at t_n

$$\mathcal{F}_{\Delta T}(U_{n-1}^0) \quad \text{for} \quad n = 1, \ldots, N. \tag{5.10}$$

At this stage, we diverge from the Parareal method. Given $\boldsymbol{x}$, store the values of $\mathcal{F}_{\Delta T} - \mathcal{G}_{\Delta T}$, using (5.9) and (5.10), in the vector

$$\boldsymbol{y} \coloneqq \big((\mathcal{F}_{\Delta T} - \mathcal{G}_{\Delta T})(x_n)\big)_{n=0,\ldots,N-1}^\intercal. \tag{5.11}$$

At this point, the inputs $\boldsymbol{x}$ and evaluations $\boldsymbol{y}$ are used to optimise the kernel hyperparameters $\boldsymbol{\theta}$ via maximum likelihood estimation—see Section 5.2.2. Conditioning the prior (5.8) on the acquisition data $\{\boldsymbol{x}, \boldsymbol{y}\}$, the GP posterior over $(\mathcal{F}_{\Delta T} - \mathcal{G}_{\Delta T})(x')$, where $x' \in \mathcal{U}$ is some initial value in the state space, is given by

$$(\mathcal{F}_{\Delta T} - \mathcal{G}_{\Delta T})(x') \mid \{\boldsymbol{x}, \boldsymbol{y}\} \sim \mathcal{N}\big(\hat{\mu}(x'), \hat{K}(x', x')\big), \tag{5.12}$$

where $\hat{\mu}(x')$ and $\hat{K}(x', x')$ are given by (5.5) and (5.6), respectively.

Now we wish to determine updated solutions U_n^1 at each time step. Given $\mathcal{F}_{\Delta T}$ has been run once, the exact solution is known at time t_1. Specifically, at t_0 we know $U_0^k = U_0 \ \forall k \geqslant 0$ and at t_1 we know $U_1^k = U_1 = \mathcal{F}_{\Delta T}(U_0^1) \ \forall k \geqslant 1$. At t_2, the exact solution $U_2 = \mathcal{F}_{\Delta T}(U_1^1)$ is unknown, hence we need to calculate its value without running $\mathcal{F}_{\Delta T}$ again. To do this, we re-write the exact solution using (5.7):

$$U_2^1 = \underbrace{\mathcal{G}_{\Delta T}(U_1^1)}_{\text{prediction}} + \underbrace{(\mathcal{F}_{\Delta T} - \mathcal{G}_{\Delta T})(U_1^1)}_{\text{correction}}. \tag{5.13}$$

Both terms in (5.13) are initially unknown, however, the prediction can be calculated rapidly at low cost while the correction can be inferred using the GP posterior (5.12) with $x' = U_1^1$. Therefore, we obtain a Gaussian distribution over the solution

$$U_2^1 \sim \mathcal{N}\big(\mathcal{G}_{\Delta T}(U_1^1) + \hat{\mu}(U_1^1), \hat{K}(U_1^1, U_1^1)\big), \tag{5.14}$$

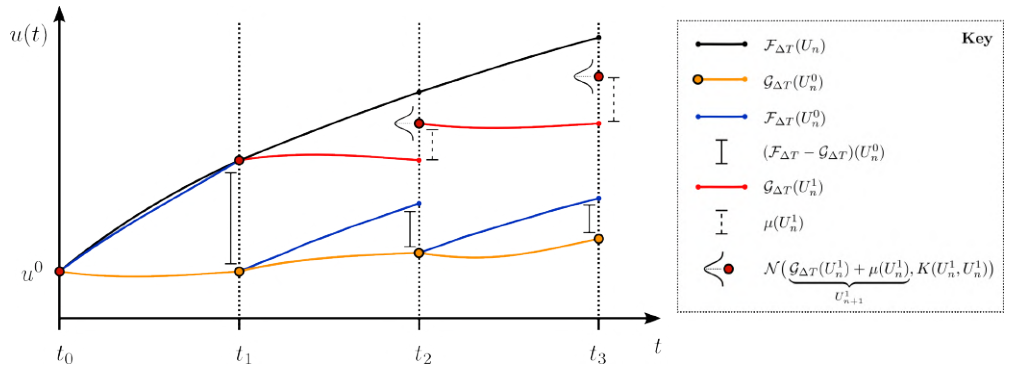


Figure 5.2: Schematic of the first iteration of GParareal. The 'exact' solution over $[t_0, t_3]$ is shown in black, with the first coarse and fine (parallel) runs given in yellow and blue respectively. Solid bars represent the residual between these solutions (5.11). The predictions, i.e. the second coarse runs, are shown in red and the corresponding corrections from the GP emulator are represented by the dashed bars. The updated solutions (5.15) at the end of the iteration are represented by the red dots. Note the black and blue lines in $[t_0.t_1]$ should overlap but have been shown not to for clarity.

with variance stemming from uncertainty in the GP emulator. Repeating this process to determine a distribution for the solution at t_3 by attempting to propagate the random variable U_2^1 using $\mathcal{G}_{\Delta T}$ is computationally infeasible for nonlinear IVPs. To tackle this and be able to propagate U_2^1, we approximate the distribution by taking its mean value, setting

$$U_2^1 = \mathcal{G}_{\Delta T}(U_1^1) + \hat{\mu}(U_1^1).$$

This approximation is a convenient way of minimising computational cost in the PC step, at a price of ignoring uncertainty in the GP emulator—we discuss possible alternatives in Section 5.5.

The update process, applying (5.7) and then approximating the Gaussian distribution by taking its expectation, is repeated sequentially for later t_n, yielding the approximate solutions

$$U_n^1 = \mathcal{G}_{\Delta T}(U_{n-1}^1) + \hat{\mu}(U_{n-1}^1) \quad \text{for} \quad n = 3, \ldots, N. \tag{5.15}$$

This process is illustrated in Figure 5.2. Finally, we impose stopping criteria (2.10), identifying which U_n^1 for $n \leqslant I$ have converged. Using the same stopping criteria as Parareal will allow us to compare the performance of both algorithms in Section 5.3.

Iteration $k \geqslant 2$

If the stopping criteria is not met, i.e. $I < N$, we can iteratively update any unconverged solutions by re-applying the previous steps. This means calculating

$\mathcal{F}_{\Delta T}(U_n^{k-1})$, $n = I, \ldots, N-1$, in parallel and then storing new evaluations $\hat{\boldsymbol{y}} = \big((\mathcal{F}_{\Delta T} - \mathcal{G}_{\Delta T})(U_n^{k-1})\big)_{n=I,\ldots,N-1}^{\intercal}$, with corresponding inputs $\hat{\boldsymbol{x}} = (U_I^{k-1}, \ldots, U_{N-1}^{k-1})^{\intercal}$. Hyperparameters are then re-optimised and the GP is re-conditioned using *all* prior acquisition data, i.e. $\boldsymbol{x} = [\boldsymbol{x}; \hat{\boldsymbol{x}}]$ and $\boldsymbol{y} = [\boldsymbol{y}; \hat{\boldsymbol{y}}]$, generating an updated posterior[5]. The update rule is then applied such that we obtain

$$U_n^k = \mathcal{G}_{\Delta T}(U_{n-1}^k) + \hat{\mu}(U_{n-1}^k) \quad \text{for} \quad n = I+2, \ldots, N.$$

Once $I = N$, the solution, the number of iterations k taken to converge, and the acquisition data $\boldsymbol{x}$ and $\boldsymbol{y}$ are returned. Recall that the acquisition data can be used in future GParareal simulations (as "legacy data") to provide the GP emulator with more data and therefore exploit additional speedup—this will be demonstrated in Section 5.3. For completeness, we fully define the GParareal scheme in the same way as Parareal and SParareal.

Definition 5.1 (GParareal)**.** For two numerical flow maps $\mathcal{F}_{\Delta T}$ and $\mathcal{G}_{\Delta T}$ (described in Section 2.2), and the GP emulator described before, the GParareal scheme is given by

$$U_0^0 = u^0, \tag{5.16a}$$
$$U_{n+1}^0 = \mathcal{G}_{\Delta T}(U_n^0), \qquad 0 \leqslant n \leqslant N-1, \tag{5.16b}$$
$$U_{n+1}^{k+1} = \mathcal{G}_{\Delta T}(U_n^{k+1}) + \hat{\mu}(U_n^{k+1}) \qquad 0 \leqslant k \leqslant n \leqslant N-1. \tag{5.16c}$$

5.2.2 Kernel hyperparameter optimisation

The hyperparameters $\boldsymbol{\theta}$ of the kernel κ will need to be optimised in light of the acquisition data $\boldsymbol{y}$ (and corresponding input data $\boldsymbol{x}$). We optimise each element of $\boldsymbol{\theta}$ such that it maximises its (log) marginal likelihood (Rasmussen, 2004). To do this, first define $g(x) \coloneqq (\mathcal{F}_{\Delta T} - \mathcal{G}_{\Delta T})(x)$ and $\boldsymbol{g} \coloneqq (g(x_j))_{j=0,\ldots,N-1}^{\intercal}$. Recall N is the length of $\boldsymbol{x}$ (and $\boldsymbol{y}$) during the first iteration. This length will increase as more data is accumulated each iteration but the following optimisation process will remain the same. Given the evaluations $\boldsymbol{y}$ are noise-free, the likelihood of obtaining such data is $p(\boldsymbol{y}|\boldsymbol{g}, \boldsymbol{x}, \boldsymbol{\theta}) = \delta(\boldsymbol{y} - \boldsymbol{g})$, where $\delta(\cdot)$ is the multidimensional Dirac delta function. All this says is that our emulator is interpolating the acquisition data, i.e. $\hat{\mu}(x_j) = y_j$ and $\hat{K}(x_j, x_j) = 0$. The marginal likelihood, given $\boldsymbol{x}$ and $\boldsymbol{\theta}$, is therefore

$$\begin{aligned} p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{\theta}) &= \int \underbrace{p(\boldsymbol{y}|\boldsymbol{g}, \boldsymbol{x}, \boldsymbol{\theta})}_{\text{likelihood}} \underbrace{p(\boldsymbol{g}|\boldsymbol{x}, \boldsymbol{\theta})}_{\text{prior}} \,\mathrm{d}\boldsymbol{g} \\ &= \int \delta(\boldsymbol{y} - \boldsymbol{g}) \mathcal{N}(\boldsymbol{g}|\boldsymbol{0}, K(\boldsymbol{x}, \boldsymbol{x})) \,\mathrm{d}\boldsymbol{g} = \mathcal{N}(\boldsymbol{y}|\boldsymbol{0}, K(\boldsymbol{x}, \boldsymbol{x})), \end{aligned}$$

[5]Here, $[\boldsymbol{a}; \boldsymbol{b}]$ denotes the vertical concatenation of column vectors $\boldsymbol{a}$ and $\boldsymbol{b}$.

where $\mathcal{N}(\boldsymbol{y}|\boldsymbol{0}, K(\boldsymbol{x},\boldsymbol{x}))$ denotes the probability density of a multivariate Gaussian distribution (3.6) evaluated at $\boldsymbol{y}$, with mean vector $\boldsymbol{0}$ and covariance matrix $K(\boldsymbol{x},\boldsymbol{x})$ that depends on $\boldsymbol{\theta}$ (recall (5.2)). Taking the logarithm, we want to maximise the log-marginal likelihood, i.e. find

$$\arg\max_{\boldsymbol{\theta}} \big[\log p(\boldsymbol{y}|\boldsymbol{x},\boldsymbol{\theta})\big] = \arg\max_{\boldsymbol{\theta}} \Big[-\frac{1}{2}\boldsymbol{y}^{\mathrm{T}}[K(\boldsymbol{x},\boldsymbol{x})]^{-1}\boldsymbol{y} - \frac{1}{2}\log|K(\boldsymbol{x},\boldsymbol{x})| - \frac{N}{2}\log 2\pi\Big],$$

where $|K(\boldsymbol{x},\boldsymbol{x})|$ is the determinant of $K(\boldsymbol{x},\boldsymbol{x})$. The hyperparameters in $\boldsymbol{\theta}$ can be estimated numerically using any preferred iterative optimisation routine and is done once per iteration—initialised using hyperparameters from the prior iteration. Given this is a serial computation, we can save runtime in later iterations by stopping the optimisation process when hyperparameters do not change significantly between iterations—this is implemented in the numerical experiments in Section 5.3.

5.2.3 Computational complexity

The complexity of GParareal can be calculated similarly to that of Parareal—refer back to Section 2.2.3 for notation. In GParareal, an additional cost is incurred when (serially) conditioning the emulator on acquisition/legacy data and optimising the hyperparameters. During the kth iteration, up to kN evaluations of $\mathcal{F}_{\Delta T} - \mathcal{G}_{\Delta T}$ have been collected, hence standard cubic complexity GP conditioning scales like $\mathcal{O}(k^3N^3)$ in terms of FLOPs (similarly, if not higher for the hyperparameters). For fixed N, let $T_{\mathrm{GP}}(k)$ represent the wallclock time taken to condition and optimise hyperparameters of the GP (using up to kN observations) at iteration k. Note this is a strictly increasing function of k (assuming hyperparameter optimisation is not stopped beyond some iteration to save compute time). Ignoring negligible serial overheads, we can write down the total wallclock time (assuming k iterations) for GParareal as

$$\begin{aligned} T_{\mathrm{GPara}} &\approx NT_{\mathcal{G}} + \sum_{i=1}^{k}\big(T_{\mathcal{F}} + (N-i)T_{\mathcal{G}} + T_{\mathrm{GP}}(i)\big) \\ &= kT_{\mathcal{F}} + (k+1)\left(N - \frac{k}{2}\right)T_{\mathcal{G}} + T_{\mathrm{GP}}, \end{aligned} \tag{5.17}$$

where $T_{\mathrm{GP}} \coloneqq \sum_{i=1}^{k} T_{\mathrm{GP}}(i)$. The approximate parallel speedup is then given by

$$S_{\mathrm{GPara}} \approx \left[\frac{k}{N} + (k+1)\left(1 - \frac{k}{2N}\right)\frac{T_{\mathcal{G}}}{T_{\mathcal{F}}} + \frac{1}{N}\frac{T_{\mathrm{GP}}}{T_{\mathcal{F}}}\right]^{-1}. \tag{5.18}$$

For completeness, the parallel efficiency is given by

$$E_{\mathrm{GPara}} \approx \frac{S_{\mathrm{GPara}}}{N} = \left[k + (k+1)\big(N - \frac{k}{2}\big)\frac{T_{\mathcal{G}}}{T_{\mathcal{F}}} + \frac{T_{\mathrm{GP}}}{T_{\mathcal{F}}}\right]^{-1}. \tag{5.19}$$

Therefore, in addition to the Parareal requirements that $k \ll N$ and $T_{\mathcal{G}} \ll T_{\mathcal{F}}$, GParareal requires that $T_{\text{GP}} \ll T_{\mathcal{F}}$ in order to maximise parallel speedup. If this is the case, the complexity of GParareal is approximately the same as Parareal.

This simple analysis suggests that if k and/or N are large, then the cost of the emulation may begin to dominate that of the fine solver, limiting the parallel speedup from GParareal—see Section 5.3 for an example of this effect. This, however, need not hinder the usability of GParareal for a number of reasons. Firstly, time-parallelisation is typically deployed on problems where additional parallel speedup is needed beyond that achieved by traditional domain decomposition, i.e. on spatio-temporal PDEs. This means that if P processors are required for the space-parallel computations of the PDE and N processors for the time-parallel computations, then NP processors are required in total. For moderate to large values of P, only leftover HPC resources are available to exploit time-parallelism and so N typically cannot be chosen very large, somewhat limiting how large T_{GP} will be. Secondly, in the scenario that both T_{GP} and $T_{\mathcal{F}}$ are small, one does not need to use a time-parallel method in the first place, as $\mathcal{F}_{\Delta T}$ can simply be run serially in this case. Thirdly, if both T_{GP} and $T_{\mathcal{F}}$ are large or of a similar order, then one can reduce T_{GP} by reducing the number of time slices N, thereby increasing $T_{\mathcal{F}}$ at the same time. We will assess these ideas in Section 5.3.

Whilst there is no way to control the final value of k obtained by either Parareal or GParareal, there are ways of reducing T_{GP} using more efficient non-cubic complexity, emulation methods. For example, one could make use of sparse GPs, parallel matrix inversion methods, or sparse approximate linear algebra techniques (Schäfer et al., 2021) to reduce the cost of evaluating the inverse kernel matrix $[K(\boldsymbol{x}, \boldsymbol{x})]^{-1}$. One could also reduce T_{GP} by clustering the input data points and training 'local' GPs in parallel (Snelson and Ghahramani, 2007) or instead use inducing points to average over input data points that are located close together in state space (Quiñonero Candela and Rasmussen, 2005; Snelson and Ghahramani, 2006)—see Murphy (2023) for additional methods. To reduce the cost of hyperparameter optimisation, one may deploy parallel optimisation routines if available or, as we implement in Section 5.3, stop the optimisation once additional data no longer improves the hyperparameter estimates.

5.2.4 Error bound analysis

In this section, we are interested in analysing the absolute error

$$e_n^k \coloneqq |U_n - U_n^k|, \tag{5.20}$$

between the exact solution and the GParareal solution at iteration k and time t_n. We show that this error has an upper bound proportional to the *fill distance* of the

dataset at iteration k (defined below). To do this, we now denote the input dataset at iteration k as $\boldsymbol{x}^k$ rather than $\boldsymbol{x}$ (because the dataset size strictly increases with each iteration of GParareal) and, similarly, denote the output dataset $\boldsymbol{y}$ as $\boldsymbol{y}^k$. We will make use of the same assumptions on the solvers $\mathcal{F}_{\Delta T}$ and $\mathcal{G}_{\Delta T}$ that were introduced in the SParareal error analysis and a known result on the consistency of the GP posterior mean $\hat{\mu}$ (5.5) to the true correction function $g = \mathcal{F}_{\Delta T} - \mathcal{G}_{\Delta T}$.

Preparatory assumptions and results

We begin by recalling the assumptions made on the solvers from Section 4.2. Namely, that $\mathcal{F}_{\Delta T}$ is assumed to be the exact solver (Assumption 4.2) and $\mathcal{G}_{\Delta T}$ is a one-step method with uniform local truncation error $\mathcal{O}(\Delta T^{p+1})$ (Assumption 4.3) and a Lipschitz condition (Assumption 4.4).

Next, we define the concepts required to state a result on the consistency of the GP posterior mean—definitions taken from Stuart and Teckentrup (2018). Firstly, define the *fill distance* $h_{\boldsymbol{x}^k}$ as the largest smallest distance between any point $v \in \mathcal{U}$ and any point $x_i \in \boldsymbol{x}^k$, i.e.

$$h_{\boldsymbol{x}^k} \coloneqq \sup_{v \in \mathcal{U}} \inf_{x_i \in \boldsymbol{x}^k} |v - x_i|.$$

It should be clear that each $x_i \in \boldsymbol{x}^k$ is also contained in $\mathcal{U}$ with the intuition being that $h_{\boldsymbol{x}^k}$ is the maximum distance any point $v \in \mathcal{U}$ can be from one in $\boldsymbol{x}^k$. In Figure 5.3, we illustrate the fill distance over a unit interval that contains data points $\boldsymbol{x}^k = (x_1, \dots, x_5)^\intercal$. If we were to obtain a new data point x_6 located somewhere in the interval $[0.1, 0.5]$, $h_{\boldsymbol{x}^k}$ would decrease, however, if located outside of $[0.1, 0.5]$ it would remain unchanged (i.e. $h_{\boldsymbol{x}^k}$ is non-increasing as more data points are added).

Secondly, we define a Hilbert space $H_\kappa(\mathcal{U})$ of functions $g \colon \mathcal{U} \to \mathbb{R}$, with inner product $\langle \cdot, \cdot \rangle_{H_\kappa(\mathcal{U})}$, as a *reproducing kernel Hilbert space* (RKHS) corresponding to a symmetric, positive-definite kernel $\kappa \colon \mathcal{U} \times \mathcal{U} \to \mathbb{R}$ if

i. $\forall v \in \mathcal{U}$, the kernel $\kappa(v, v') \in H_\kappa(\mathcal{U})$ as a function of its second argument,

ii. $\forall v \in \mathcal{U}$ and $g \in H_\kappa(\mathcal{U})$, the inner product $\langle g, \kappa(v, \cdot) \rangle_{H_\kappa(\mathcal{U})} = g(v)$ (reproducing property).

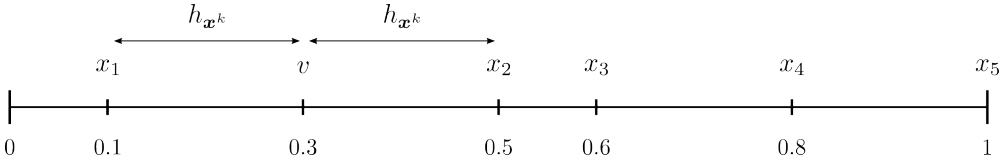


Figure 5.3: Illustration of the fill distance in one dimension where $\mathcal{U} = [0, 1] \subset \mathbb{R}$ and $\boldsymbol{x}^k = (0.1, 0.5, 0.6, 0.8, 1)^\intercal$. The fill distance is $h_{\boldsymbol{x}^k} = 0.2$.

A Hilbert space can be loosely thought of as a complete vector space $\mathcal{U}$ equipped with an inner product that defines a distance function. Less formally, the definition of the RKHS means that if two functions in the RKHS are "close" under a norm, then they are "close" point-wise as well. This is key to the following result on GP posterior mean consistency, adapted from Wendland (2004, Theorem 11.14), where it is assumed that the function we are trying to emulate (g) is in the RKHS $H_\kappa(\mathcal{U})$ where our kernel also exists.

Theorem 5.2 (GP posterior mean consistency)**.** *Suppose $\mathcal{U} \subset \mathbb{R}$ is a bounded interval and let κ be the SE kernel. Denote the GP posterior mean, built using $\boldsymbol{x}^k$, $\boldsymbol{y}^k$, and κ (5.5) as $\hat{\mu}$ and the function being emulated as $g \in H_\kappa(\mathcal{U})$. Then, for every $\tau \in \mathbb{N}$, there exist constants $h_0(\tau)$ and $C_\tau > 0$ such that*

$$|g(v) - \hat{\mu}(v)| \leqslant C_\tau h_{\boldsymbol{x}^k}^\tau |g|_{H_\kappa(\mathcal{U})} \quad \forall v \in \mathcal{U},$$

provided that $h_{\boldsymbol{x}^k} \leqslant h_0(\tau)$. Note that $|g|^2_{H_\kappa(\mathcal{U})} = \langle g, g \rangle_{H_\kappa(\mathcal{U})}$.

This result states that error of the GP posterior mean is proportional to the fill distance $h_{\boldsymbol{x}^k}$ (which ideally is small). The parameter τ can be chosen arbitrarily, however, if choosing a larger τ, the constants C_τ and $h_0(\tau)$ may increase and decrease, respectively—see Wendland (2004, Section 11.14) for further discussion. Also see Wendland (2004, Theorem 11.14) for a more general version of this result that holds for derivatives and when $\mathcal{U} \subset \mathbb{R}^d$. It should be noted that Theorem 5.2 only holds when $g \in H_\kappa(\mathcal{U})$, i.e. the function of interest lies within the RKHS of the SE kernel. If this is not the case, convergence issues may arise (see Karvonen (2022); Karvonen and Oates (2022)) and one would need to choose an alternative kernel function that reflects more of what is known about the structure of g. For consistency results involving Matérn kernels, see Stuart and Teckentrup (2018).

Error bound for GParareal solutions

Theorem 5.3 (GParareal error bound)**.** *Suppose the GParareal scheme (5.16) satisfies Assumptions 4.2, 4.3, and 4.4, and that the conditions required for Theorem 5.2 hold. Then, the absolute error (5.20) of the GParareal solution to the autonomous scalar ODE, i.e. $\boldsymbol{f}(t, \boldsymbol{u}(t)) \coloneqq f(u(t))$ in (2.1), at iteration k and time t_n satisfies*

$$e_n^k \leqslant \begin{cases} \Lambda_k \displaystyle\sum_{i=0}^{n-(k+1)} A^i & 1 \leqslant k < n \leqslant N, \\ 0 & 0 \leqslant n \leqslant k \leqslant N. \end{cases}$$

where $A = C_1 \Delta T^{p+1} + L_{\mathcal{G}}$ and $\Lambda_k = C_\tau h_{\boldsymbol{x}^k}^\tau |g|_{H_\kappa(\mathcal{U})}$.

Proof. First, consider the case $0 \leqslant n \leqslant k \leqslant N$. For $n = 0$, recall that $U_0^k = U_0 \; \forall k \geqslant$

0 by definition, hence $e_0^k = 0 \; \forall k \geqslant 0$. For $n = 1$, we seek $U_1^1 = \mathcal{F}_{\Delta T}(U_0^1)$ which we in fact know from applying $\mathcal{F}_{\Delta T}$ to U_0^0 during the prior iteration (i.e. $k = 0$). Therefore, we have that

$$\begin{aligned} U_1^1 = \mathcal{F}_{\Delta T}(U_0^1) = \mathcal{F}_{\Delta T}(U_0^0) = \mathcal{F}_{\Delta T}(U_0) = U_1 \quad &\Rightarrow \quad U_1^k = U_1 \; \forall k \geqslant 1 \\ &\Rightarrow \quad e_1^k = 0 \; \forall k \geqslant 1. \end{aligned}$$

We can repeat this process up to $n = N$ to show that

$$\begin{aligned} U_N^N = \mathcal{F}_{\Delta T}(U_{N-1}^N) = \mathcal{F}_{\Delta T}(U_{N-1}^{N-1}) = \mathcal{F}_{\Delta T}(U_{N-1}) = U_N \quad &\Rightarrow \quad U_N^k = U_N \; \forall k \geqslant N \\ &\Rightarrow \quad e_N^k = 0 \; \forall k \geqslant N. \end{aligned}$$

Now, consider the case $1 \leqslant k < n \leqslant N$. Using the update rule (5.16c), that $\mathcal{F}_{\Delta T}$ is the exact solver (4.5), and adding and subtracting the terms $g(U_n^k)$ and $\mathcal{G}_{\Delta T}(U_n)$, we can write

$$\begin{aligned} e_{n+1}^k = |U_{n+1} - U_{n+1}^k| &= |\mathcal{F}_{\Delta T}(U_n) - \big(\mathcal{G}_{\Delta T}(U_n^k) + \hat{\mu}(U_n^k)\big)| \\ &= |\mathcal{F}_{\Delta T}(U_n) - \big(\mathcal{G}_{\Delta T}(U_n^k) + \hat{\mu}(U_n^k)\big) \\ &\qquad + g(U_n^k) - g(U_n^k) + \mathcal{G}_{\Delta T}(U_n) - \mathcal{G}_{\Delta T}(U_n)|. \end{aligned}$$

Applying the triangle inequality and the definition of g, we obtain

$$\begin{aligned} e_{n+1}^k \leqslant |\big(\mathcal{F}_{\Delta T}(U_n) - \mathcal{G}_{\Delta T}(U_n)\big) - \big(\mathcal{F}_{\Delta T}(U_n^k) - \mathcal{G}_{\Delta T}(U_n^k)\big)| \\ + |\mathcal{G}_{\Delta T}(U_n) - \mathcal{G}_{\Delta T}(U_n^k)| + |g(U_n^k) - \hat{\mu}(U_n^k)|. \end{aligned}$$

On the right hand side, the first term can be bounded using (4.7), the second by (4.8), and the third using Theorem 5.2, yielding the recursion

$$e_{n+1}^k \leqslant A e_n^k + \Lambda_k,$$

where $A = C_1 \Delta T^{p+1} + L_{\mathcal{G}}$ and $\Lambda_k = C_\tau h_{\boldsymbol{x}^k}^\tau |g|_{H_\kappa(\mathcal{U})}$. This recursion can be solved using the initial condition $e_n^k = 0 \; \forall k \geqslant n$ to obtain the desired result. □

Theorem 5.3 shows that the error is proportional to the fill distance at iteration k and that GPareareal will recover the exact solution at time t_n after $k = n$ iterations. This result is rather general in the sense that we consider the fill distance with respect to the entire space $\mathcal{U} \subset \mathbb{R}$, whereas in reality we should measure the fill distance with respect to a moderately sized compact interval $\mathcal{V} \subset \mathcal{U}$ in which the solution $u(t)$ lies $\forall t \in [t_0, T]$. Essentially, the accuracy of the GP posterior mean outside of $\mathcal{V}$ is inconsequential to the GPareareal scheme because the mean will never be evaluated outside of $\mathcal{V}$. Also note, the result will generalise for GPareareal applied to systems of ODEs by using norms and the generalised version of Theorem 5.2 in Wendland

(2004). In the multivariate case ($d > 1$), however, a larger number of acquisition data points will be required to ensure the fill distance remains small. This is something we will need to bear in mind when applying GParareal to large systems of ODEs—a possible remedy for this situation is explored in Section 5.4.

5.2.5 Generalisation to ODE systems

The methodology in Section 5.2.1 can be generalised to solve systems of d autonomous ODEs. Accordingly, the correction term we wish to emulate would be vector-valued, i.e. $\mathcal{U} \subset \mathbb{R}^d$, hence we require a vector-valued (or multi-output) GP, rather than a scalar GP.

The simplest approach is to model each output of $\mathcal{F}_{\Delta T} - \mathcal{G}_{\Delta T}$ independently, whereby we use d scalar GPs (sharing the same vector-valued inputs in state space) to emulate each output. This requires initialising d GP emulators, each with their own covariance kernel κ_i (usually the same for consistency) and corresponding hyperparameters $\boldsymbol{\theta}_i$—to be optimised independently using their own respective observation datasets $\boldsymbol{y}^{(i)}$, $i = 1, \ldots, d$. In this case, the d GP emulators can be optimised independently of one another and so we make use of the idle processors to carry out these computations in an embarrassingly parallel fashion. This reduces the optimisation costs for d ODEs by a factor of d each iteration. We adopt this simple approach in our implementation of GParareal, however, it should be noted that this limits us to only being able to solve systems of $d \leqslant N$ ODEs, as we only have N processors available.

The more complex approach is to jointly emulate the outputs of $\mathcal{F}_{\Delta T} - \mathcal{G}_{\Delta T}$ by modelling cross-covariances between outputs via the method of co-kriging (Cressie, 1993). A number of co-kriging techniques exist (see Álvarez et al. (2011) for a brief overview), one of which is the linear model of coregionalisation that models the joint, block-diagonal, covariance prior between multiple outputs using a linear combination of the separate kernels κ_i. This is similar to when we considered correlated randoms variables in SParareal. Testing revealed that this method did not improve performance enough (i.e. there was no reduction in k) to justify the added computational complexity each iteration (results not reported). The d scalar output GPs discussed before have complexity $\mathcal{O}(k^3N^3)$ whereas these multi-output GPs have complexity $\mathcal{O}(d^3k^3N^3)$ (per iteration). Some applications may require correlated output dimensions, hence we note the methodology here for any interested readers.

As a final note, to solve nonautonomous systems (2.1), there are two possible approaches. One is to include time as an extra input dimension to each of the d scalar GPs—this requires a more carefully selected covariance kernel. The other way is to rewrite the d-dimensional nonautonomous system as a system of $d + 1$ autonomous equations and solve as described above—we do this in Section 5.3.3.

5.3 Numerical experiments: nonlinear ODEs

In this section, we present numerical experiments to compare the performance of GParareal and Parareal on a number of low-dimensional nonlinear ODE systems, namely the FitzHugh–Nagumo model, the chaotic Rössler system, a nonautonomous system, and the double pendulum system.

For simplicity, $\mathcal{F}_{\Delta T}$ and $\mathcal{G}_{\Delta T}$ are chosen to be explicit RK methods of order $q, p \in \{1, 2, 4, 8\}$, respectively ($q \geqslant p$). As before, $N_{\mathcal{F}}$ and $N_{\mathcal{G}}$ denote the number of time steps each solver uses over $[t_0, T]$. For these experiments we built our own cubic complexity GP emulator to highlight the effectiveness of GParareal using standard out-the-box methods, postponing the implementation of more efficient and sophisticated emulation methods to a future work. In the multivariate setting (recall Section 5.2.5), we use a scalar output GP emulator (with isotropic SE covariance kernel) to model each output dimension of $\mathcal{F}_{\Delta T} - \mathcal{G}_{\Delta T}$ and assign each one its own processor. Hyperparameter optimisation is carried out at each iteration, stopping when the (maximal) absolute difference between hyperparameters at successive iterations is smaller than 10^{-2}.

5.3.1 FitzHugh–Nagumo model

In this experiment, we consider the FitzHugh–Nagumo (FHN) model (FitzHugh, 1961; Nagumo et al., 1962) given by

$$\frac{\mathrm{d}u_1}{\mathrm{d}t} = c\big(u_1 - \frac{u_1^3}{3} + u_2\big), \quad \frac{\mathrm{d}u_2}{\mathrm{d}t} = -\frac{1}{c}(u_1 - a + bu_2), \tag{5.21}$$

and parameters $(a, b, c) = (0.2, 0.2, 3)$. We integrate (5.21) over $t \in [0, 40]$, dividing the interval into $N = 40$ slices, and set the tolerance for both GParareal and Parareal to $\varepsilon = 10^{-6}$. We use solvers $\mathcal{G}_{\Delta T} = \text{RK2}$ and $\mathcal{F}_{\Delta T} = \text{RK4}$ with $N_{\mathcal{G}} = 160$ and $N_{\mathcal{F}} = 1.6 \times 10^5$ steps, respectively.

In Figure 5.4(a), we solve (5.21) with initial condition $\boldsymbol{u}^0 = (-1, 1)^\intercal$ using both algorithms. Observe that the accuracy of GParareal is of approximately the same order as the solution obtained using Parareal—when comparing both to the serially obtained fine solution (Figure 5.4(b)). Note, however, that in Figure 5.4(c), GParareal takes six fewer iterations to converge to these solutions than Parareal does. As a result, GParareal locates a solution in faster wallclock time than Parareal, see Figure 5.4(d), with an almost 5× speedup vs. the serial solver—over twice the 2.4× speedup obtained by Parareal. Note that we increase $N_{\mathcal{F}}$ to 1.6×10^8 to ensure $\mathcal{F}_{\Delta T}$ is expensive to run and realise parallel speedup in Figure 5.4(d) (as both algorithms require $T_{\mathcal{G}}/T_{\mathcal{F}} \ll 1$).

To compare the convergence of both methods more broadly, we solve the FHN model (5.21) for a range of initial values. The heatmap in Figure 5.5(a) illustrates

how the convergence of Parareal is highly dependent, not just on the solvers in use, but also the initial values at $t = 0$, taking anywhere from 10 to 15 iterations to converge. For some initial values, Parareal does not converge at all, with solutions blowing up (returning `NaN` values) due to numerical instabilities generated by $\mathcal{G}_{\Delta T}$. In direct contrast, see Figure 5.5(b), GPararealconverges sooner and more uniformly due to the flexibility provided by the emulator, taking just five or six iterations to reach tolerance for all the initial values tested. This demonstrates how using an emulator can enable convergence even when $\mathcal{G}_{\Delta T}$ has low numerical stability.

Until now, GPararealsimulations have been carried out using only acquisition data. In Figure 5.6, we demonstrate how GPararealcan use both acquisition and legacy data to converge in fewer iterations than without the legacy data. Approximately $kN = 5 \times 40 = 200$ legacy data points, obtained solving (5.21) for $\boldsymbol{u}^0 = (-1, 1)^\intercal$,

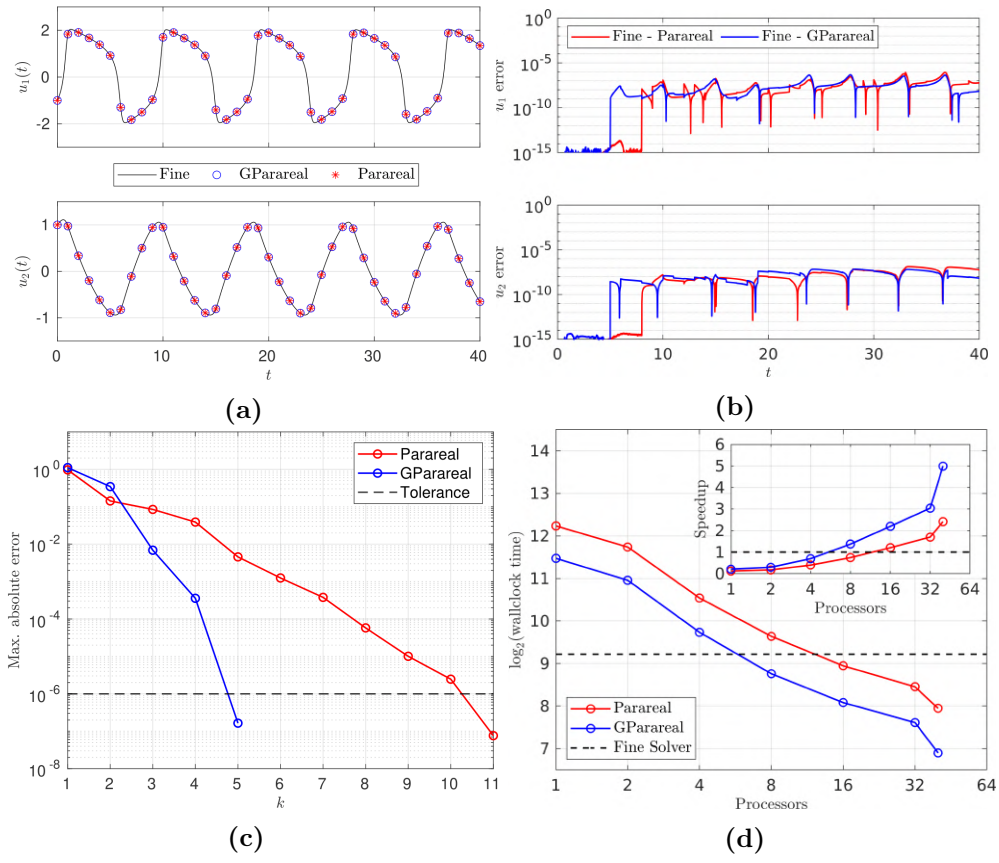
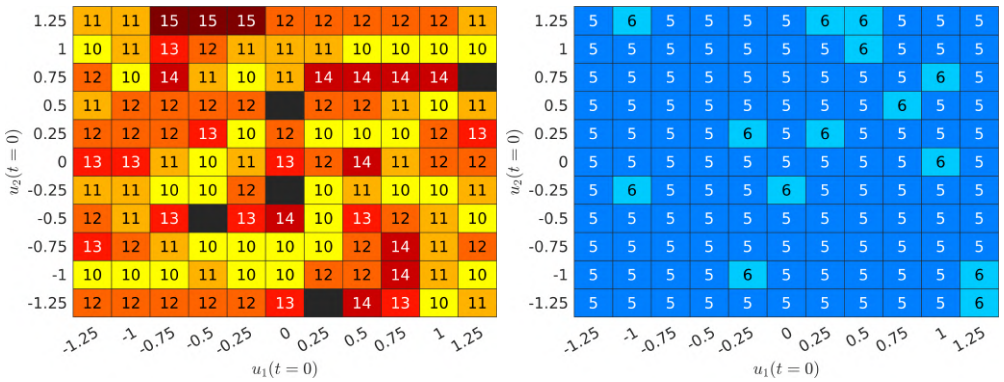


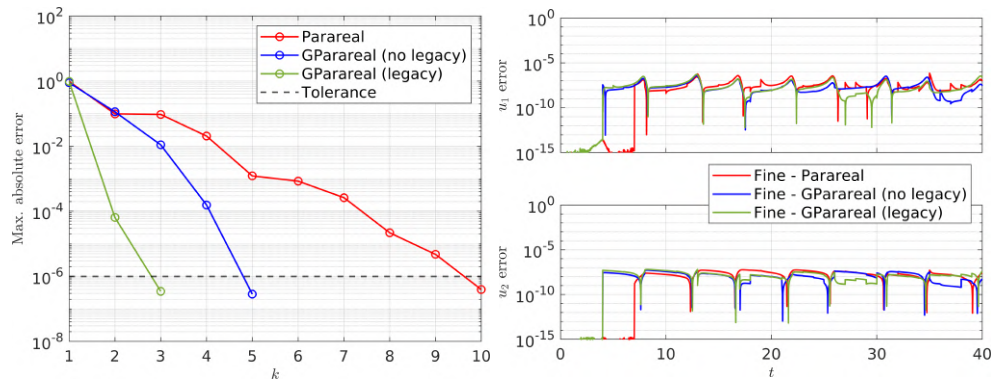
Figure 5.4: Numerical results obtained solving the FHN model (5.21) for $\boldsymbol{u}^0 = (-1, 1)^\intercal$. (a) Time-dependent solutions using the fine solver, GPararealand Parareal—both GPararealand Parareal plotted only at times $\boldsymbol{t}$ for clarity. (b) The corresponding absolute errors between solutions from GPararealand Parareal vs. the fine solution. (c) Maximum absolute errors (2.10) of each algorithm at successive iterations k until tolerance $\varepsilon = 10^{-6}$ is met. (d) Median wallclock times (taken over 5 runs) of both algorithms against the number of processors (up to 40). The inset plot displays the corresponding parallel speedup.



(a) Parareal **(b)** GParareal

Figure 5.5: Heat maps displaying the number of iterations taken until convergence k of (a) Parareal and (b) GParareal when solving the FHN model (5.21) for different initial values $\boldsymbol{u}^0 \in [-1.25, 1.25]^2$. Black boxes indicate where Parareal returned a `NaN` value during simulation.

are stored and made available to the GP emulator when solving (5.21) for alternate initial values $\boldsymbol{u}^0 = (0.75, 0.25)^\intercal$. In Figure 5.6(a), we can see that convergence takes two fewer iterations with the legacy data than without. The accuracy of the solutions obtained from these simulations is again shown to be of the order of the Parareal solution in both cases—see Figure 5.6(b). Repeating the experiment from Figure 5.5(b) with the same legacy data for a range of initial values we see that k is either unchanged or improved in all cases, see Figure 5.7. It should be noted



(a) **(b)**

Figure 5.6: Numerical simulations solving (5.21) for $\boldsymbol{u}^0 = (0.75, 0.25)^\intercal$ using GParareal with and without access to legacy data, i.e. $\mathcal{F}_{\Delta T} - \mathcal{G}_{\Delta T}$ data obtained solving (5.21) for $\boldsymbol{u}^0 = (-1, 1)^\intercal$. The Parareal simulation of the same problem is also shown for comparison. (a) Maximum absolute errors (2.10) against iteration number k until tolerance $\varepsilon = 10^{-6}$ met. (b) Time-dependent errors of the corresponding numerical solutions from each simulation vs. the fine solution.

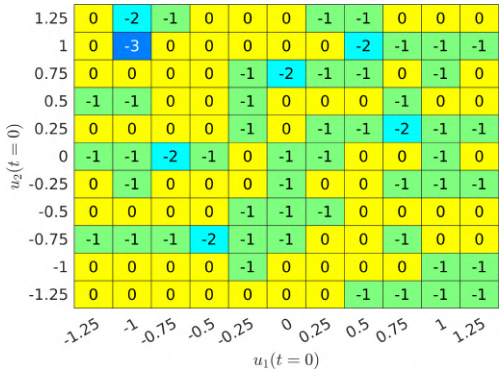


Figure 5.7: Heat map displaying the decrease in the number of iterations taken until convergence of GPareareal when solving (5.21) for different initial values $\boldsymbol{u}^0 \in [-1.25, 1.25]^2$ *with* legacy data compared to without, i.e. compared to Figure 5.5(b). Legacy data was obtained by solving (5.21) for $\boldsymbol{u}^0 = (-1, 1)^\intercal$.

that conditioning the GP and optimising hyperparameters using the legacy data comes at extra (serial) computational cost and checks should be made to ensure that $T_{\mathcal{F}} \gg T_{\text{GP}}$. We will examine the effect of legacy data on GPareareal runtimes in the next section and more so in Chapter 6. These results illustrate that using GPareareal (with or without legacy data) we can solve and evaluate the dynamics of the FHN model in significantly fewer iterations than Parareal.

5.3.2 Rössler system

Next we solve the Rössler system,

$$\frac{\mathrm{d}u_1}{\mathrm{d}t} = -u_2 - u_3, \quad \frac{\mathrm{d}u_2}{\mathrm{d}t} = u_1 + \hat{a}u_2, \quad \frac{\mathrm{d}u_3}{\mathrm{d}t} = \hat{b} + u_3(u_1 - \hat{c}), \tag{5.22}$$

with parameters $(\hat{a}, \hat{b}, \hat{c}) = (0.2, 0.2, 5.7)$ that cause the system to exhibit chaotic behaviour (Rössler, 1976). We wish to integrate (5.22) over $t \in [0, 340]$ with initial values $\boldsymbol{u}^0 = (0, -6.78, 0.02)^\intercal$ and solvers $\mathcal{G}_{\Delta T} = \text{RK1}$ and $\mathcal{F}_{\Delta T} = \text{RK4}$. The interval is divided into $N = 40$ time slices, $N_{\mathcal{G}} = 9 \times 10^4$ coarse steps, and $N_{\mathcal{F}} = 4.5 \times 10^8$ fine steps. The tolerance is set to $\varepsilon = 10^{-6}$.

In this experiment, rather than obtaining legacy data by solving (5.22) using alternative initial values (as we did in Section 5.3.1), we instead generate the data by integrating over a shorter time interval. This is particularly useful if we are unsure how long to integrate our system for, i.e. to reach some long-time equilibrium state or reveal certain dynamics of the system, as is the case in many real-world dynamical systems. For example, systems that feature random noise may exhibit metastability, in which trajectories spend (a long) time in certain states (regions of phase space) before transitioning to another state (Grafke et al., 2017; Legoll et al., 2022). Such rare metastability may not be revealed/observed until the system has been evolved

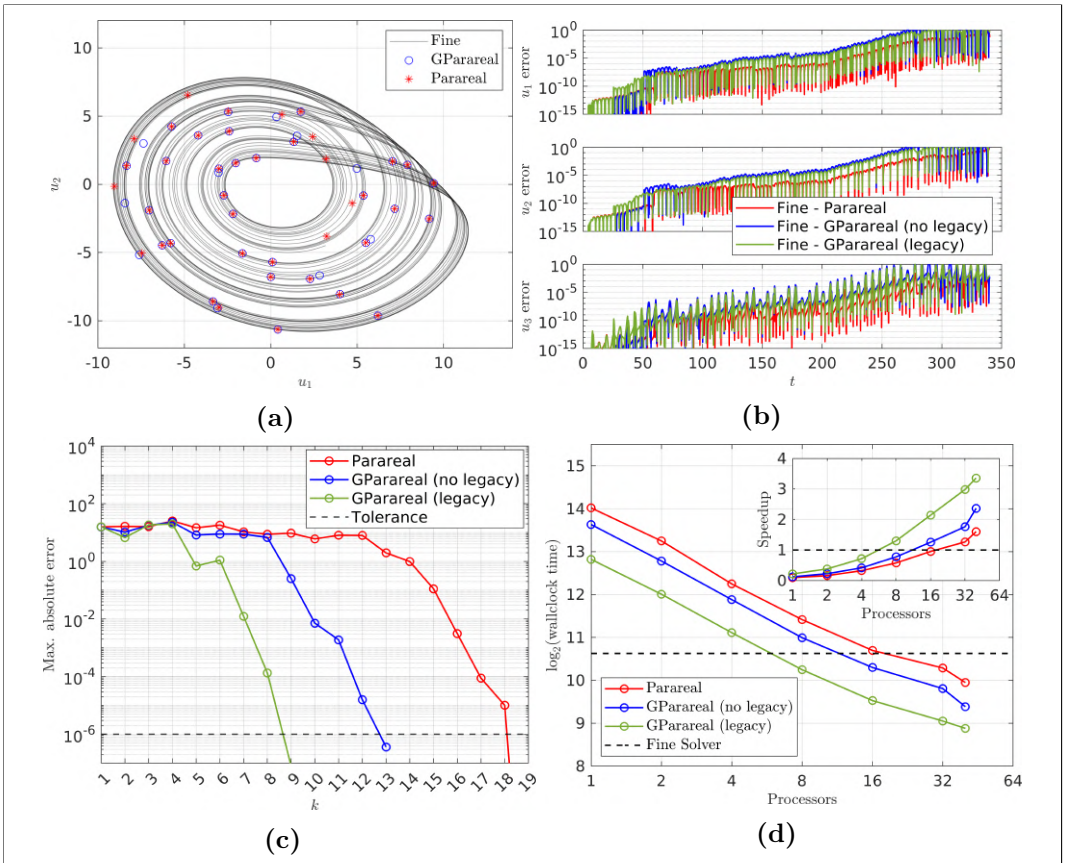


Figure 5.8: Numerical results obtained solving the Rössler system (5.22) over $t \in [0, 340]$. (a) Solutions from the fine solver, GParareal (with legacy data), and Parareal plotted in the (u_1, u_2)-plane—both GParareal and Parareal plotted only at times $\boldsymbol{t}$ for clarity. (b) The corresponding absolute errors between solutions from GParareal and Parareal vs. the fine solution. (c) Maximum absolute errors (2.10) of each algorithm at successive iterations k until tolerance $\varepsilon = 10^{-6}$ is met. (d) Median wallclock times (taken over 5 runs) of each simulation against the number of processors (up to 40). The inset plot displays the corresponding parallel speedup.

over a sufficiently large time interval. We propose integrating over a 'short' time interval, assessing the relevant characteristics of the solution obtained, and then integrating over a longer time interval (using the legacy data) if required. Note that to do this, all parameters in both simulations must remain the same, with the exception of the time step widths—to ensure the legacy data is usable in the GP emulator in the longer simulation. Suppose we solve (5.22) over $t \in [0, 170]$, then we need to reduce N, $N_{\mathcal{G}}$, and $N_{\mathcal{F}}$ by a factor of two, i.e. use $N^{(2)} = N/2$, $N_{\mathcal{G}}^{(2)} = N_{\mathcal{G}}/2$, and $N_{\mathcal{F}}^{(2)} = N_{\mathcal{F}}/2$ in the shorter simulation.

The legacy simulation, integrating over $[0, 170]$, takes nine iterations to converge using GParareal (ten for Parareal), giving us approximately $kN^{(2)} = 9 \times 20 = 180$ legacy evaluations of $\mathcal{F}_{\Delta T} - \mathcal{G}_{\Delta T}$ (results not shown). Integrating (5.22) over the full interval $[0, 340]$, GParareal converges in four fewer iterations with the legacy data

than without—refer to Figure 5.8(c). In Figure 5.8(d) we can see that using the legacy data achieves a higher numerical speedup (3.4×) compared to without (2.4×) and compared to Parareal (1.6×). In Figure 5.8(a) we see the trajectories from each simulation converging toward the Rössler attractor and Figure 5.8(b) illustrates GPararealretaining a similar numerical accuracy to Parareal with and without the legacy data. Note the steadily increasing errors for both algorithms is due to the chaotic nature of the Rössler system.

5.3.3 Nonautonomous system

Next, we consider a nonautonomous system of ODEs to demonstrate how GPararealhandles explicit time dependence. We solve

$$\frac{\mathrm{d}u_1}{\mathrm{d}t} = -u_2 + u_1\big(\frac{t}{500} - u_1^2 - u_2^2\big), \quad \frac{\mathrm{d}u_2}{\mathrm{d}t} = u_1 + u_2\big(\frac{t}{500} - u_1^2 - u_2^2\big), \tag{5.23}$$

over $t \in [-20, 500]$—adapted from Trefethen et al. (2017). As described in Section 5.2.5, we transform this two-dimensional nonautonomous system into a three-dimensional autonomous system by introducing an additional variable $u_3(t) = t$, where $\mathrm{d}u_3/\mathrm{d}t = 1$. Given that we know $u_3(t)$ explicitly, the third dimension of $\mathcal{F}_{\Delta T} - \mathcal{G}_{\Delta T}$ need not be modelled with a GP. However, given the GPs are run in parallel anyway, this does not add to the cost of running GPararealanyway.

We select solvers $\mathcal{G}_{\Delta T}$ = RK1 and $\mathcal{F}_{\Delta T}$ = RK8 with $N_{\mathcal{G}} = 2048$ and $N_{\mathcal{F}} = 5.12 \times 10^5$ steps, respectively. We use $N = 32$ time slices, initial condition $\boldsymbol{u}^0 = (0.1, 0.1, -20)^\intercal$, and a stopping tolerance of $\varepsilon = 10^{-6}$. In Figure 5.9, we plot the

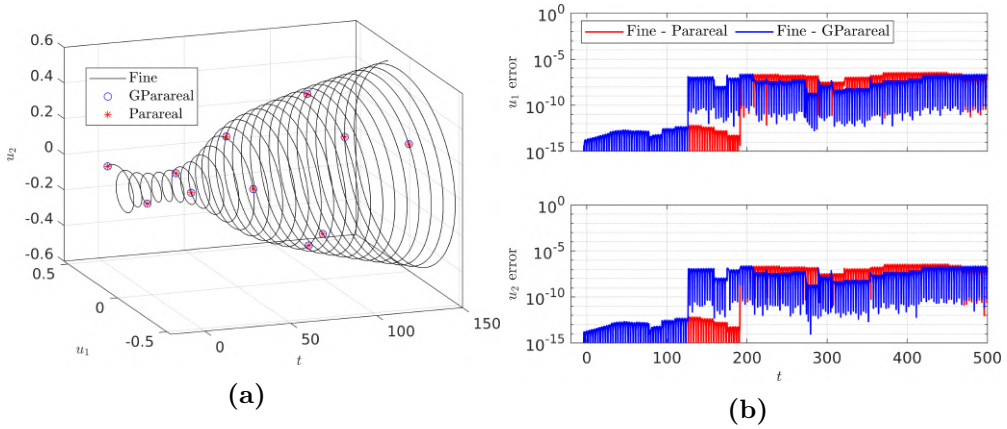


Figure 5.9: Numerical results obtained solving the nonautonomous system (5.23). (a) Time-dependent solutions using the fine solver, GPararealand Parareal—both GPararealand Parareal plotted only at times $\boldsymbol{t}$ on $[-20, 150]$ for clarity. (b) The corresponding absolute errors between solutions from GPararealand Parareal vs. the fine solution, having converged after 10 and 20 iterations, respectively.

Table 5.1: Numerical wallclock time, speedup, and efficiency results obtained solving the nonautonomous system (5.23) for $N \in \{32, 64, 128, 256, 512\}$ with (a) Parareal and (b) GParareal. Theoretical results calculated using (2.11)–(2.13) and (5.17)–(5.19) are shown in brackets. All timings are measured in seconds.

N	k_{para}	$T_{\mathcal{G}}$	$T_{\mathcal{F}}$	T_{GP}	T_{serial}	T_{para}	S_{para}	E_{para}
32	20	1.60E−4	4.23E3	—	1.35E5	8.92E4 (8.47E4)	1.52 (1.60)	0.05 (0.05)
64	31	9.80E−5	2.10E3	—	1.35E5	6.75E4 (6.52E4)	2.00 (2.06)	0.03 (0.03)
128	55	9.10E−5	1.06E3	—	1.35E5	6.47E4 (5.81E4)	2.09 (2.33)	0.02 (0.02)
256	99	6.90E−5	5.23E2	—	1.34E5	5.64E4 (5.17E4)	2.37 (2.59)	0.01 (0.01)
512	151	6.30E−5	2.62E2	—	1.34E5	4.42E4 (3.95E4)	3.03 (3.39)	0.01 (0.01)

(a) Parareal results

N	k_{GPara}	$T_{\mathcal{G}}$	$T_{\mathcal{F}}$	T_{GP}	T_{serial}	T_{GPara}	S_{GPara}	E_{GPara}
32	10	1.60E−4	4.23E3	5.81	1.35E5	4.33E4 (4.23E4)	3.13 (3.20)	0.10 (0.10)
64	14	9.80E−5	2.10E3	24.74	1.35E5	3.20E4 (2.95E4)	4.21 (4.57)	0.07 (0.07)
128	16	9.10E−5	1.06E3	3.01E2	1.35E5	1.90E4 (1.72E4)	7.13 (7.86)	0.06 (0.06)
256	18	6.90E−5	5.23E2	1.24E3	1.34E5	1.17E4 (1.06E4)	11.42 (12.57)	0.04 (0.05)
512	15	6.30E−5	2.62E2	1.62E4	1.34E5	2.10E4 (2.02E4)	6.34 (6.65)	0.01 (0.01)

(b) GParareal results

solutions and corresponding errors generated by each of the solvers over time. Again, the results illustrate good convergence to the fine solver solution, with GParareal taking 10 iterations to locate the solution and Parareal taking 20. We suspect that the superior performance of GParareal is partially due to the almost periodic nature of the solutions in Figure 5.9(a), enabling the emulator to reproduce the dynamics of $\mathcal{F}_{\Delta T} - \mathcal{G}_{\Delta T}$ quite well.

Next, we run a strong scaling experiment to measure the effect of increasing the number of time slices N (and hence processors) on convergence, wallclock time, and speedup—see Table 5.1. To do this, we increase $N_{\mathcal{F}}$ to 5.12×10^{10}, so that $\mathcal{F}_{\Delta T}$ is sufficiently expensive to observe speedup. We observe a good match between all numerical and theoretical results for both Parareal and GParareal—this is depicted in Figure 5.10. Firstly, notice that k_{para} (the number of iterations for Parareal to converge) increases with N whilst k_{GPara} remains largely unaffected, leading to speedups for GParareal being roughly $2\times$ to $4\times$ that of Parareal. For both algorithms, the cost of $T_{\mathcal{G}}$ and $T_{\mathcal{F}}$ decreases as N increases (due to fewer time steps per time slice), whilst T_{GP} increases in GParareal (due to increasing numbers of data points used to train the GP emulators in each simulation). Up to $N = 128$, $T_{\text{GP}} < T_{\mathcal{F}}$

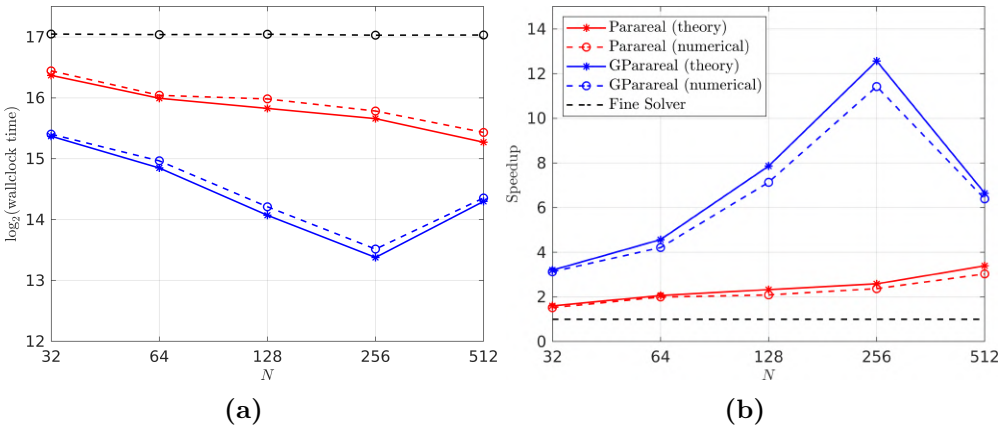


Figure 5.10: Strong scaling results obtained solving the nonautonomous system (5.23) for $N \in \{32, 64, 128, 256, 512\}$. (a) Wallclock times using the fine solver (dashed black), GParareal (dashed blue), and Parareal (dashed red). Corresponding theoretical results are given by the solid lines, calculated using (2.11) and (5.17), respectively. (b) The corresponding speedup results using the same lines and colours—theoretical results calculated using (2.12) and (5.18), respectively.

and so we observe increasing parallel speedup for GParareal. At $N = 256$, however, there is a turning point where the cost of training the GP is 2.4 times larger than $T_{\mathcal{F}}$, meaning that speedup is severely restricted. This is exacerbated further when using $N = 512$ processors, where T_{GP} is now almost 62 times the size of $T_{\mathcal{F}}$. Even though using the GP emulator massively helps reduce the number of iterations k for every value of N (compared to Parareal), these results clearly highlight the severe impact that the serial cost of the GP optimisation/conditioning has on realisable parallel speedup from GParareal. In turn, this hinders the parallel efficiency of the algorithm.

5.3.4 Double pendulum system

Consider now the double pendulum system: a simple pendulum of mass m, rod length ℓ, connected to another simple pendulum of equal mass m, rod length ℓ, acting

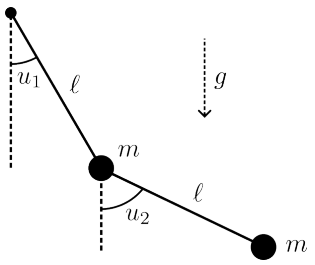


Figure 5.11: A schematic of the double pendulum system.

under gravity g (see Figure 5.11). Four ODEs govern the dynamics of this system:

$$\begin{aligned}
\frac{\mathrm{d}u_1}{\mathrm{d}t} &= u_3, \\
\frac{\mathrm{d}u_2}{\mathrm{d}t} &= u_4, \\
\frac{\mathrm{d}u_3}{\mathrm{d}t} &= \frac{-u_3^2 f_1(u_1,u_2) - u_4^2 \sin(u_1-u_2) - 2\sin(u_1) + \cos(u_1-u_2)\sin(u_2)}{f_2(u_1,u_2)}, \\
\frac{\mathrm{d}u_4}{\mathrm{d}t} &= \frac{2u_3^2 \sin(u_1-u_2) + u_4^2 f_1(u_1,u_2) + 2\cos(u_1-u_2)\sin(u_1) - 2\sin(u_2)}{f_2(u_1,u_2)},
\end{aligned} \tag{5.24}$$

where $f_1(u_1,u_2) = \sin(u_1-u_2)\cos(u_1-u_2)$ and $f_2(u_1,u_2) = 2-\cos^2(u_1-u_2)$ (Danby, 1997). Note that m, ℓ, and g have been scaled out of (5.24) by letting $\ell = g$. The variables u_1 and u_2 measure the angles between each pendulum and the vertical axis, while u_3 and u_4 measure the corresponding angular velocities.

For this experiment, we select solvers $\mathcal{G}_{\Delta T} = \text{RK1}$ and $\mathcal{F}_{\Delta T} = \text{RK8}$ with $N_{\mathcal{G}} = 3072$ and $N_{\mathcal{F}} = 2.1504 \times 10^5$ steps, respectively. We integrate over $t \in [0, 80]$, using $N = 32$ time slices with a stopping tolerance $\varepsilon = 10^{-6}$. In Figure 5.12, we plot solutions for u_1 and u_2 over time using initial conditions $\boldsymbol{u}^0 = (2, 0.5, 0, 0)^\intercal$, i.e. the pendulums are positioned at some (positive) initial angles and released from rest. Observe how both pendulums move chaotically, with the inner pendulum oscillating within $[-\pi, \pi]$ and the outer pendulum oscillating between odd multiples of π, "turning over" a number of times[6]. We attain good solution accuracy from GParareal with respect to the fine solution with errors slowly increasing over time

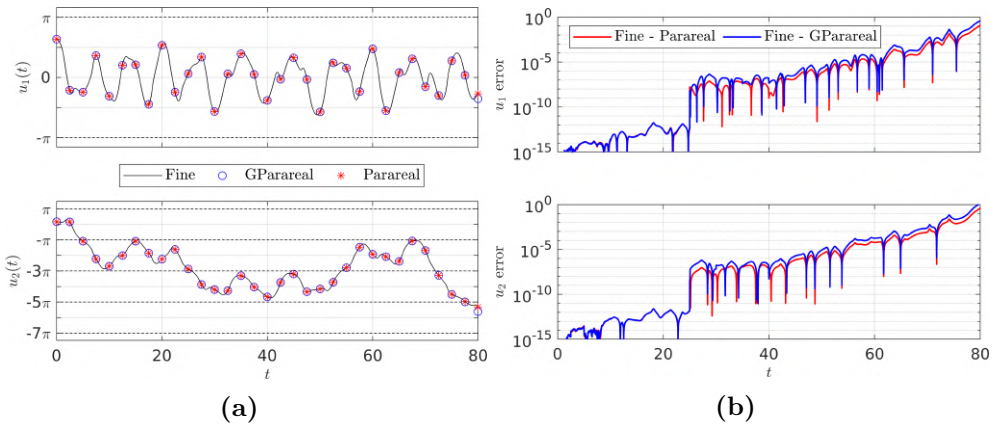
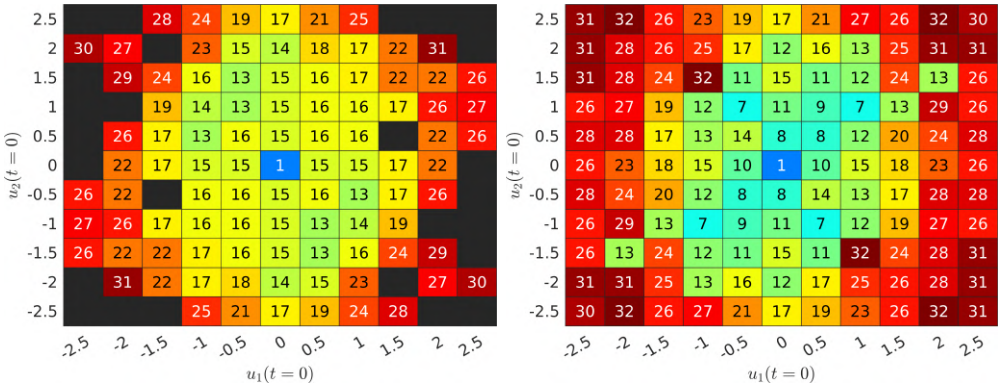


Figure 5.12: Numerical results obtained solving the double pendulum system (5.24). (a) Time-dependent solutions for u_1 and u_2 using the fine solver, GParareal, and Parareal—both GParareal and Parareal plotted only at times $\boldsymbol{t}$ for clarity. Dashed lines indicate "turning over" angles, at which either pendulum passes through an odd multiple of π. (b) The corresponding absolute errors between solutions from GParareal and Parareal vs. the fine solution, having converged after 23 and 22 iterations, respectively.

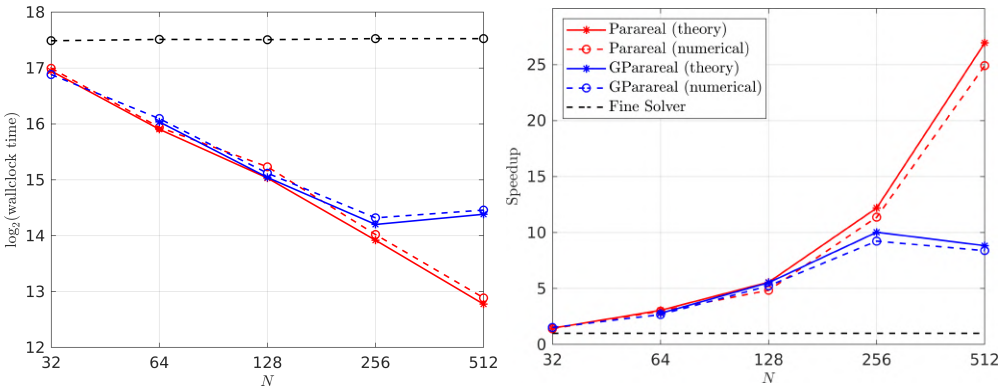
[6]See code repository for an animation.



(a) Parareal **(b)** GParareal

Figure 5.13: Heat maps displaying the number of iterations taken until convergence k of (a) Parareal and (b) GParareal when solving the double pendulum system (5.24) for different initial angles $(u_1(0), u_2(0))^\intercal \in [-2.5, 2.5]^2$ and initial angular velocities $(u_3(0), u_4(0))^\intercal = (0, 0)^\intercal$, i.e. the pendulums are released from rest. Black boxes indicate where Parareal returned a `NaN` value during simulation.

due to the chaotic nature of the system, much like what was seen in the Rössler experiments in Section 5.3.2. We plot k for various initial angles in Figure 5.13 to highlight the system's sensitivity to initial conditions. For small initial angles, GParareal converges sooner than Parareal, but for much larger angles both algorithms use almost all of the 32 iterations to locate a solution (and in some cases, Parareal does not return a solution).



(a) **(b)**

Figure 5.14: Strong scaling results obtained solving the double pendulum system (5.24) for $N \in \{32, 64, 128, 256, 512\}$. (a) Wallclock times using the fine solver (dashed black), GParareal (dashed blue), and Parareal (dashed red). Corresponding theoretical results are given by the solid lines, calculated using (2.11) and (5.17), respectively. (b) The corresponding speedup results using the same lines and colours—theoretical results were calculated using (2.12) and (5.18), respectively.

In Table 5.2 and Figure 5.14, we again examine the strong scaling of both algorithms. To do this, we increase the number of fine time steps to $N_{\mathcal{F}} = 2.1504 \times 10^{10}$. We purposefully choose an initial condition ($\boldsymbol{u}^0$ above) for which both algorithms converge in approximately the same number of iterations, so that we can directly observe how the increasing GP cost affects the performance of GPareareal for large N. Under these circumstances, we can think of the wallclock time for GPareareal as (approximately) the wallclock time of Parareal plus the wallclock time of the GP conditioning/optimisation. For $N \leqslant 128$, we observe that $T_{\text{GP}} < T_{\mathcal{F}}$ and so the speedup of GPareareal and Parareal are approximately the same. In these cases, using GPareareal is no more costly than using Parareal, with the additional benefit of being able to re-use the acquisition data for a future simulation, if needed. For $N \geqslant 256$, we begin to observe $T_{\text{GP}} > T_{\mathcal{F}}$ (or larger), so the numerical speedup of GPareareal begins to plateau. These results (and those in the prior section) make it clear that including the learning costs within the PinT framework is of paramount importance as they can have a huge impact on realisable parallel speedup. We will examine the effect of T_{GP} on final GPareareal runtimes further in Chapter 6.

Table 5.2: Numerical wallclock time, speedup, and efficiency results obtained solving the double pendulum system (5.24) for $N \in \{32, 64, 128, 256, 512\}$ with (a) Parareal and (b) GPareareal. Theoretical results calculated using (2.11)–(2.13) and (5.17)–(5.19) are shown in brackets. All timings are measured in seconds.

N	k_{para}	$T_{\mathcal{G}}$	$T_{\mathcal{F}}$	T_{GP}	T_{serial}	T_{para}	S_{para}	E_{para}
32	22	2.53E−4	5.75E3	—	1.84E5	1.31E5 (1.26E5)	1.41 (1.45)	0.04 (0.05)
64	21	1.46E−4	2.93E3	—	1.87E5	6.29E4 (6.14E4)	2.97 (3.05)	0.05 (0.05)
128	23	1.27E−4	1.46E3	—	1.86E5	3.85E4 (3.35E4)	4.84 (5.57)	0.04 (0.04)
256	21	9.10E−5	7.35E2	—	1.89E5	1.66E4 (1.55E4)	11.36 (12.19)	0.04 (0.05)
512	19	7.00E−5	3.69E2	—	1.89E5	7.58E3 (7.01E3)	24.90 (26.94)	0.05 (0.05)

(a) Parareal results

N	k_{GPara}	$T_{\mathcal{G}}$	$T_{\mathcal{F}}$	T_{GP}	T_{serial}	T_{GPara}	S_{GPara}	E_{GPara}
32	21	2.53E−4	5.75E3	21.44	1.84E5	1.21E5 (1.21E5)	1.52 (1.52)	0.05 (0.05)
64	23	1.46E−4	2.93E3	35.22	1.87E5	7.00E4 (6.72E4)	2.67 (2.78)	0.04 (0.04)
128	23	1.27E−4	1.46E3	2.63E2	1.86E5	3.56E4 (3.36E4)	5.24 (5.52)	0.04 (0.04)
256	23	9.10E−5	7.35E2	1.87E3	1.89E5	2.04E4 (1.75E4)	9.24 (10.03)	0.04 (0.04)
512	22	7.00E−5	3.69E2	1.33E4	1.89E5	2.25E4 (1.04E4)	8.38 (8.83)	0.02 (0.02)

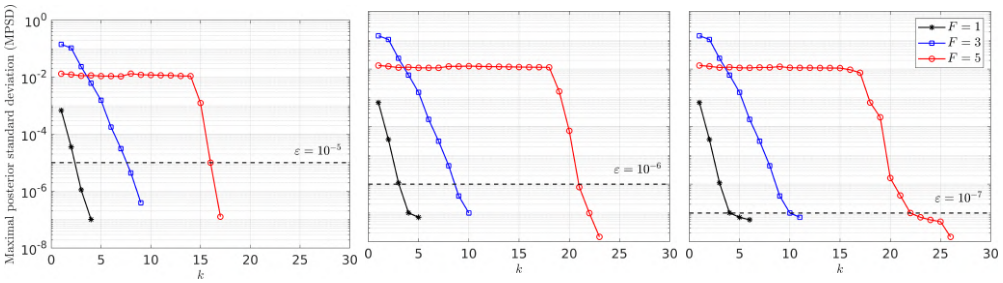
(b) GPareareal results

5.4 Improving convergence: GPareareal + fallback

In Section 5.2.4, we derived an error bound (Theorem 5.3) for solutions obtained from GPareareal at a given iteration that was proportional to the fill distance of the dataset $h_{\boldsymbol{x}^k}$. We know that the fill distance may become large when there is insufficient acquisition data available, an effect that is exacerbated when the dimension d is large. Therefore when solving d-dimensional systems of ODEs, GPareareal may require increasingly large amounts of data to model $\mathcal{F}_{\Delta T} - \mathcal{G}_{\Delta T}$ accurately and may therefore take a large number of iterations to converge. The purpose of this section is to investigate the behaviour of the posterior variance arising from the GP predictions of $\mathcal{F}_{\Delta T} - \mathcal{G}_{\Delta T}$, which up until now has been ignored, and gives us an indication of how accurate the emulators are as the amount of acquisition data increases, i.e. as k increases. Intuitively, one would expect the posterior variance of the GP predictions to be larger during early iterations of GPareareal due to the lack of acquisition data available (in the absence of any legacy data).

In the univariate setting ($d = 1$), GPareareal queries the scalar-output GP emulator for $\mathcal{F}_{\Delta T} - \mathcal{G}_{\Delta T}$ at a number of input locations each iteration, obtaining a Gaussian posterior distribution with some mean and variance (recall (5.12)). At each iteration, we will store the maximal posterior variance—in fact we will store its square root, the maximal posterior standard deviation (MPSD). Given we are interested in the multivariate setting ($d > 1$), the setup will be the same, except that we have d independent emulators which return d MPSD values, again of which we take the largest.

In Figure 5.15, we plot the MPSD against k (until the stopping tolerance is met) for the Lorenz96 system (E.1) using $F \in \{1, 3, 5\}$ and $d = 50$ (refer to Appendix E for details of Lorenz96). The parameters used for the GPareareal simulations are given in Table 5.3. Across the panels we decrease the stopping tolerance ε (recall (2.10))



(a) $\varepsilon = 10^{-5}$ **(b)** $\varepsilon = 10^{-6}$ **(c)** $\varepsilon = 10^{-7}$

Figure 5.15: MPSD at each iteration k in simulations of GPareareal. Each panel shows results when solving the Lorenz96 (E.1) with forcing $F \in \{1, 3, 5\}$ (other parameters given in Table 5.3) and different stopping tolerances: (a) $\varepsilon = 10^{-5}$, (b) $\varepsilon = 10^{-6}$, and (c) $\varepsilon = 10^{-7}$ (dashed black lines).

to highlight the behaviour of the MPSD when GParareal stops. In the $F = 1$ and $F = 3$ cases, we can see that the MPSD consistently decreases as more acquisition data is accumulated each iteration, with GParareal stopping one or two iterations after it drops below ε. As expected, decreasing the stopping tolerance makes it more difficult for GParareal to stop sooner—reflected by the increased number of iterations required to stop in each panel. In the $F = 5$ cases, we see that the emulators are struggling to accurately infer $\mathcal{F}_{\Delta T} - \mathcal{G}_{\Delta T}$ until around $k = 15$, with the MPSD remaining approximately constant. This suggests that GParareal spends a lot of time "jumping around" state space looking for the correct solution states. In the next section, we propose a remedy to avoid slow convergence by making use of the GP posterior variance. In addition, all results showed that GParareal will not stop iterating until the emulators are sufficiently accurate, i.e. until the MPSD is smaller than ε, another property that we can exploit to accelerate convergence[7]. Overall, the $F = 5$ cases highlight the need for the emulators to learn $\mathcal{F}_{\Delta T} - \mathcal{G}_{\Delta T}$ sufficiently quickly during early iterations to avoid slow convergence.

5.4.1 The modification

To improve the slow convergence that occurs when the GPs are insufficiently trained, we investigate the use of a Parareal "fallback" correction. The intuition is that when the GP posterior variance at the input location of interest is deemed "too large" we may expect the posterior mean of $\mathcal{F}_{\Delta T} - \mathcal{G}_{\Delta T}$ to be a poor estimate of the exact correction we need. In this scenario, the idea is to discard the GP posterior mean and instead "fallback" and use the classic Parareal correction—similar to the idea in SParareal where we chose the first "random" sample to be the classic Parareal PC solution. We do not necessarily know whether the fallback will give us a "better" correction than the GP posterior mean, however, the hope is that it will be useful during earlier iterations of GParareal when we expect the GP to have poor accuracy

Table 5.3: Parameters used to solve the Lorenz96 system (E.1) for different levels of forcing F—each row corresponds to a different experiment. These parameters are used in the simulations in Figures 5.15, 5.16, and 5.17.

F	d	$[t_0, T]$	N	$N_{\mathcal{G}}$	$N_{\mathcal{F}}$	$\mathcal{G}_{\Delta T}$	$\mathcal{F}_{\Delta T}$
1	50	$[0, 100]$	50	5E3	1E5	RK1	RK8
3	50	$[0, 100]$	50	5E3	1E5	RK1	RK8
5	50	$[0, 100]$	50	1E4	1E5	RK2	RK8

[7]Note that posterior variances will struggle to go below the pre-set "jitter" in the GP emulators. The jitter is a constant added to the diagonal of the covariance matrix $K(\boldsymbol{x}, \boldsymbol{x})$ to ensure numerical stability during inversion (typically via a Cholesky decomposition). If set too large, the jitter will lead to large posterior variances and non-convergence (the jitter is set to 10^{-14} in these experiments).

due to a lack of acquisition data.

At some iteration k, we query the (trained) GP emulator at input location U_n^k, obtaining the Gaussian posterior

$$(\mathcal{F}_{\Delta T} - \mathcal{G}_{\Delta T})(U_n^k) \mid \{\boldsymbol{x}^k, \boldsymbol{y}^k\} \sim \mathcal{N}\big(\hat{\mu}(U_n^k), \hat{K}(U_n^k, U_n^k)\big).$$

We then use posterior mean $\hat{\mu}(U_n^k)$ (5.5) in the GParareal update (5.16c), ignoring the posterior variance $\hat{K}(U_n^k, U_n^k)$ (5.6), its estimate of uncertainty. The proposal is to check, each time we query the GP, whether the corresponding variance is large, i.e. check whether

$$\hat{K}(U_n^k, U_n^k) < \omega^2, \tag{5.25}$$

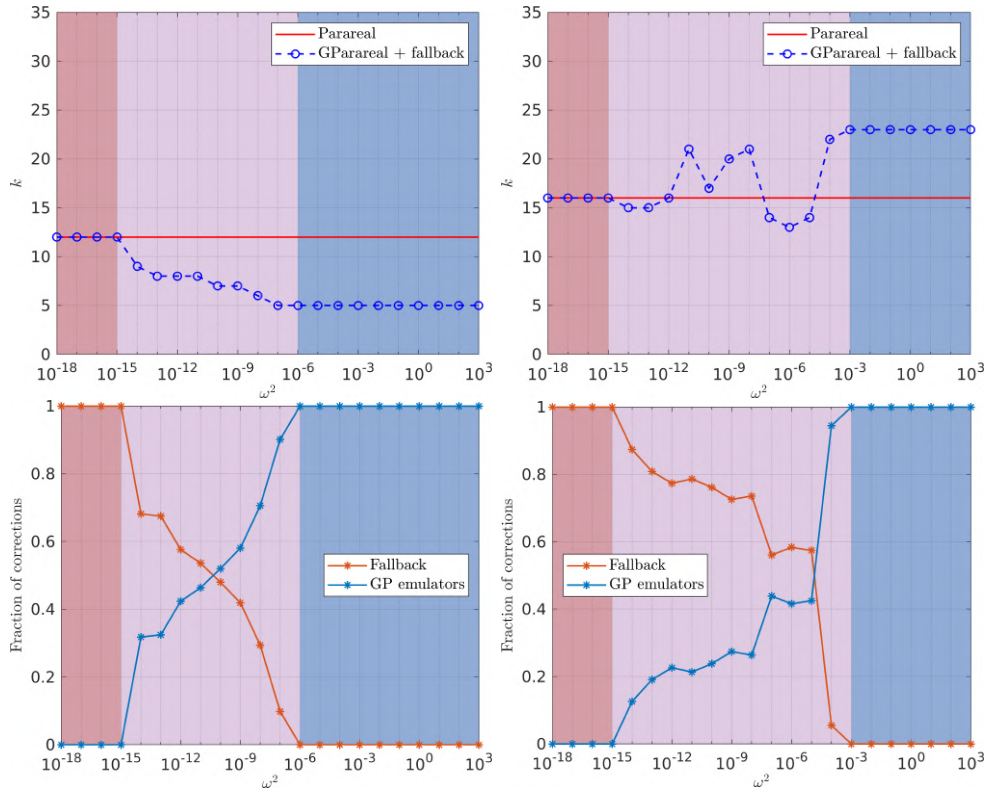
for some pre-defined "switching" tolerance $\omega^2 > 0$. If the tolerance in (5.25) is met, then we accept the GP posterior mean and carry out the update as usual in (5.16c). If the tolerance is exceeded, however, we reject the posterior mean and instead use the Parareal PC update (2.9c). This fallback correction is readily available as it is an element of the dataset that has been used to train the GP, i.e. $(\mathcal{F}_{\Delta T} - \mathcal{G}_{\Delta T})(U_n^k) \in \boldsymbol{y}^k$. Note that in the multivariate case ($d > 1$), the criterion in (5.25) is checked for each output dimension of $\mathcal{F}_{\Delta T} - \mathcal{G}_{\Delta T}$ (i.e for each of the d emulators) and so some output dimensions may use the GP emulators and some may use the Parareal fallback.

This modified version of GParareal, henceforth referred to as "GParareal + fallback", enables the automatic switching between GP and Parareal corrections depending on how well trained the GP emulator is at a given iteration. The value of ω^2, however, must be chosen *a priori* to simulation. If chosen too small, the GP posterior means will always be rejected, meaning that we are essentially running classic Parareal (at higher cost because of the GP training/optimisation). If chosen too large, the GP corrections will always be accepted, meaning we are running standard GParareal. Between these extremes should lie values of ω^2 for which GParareal + fallback automatically switches between the two different corrections, ideally converging in fewer iterations than standard GParareal. In this case, we would expect more fallback corrections to be made during early iterations when limited acquisition data is available to the GP emulator and more GP corrections during later iterations once the emulator is sufficiently well trained. We will investigate this numerically in the next section.

5.4.2 Numerical experiments

In this section, we test GParareal + fallback for various values of ω^2 on the Lorenz96 system (E.1) with $F \in \{1, 5\}$, see Figure 5.16. In the $F = 1$ case, we observe (left panels) that for small ω^2 only fallback corrections are made (bottom left panel) and

so both Parareal and GParareal + fallback converge in $k = 12$ iterations as expected (top left panel). As ω^2 is increased, GParareal + fallback takes fewer iterations as an increasing proportion of corrections are made by the GP emulators. Once ω^2 hits 10^{-6}, all corrections are being made by the GP emulators and the iteration count flattens off at $k = 5$. In this particular example, GParareal + fallback converges in fewer iterations than Parareal (whenever the GP emulator is used), further demonstrating the power of using the GP emulators within the PinT framework. The $F = 5$ case (right panels) highlights the benefit of the Parareal fallback more clearly. For $\omega^2 \geqslant 10^{-3}$, we see that no fallback corrections are made (standard GParareal) and convergence occurs in $k = 23$ iterations, seven more than Parareal ($k = 16$). Between $10^{-15} < \omega^2 < 10^{-3}$, however, we observe a range of behaviour, converging optimally in $k = 13$ iterations when $\omega^2 = 10^{-6}$. This provides us with evidence that by using GParareal in combination with the fallback corrections, convergence can occur in fewer iterations than both standard GParareal *and* Parareal. There is clearly a



(a) $F = 1$ **(b)** $F = 5$

Figure 5.16: Iterations until convergence k of Parareal and GParareal + fallback (top row) and the corresponding fraction of corrections made using the fallback or the GP emulators (bottom row) against varying ω^2 when solving the Lorenz96 system (E.1) with (a) $F = 1$ and (b) $F = 5$. Shaded areas correspond to which correction term has been called in GParareal + fallback (red = fallback only, blue = GP emulators only, and purple = a mixture).

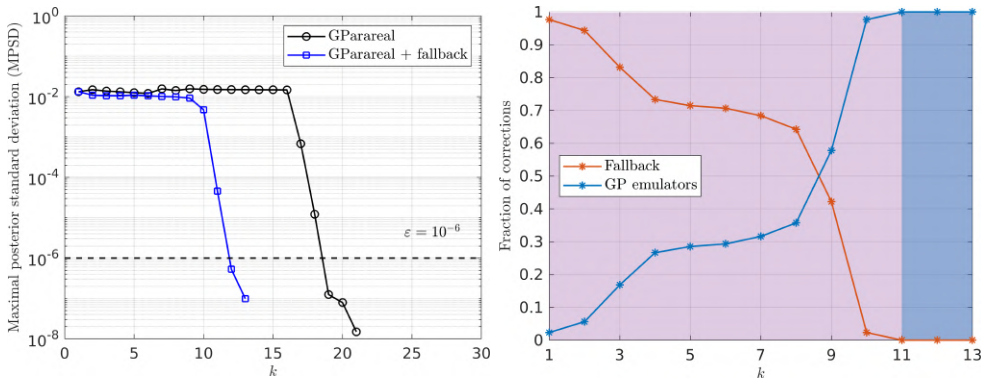


Figure 5.17: (a) MPSD at each iteration k in simulations of GPararealI and GPararealI + fallback ($\omega^2 = 10^{-6}$) when solving the Lorenz96 system (E.1) with the $F = 5$ parameters in Table 5.3. The dashed black line indicates the stopping tolerance $\varepsilon = 10^{-6}$. (b) Fraction of corrections made by the Parareal fallback and the GP emulators each iteration in the GPararealI + fallback simulation. Shaded areas correspond to which correction term has been called (red = fallback only, blue = GP emulators only, and purple = a mixture).

nonlinear relationship between k and ω^2 and whilst this has only been shown for this particular ODE system, the results certainly are encouraging.

Now let us focus on the $F = 5$ case and examine the behaviour of the MPSD when running both GPararealI and GPararealI + fallback (with $\omega^2 = 10^{-6}$) in Figure 5.17(a). We see that by using the fallback corrections, the uncertainty in the GP emulators decrease rapidly after 9 iterations compared to 18 without, reaching tolerance much sooner. We can see this process in effect more clearly in Figure 5.17(b), where during early iterations most of the corrections are carried out using the fallback with the GP emulators taking over after iteration $k = 8$ (when sufficient acquisition data is available). The fallback correction seems to have helped reduce the number of iterations spent exploring the state space as we mentioned before.

As a final remark, recall that in each panel of Figure 5.15 and Figure 5.17(a) we observed that GPararealI does not stop iterating until the MPSD reaches the stopping tolerance. This behaviour suggests that one could use this as an additional stopping criterion in GPararealI (and subsequently GPararealI + fallback). If this were the case, the algorithm would stop at iteration k if (i) all N time slices meet the standard stopping criterion (2.10) or (ii) the MPSD from all of the emulators is below ε, i.e.

$$\max_n \sqrt{\hat{K}(U_n^k, U_n^k)} < \varepsilon. \tag{5.26}$$

The intuition is that the improvement in solution accuracy is minimal once the MPSD is very small. This should save an extra iteration or two, which we know can be extremely beneficial in terms of speedup, at the expense of a small decrease in solution

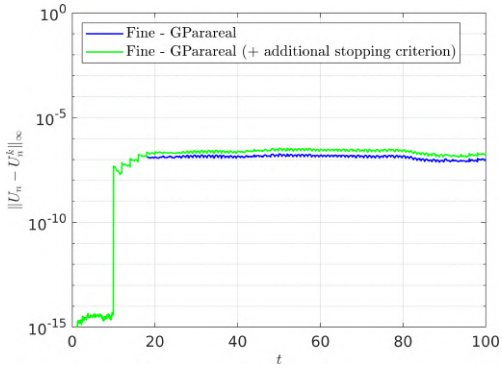


Figure 5.18: Effect of using the additional stopping criterion (5.26) on final GParareal solution accuracy. Shown are the maximal errors between solutions from GParareal vs. the fine solution when solving the Lorenz96 system (E.1) with parameters from the first row of Table 5.3.

accuracy[8]. For example, when solving the Lorenz96 system (with parameters from the first row of Table 5.3) we save a single iteration of GParareal using the additional criterion in (5.26) at little cost in terms of solution accuracy (see Figure 5.18).

This series of experiments has shown that we can make use of the (previously ignored) GP posterior variance (5.6) within GParareal. While it may seem counter-intuitive to use a Parareal fallback correction given that GParareal was developed to improve the corrections made by Parareal, we have seen that by using both types of correction, convergence can occur in fewer iterations than both algorithms (at no extra computational cost). In the situation that more (or perhaps all) of the corrections are being made by the Parareal fallback, one may ask: why not just use Parareal? The answer would be that by using GParareal + fallback, we combine the power of having both corrections available (as seen in Figure 5.16(b)) and we can reuse the acquisition data (plus legacy data if we have any) in a future simulation, whereas with just Parareal we cannot do this.

The question remains, however, over how to select ω^2 prior to simulation (without knowing beforehand whether GParareal or Parareal converges in fewer iterations). The results in Figure 5.16(b) suggest that to converge in the fewest iterations, one should choose $\omega^2 \in [10^{-7}, 10^{-5}]$. This is where the stopping tolerance lies ($\varepsilon = 10^{-6}$) but is likely to just be coincidental for this IVP. As a rule of thumb, one could run GParareal + fallback with a large value of ω^2 (e.g. run standard GParareal), assess if convergence is slow, and then if necessary run it again with $\omega^2 \approx \varepsilon$ to see whether convergence improves. This process is far from rigorous and so further testing on different IVPs is needed to reveal a more robust method for choosing ω^2.

[8]Retrospectively thinking, this type of stopping criterion could also be used within SParareal. One could stop SParareal when the largest standard deviation from the sampling rule is smaller than ε, as perturbations smaller than this are unlikely to improve the solution any further.

5.5 Discussion and further work

In this chapter, we presented GParareal, a learning-based time-parallel algorithm that iteratively locates a numerical solution to a system of ODEs (in parallel) by using a GP emulator to infer the correction term $\mathcal{F}_{\Delta T} - \mathcal{G}_{\Delta T}$. The numerical experiments reported in Section 5.3 demonstrate that GParareal performs favourably compared to Parareal, converging in fewer iterations and achieving increased parallel speedup for a number of low-dimensional nonlinear ODE systems. We also demonstrate how GParareal can make use of legacy data, i.e. prior $\mathcal{F}_{\Delta T}$ and $\mathcal{G}_{\Delta T}$ data obtained during a previous simulation of the same system (using different ICs or a shorter time interval), to pre-train the emulator and converge even faster.

In Section 5.3.1, using just acquisition data obtained during simulation, GParareal achieves an almost two-fold increase in speedup over Parareal when solving the FHN model. Simulating over a range of initial values, GParareal converged in fewer than half the iterations taken by Parareal and, in some cases, managed to converge when the coarse solver was too poor for Parareal. When using legacy data, GParareal could converge in even fewer iterations. Similar results were illustrated for the Rössler system in Section 5.3.2 but with legacy data obtained from a prior simulation over a shorter time interval—beneficial when one does not know how long to integrate a system for. In Sections 5.3.3 and 5.3.4, GParareal was tested on a larger number of processors (up to 512), verifying the theoretical computational complexity results given in Section 5.2.3. These strong scaling experiments showed that the cost of conditioning and optimising the GP emulator needs to be accounted for and also be much smaller than the cost of running the fine solver in order to maximise speedup. In all cases, the solutions generated by GParareal were of a numerical accuracy comparable to those found using Parareal.

GParareal in its current form may, however, suffer from the curse of dimensionality in two ways. First, an increasing number of data points each iteration, $\mathcal{O}(kN)$, increases the cost of conditioning and optimising the standard cubic complexity GP we implemented—observed numerically in Sections 5.3.3 and 5.3.4. To aleviate this problem, some of the non-standard (non-cubic complexity) GP emulation techniques mentioned in Section 5.2.3 could be implemented. Second, trying to emulate a nonlinear d-dimensional function $\mathcal{F}_{\Delta T} - \mathcal{G}_{\Delta T}$ is difficult if insufficiently many data points are available to the emulator. To tackle this issue, we proposed a modification to GParareal (in Section 5.4) that makes use of a Parareal fallback (instead of the GP posterior mean) when the GP posterior variance is deemed too large, i.e. greater than some switching tolerance $\omega^2 > 0$. Numerical experiments showed that during early iterations, the Parareal fallback was used more frequently (when the GP emulators had insufficient acquisition data) while the GP emulator corrections were made more frequently during later iterations (when enough acquisition data was available). The

modification was observed to be most useful in situations where (standard) GParareal took more iterations to converge than Parareal, however, choosing ω^2 remains an open problem.

Alternatively, one may tackle the dimensionality issue by generating more acquisition data with additional $\mathcal{F}_{\Delta T}$ and $\mathcal{G}_{\Delta T}$ runs (in parallel) using the idle processors—this should come at little to no extra computational cost (note that training costs would increase each iteration). Another option could be to use legacy data generated by evaluating $\mathcal{F}_{\Delta T} - \mathcal{G}_{\Delta T}$ at specific input locations, chosen by an appropriate space-filling design (e.g. Latin hypercube sampling) that satisfies certain fill distance requirements in the state space—something we will do in Chapter 6. However, as discussed in Section 5.2.4, the accuracy of the GP emulator (and therefore GParareal, see Theorem 5.3) is strongly controlled by the fill distance, which is generally difficult to restrict when d is large (even when large datasets are available). A final option could be to consider dimensionality reduction techniques (e.g. principal component analysis) and instead project the acquisition/legacy datasets onto a lower dimensional space from which the emulation may be easier.

In equation (5.14), we approximate a Gaussian distribution by taking its expected value, ignoring uncertainty in the GP posterior for $\mathcal{F}_{\Delta T} - \mathcal{G}_{\Delta T}$. In this setting, the GP emulator is used to interpolate the $\mathcal{F}_{\Delta T} - \mathcal{G}_{\Delta T}$ data, hence it is perfectly acceptable to swap it out for any other sufficiently accurate interpolation method, e.g. kernel ridge regression (Kanagawa et al., 2018). One alternative to approximating (5.14) by its expected value could be to draw a random sample instead. This would yield a sampling-based PinT scheme that would return a stochastic solution to the ODE, much like SParareal. It is unclear how this algorithm would perform vs. Parareal (or even SParareal), however, it could still make use of legacy data and perhaps generate a measure of uncertainty over the solution following successive independent simulations. Even though GParareal in its current form does not return a probabilistic solution to the IVP (2.1), we believe that it constitutes a positive step in this direction. In the next chapter, we will push GParareal to its limits by investigating how well it can perform when solving PDE problems.

Chapter 6

GParareal II: application to PDEs

Overview

Modelling complex real-world systems typically involves solving systems of PDEs which evolve in both space and time. PDEs are typically re-written as large systems of ODEs (2.1) using a variety of different spatial discretisation schemes (e.g. finite difference, element, and volume methods) that have varying degrees of accuracy and may have certain desirable properties for given systems (e.g. stability). In this chapter, we continue to investigate the performance of GParareal by solving PDE problems of increasing size d. Our main focus is to determine to what extent GParareal suffers from some of the issues (e.g. high GP emulator runtimes) discussed and observed in Chapter 5. For GParareal to be able to solve real-world problems we need to understand and quantify the scale of these limitations and propose options to mitigate some of the more negative effects.

Before diving into the numerical experiments, we briefly mention the solution of linear IVPs in Section 6.1 and discuss why we do not consider solving them with GParareal. In Section 6.2, we solve some nonlinear PDEs, applying standard GParareal (without the fallback correction) to the one-dimensional viscous Burgers' equation and the two-dimensional FitzHugh–Nagumo (2D FHN) system. For Burgers' equation we analyse the performance of GParareal when solving for different initial conditions and when using different types of legacy data, measuring the impact that GP training/querying costs have on realisable speedup. In light of these experiments, we propose storing Cholesky decompositions of covariance matrices following hyperparameter optimisation, resulting in a significant reduction in GP querying time. In the 2D FHN experiments, we increase the number of spatial discretisation points and observe that GParareal converges in fewer iterations than Parareal in almost all cases. However, the GP emulation costs take up a significant

proportion of the GParareal runtime, leading to severe speedup degradation. To mitigate these effects, we optimise GParareal further by pre-computing distance matrices in the kernel evaluations. We conclude by briefly discussing the impact of these improvements, ready for a full and thorough evaluation of our probabilistic PinT algorithms in Chapter 7.

6.1 Some remarks on linear PDEs

Having overlooked linear problems in Chapter 5, we should point out that when the vector field in (2.1) is linear, we do not need to use a GP to emulate the correction term in GParareal because it is simply a linear operator. By modifying GParareal, we could learn this linear operator directly (using only acquisition data from the first iteration) and solve linear IVPs in one iteration, however, this is not necessary.

As mentioned before, when spatially discretising a linear (autonomous) PDE, we obtain a system of ODEs (4.26), where Q is typically a large (depending on the number of spatial points) and possibly sparse real matrix. We know that this system has an analytical solution $\boldsymbol{u}(t) = \boldsymbol{u}(t_0)e^{Q(t-t_0)}$ which requires the evaluation of the matrix exponential $e^{Q(t-t_0)} = \sum_{i=0}^{\infty}(Q(t-t_0))^i/i!$. To find the solution at evenly-spaced time points $\boldsymbol{t} = (t_0, \ldots, t_N)$, one can simply compute the matrix exponential $e^{Q\Delta T}$ once and then apply it serially (via matrix multiplication) to the previous state, i.e. one can then rapidly evaluate $\boldsymbol{u}(t_{n+1}) = \boldsymbol{u}(t_n)e^{Q\Delta T}$. To calculate the matrix exponential one can use Krylov subspace methods, the scaling and squaring method (Higham, 2005) or Padé approximations (Arioli et al., 1996), among others. For particularly large systems, the multiplication of the matrix exponential and the previous state can also be parallelised. This approach of solving linear autonomous PDEs yields very high parallel efficiencies and is more accurate than using iterative PinT methods such as Parareal and GParareal.

For inhomogeneous nonautonomous linear IVPs, e.g. $\boldsymbol{f}(t, \boldsymbol{u}(t)) := Q\boldsymbol{u}(t) + \boldsymbol{h}(t)$ (where $\boldsymbol{h}(t)$ is a difficult-to-integrate source term), Gander and Güttel (2013) propose using the *ParaExp* algorithm. ParaExp exploits the fact that the homogeneous term $Q\boldsymbol{u}(t)$ can be integrated rapidly using the matrix exponential method suggested above, whilst the inhomogeneous part $\boldsymbol{h}(t)$ is more costly and therefore parallelisable. This non-iterative method can achieve parallel efficiencies in excess of 0.84 for both diffusive and non-diffusive problems. In addition, they discuss other methods for solving this problem as well as the homogeneous autonomous problem discussed before. Given that these methods are already highly efficient and accurate, our focus will remain on solving nonlinear IVPs with GParareal.

6.2 Numerical experiments: nonlinear PDEs

We now turn our attention back to GPareareal, assessing its performance when solving nonlinear PDEs.

6.2.1 One-dimensional viscous Burgers' equation

We begin our experiments by solving the viscous Burgers' equation

$$\begin{aligned} \text{PDE} \quad & v_t = \nu v_{xx} - v v_x && (x,t) \in (-L, L) \times (t_0, T] \\ \text{IC} \quad & v(x, t_0) = v_0(x) && x \in [-L, L] \\ \text{BCs} \quad & v(-L, t) = v(L, t) \\ & v_x(-L, t) = v_x(L, t) && t \in [t_0, T] \end{aligned} \tag{6.1}$$

for $v(x,t)$, where ν is the diffusion coefficient. We discretise the spatial domain using $d+1$ equally spaced spatial points $x_{j+1} = x_j + \Delta x$ where $\Delta x = 2L/d$ (for $j = 0, \ldots, d$). Writing $u_j(t) := v(x_j, t)$, the semi-discretised problem is then given by

$$\frac{\mathrm{d}\boldsymbol{u}}{dt} = \nu D_{xx}^{(2)} \boldsymbol{u} - \boldsymbol{u} \circ (D_x^{(2)} \boldsymbol{u}), \quad t \in (t_0, T],$$

where

$$D_{xx}^{(2)} = \frac{1}{(\Delta x)^2} \begin{bmatrix} -2 & 1 & & & 1 \\ 1 & -2 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & -2 & 1 \\ 1 & & & 1 & -2 \end{bmatrix}, \quad D_x^{(2)} = \frac{1}{2\Delta x} \begin{bmatrix} 0 & 1 & & & -1 \\ -1 & 0 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 0 & 1 \\ 1 & & & -1 & 0 \end{bmatrix}, \tag{6.2}$$

are second-order accurate finite difference matrices (Fornberg, 1988). Note that we have enforced the periodic boundary conditions via the first and last rows of these spatial operators. There are more accurate/stable spatial discretisation methods for dealing with the nonlinear term in (6.1), however, for our experiments finite differences will suffice.

Before proceeding with the experiments we remark on the implementation of different boundary conditions. When attempting to implement Dirichlet boundary conditions (e.g. $v(-L,t) = v(L,t) = 0$) in (6.1) we found that the GP emulators run into numerical stability issues when trying to Cholesky decompose covariance matrices. The instabilities arise because a number of the acquisition data points generated are "too close" to one another in state space—due to the fixed solution values on the boundaries. A similar problem occurs for Neumann boundary conditions (e.g. $v_x(-L,t) = v_x(L,t) = 0$) if the solution values on the boundaries do not change very much over time. With periodic conditions, solutions values on the boundaries

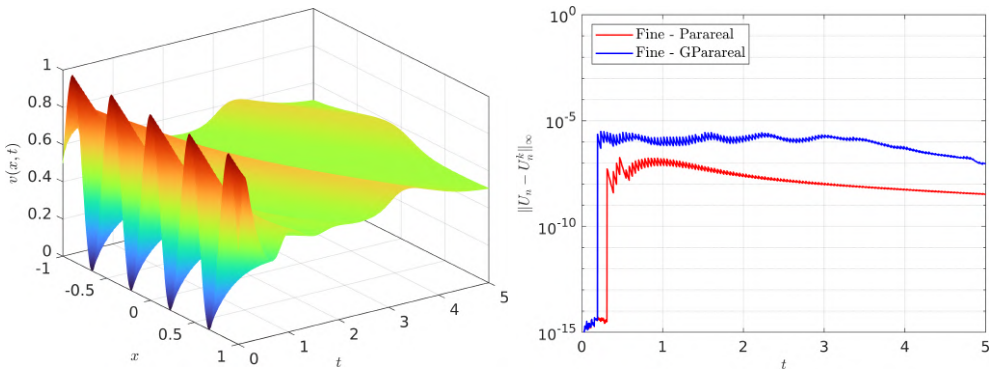


Figure 6.1: Numerical results obtained solving the viscous Burgers' equation (6.1) over $(x,t) \in [-1,1] \times [0,5]$. (a) Solution $v(x,t)$ obtained using the fine solver. (b) The maximal errors between solutions from Parareal and GParareal vs. the fine solution, having converged in 9 and 4 iterations, respectively.

are always changing (see Figure 6.1(a)) and so the instabilities do not arise. One way to avoid these numerical issues could be to remove input data points that are too close (using some pre-defined radius) to one another—although we did not test this here. On a similar thread, we attempted to discretise the spatial domain using pseudospectral methods (Trefethen, 2000), where spatial points are designed to cluster near boundaries (where interesting dynamics often occur). This meant, however, that data points were again too close to one another and so numerical instabilities arose. One thing to investigate in the future is whether alternative spatial discretisation methods (e.g. finite elements, finite volume) are compatible with GParareal and whether there are ways to avoid these numerical instabilities.

In the following experiments, we set $N = d = 128$, $L = 1$, $\nu = 10^{-2}$, integrate over $t \in [0,5]$, and select solvers $\mathcal{F}_{\Delta T} = \text{RK8}$ and $\mathcal{G}_{\Delta T} = \text{RK1}$. In Figure 6.1, we solve (6.1) using $N_{\mathcal{F}} = 256,000$ and $N_{\mathcal{G}} = 512$ time steps—the stopping tolerance for Parareal is set to $\varepsilon = 10^{-6}$. Figure 6.1(a) illustrates how the (periodic) initial condition $v_0(x) = \frac{1}{2}(\cos(\frac{9}{2}\pi x) + 1)$ (henceforth referred to as "IC1") is propagated to the right (in space) and diffuses over time. In Figure 6.1(b), we see the solution error following convergence of Parareal and GParareal—notice that while the error from GParareal is worse than Parareal, it does converge in fewer than half the iterations (4 versus 9). This again demonstrates that the emulation process in GParareal works very well for a much larger system of equations than has been previously tested. More detailed numerical results for these simulations are shown in the first row of Table 6.1(a-b). The second rows of these tables are results obtained when solving for a different initial condition $v_0(x) = \frac{1}{2}(\cos(\frac{3}{2}\pi x) + 1)$ (referred to as "IC2").

What is noticeable from the GParareal simulations of IC1 and IC2 is that even though speedup exceeds that of Parareal in both cases, it is severely hindered

Table 6.1: Numerical wallclock time, speedup, and efficiency results obtained solving the viscous Burgers' equation (6.1) with GPararealfor different initial conditions and legacy data—refer to the main text for details of each of the problems solved. Theoretical results calculated using (2.11)–(2.13) and (5.17)–(5.19) are shown in brackets. All timings are measured in seconds and have been averaged over five independent simulations.

Problem	N	k_{para}	$T_{\mathcal{G}}$	$T_{\mathcal{F}}$	T_{GP}	T_{serial}	T_{para}	S_{para}	E_{para}
IC1	128	9	4.70E−5	72.22	—	9.24E3	754.78 (650.04)	12.25 (14.22)	0.10 (0.11)
IC2	128	10	4.70E−5	72.34	—	9.27E3	858.00 (723.42)	10.79 (12.80)	0.08 (0.10)

(a) Parareal results

Problem	N	k_{GPara}	$T_{\mathcal{G}}$	$T_{\mathcal{F}}$	T_{GP}	T_{serial}	T_{GPara}	S_{GPara}	E_{GPara}
IC1	128	4	4.70E−5	72.22	212.76	9.24E3	528.82 (501.67)	17.48 (18.43)	0.14 (0.14)
IC2	128	5	4.70E−5	72.34	368.02	9.27E3	774.32 (729.74)	11.96 (12.69)	0.09 (0.10)
IC2(L)	128	5	7.30E−5	71.88	1.50E3	9.20E3	1.89E3 (1.86E3)	4.87 (4.95)	0.04 (0.04)
IC2(LHS)	128	4	8.20E−5	72.64	1.20E3	9.30E3	1.55E3 (1.49E3)	6.00 (6.24)	0.05 (0.05)

(b) GPararealresults

Problem	N	k_{GPara}	$T_{\mathcal{G}}$	$T_{\mathcal{F}}$	T_{GP}	T_{serial}	T_{GPara}	S_{GPara}	E_{GPara}
IC1	128	4	6.50E−5	72.06	54.79	9.22E3	367.91 (343.06)	25.07 (26.89)	0.20 (0.21)
IC2	128	5	6.00E−5	74.33	90.91	9.51E3	501.02 (462.62)	19.00 (20.57)	0.15 (0.16)
IC2(L)	128	5	6.90E−5	72.02	366.64	9.22E3	791.23 (726.77)	11.65 (12.68)	0.09 (0.10)
IC2(LHS)	128	4	6.20E−5	72.09	256.06	9.23E3	569.89 (544.46)	16.19 (16.95)	0.13 (0.13)

(c) Optimised GPararealresults

by size of T_{GP}—accounting for approximately 40% and 48% of the total runtime T_{GPara}, respectively. Clearly, the proportion of time spent on GP optimisation and querying is far too high (compared to $T_{\mathcal{F}}$) and so we need to optimise the way GPararealcarries out the emulation process. If we delve deeper into the breakdown of T_{GP} itself (for IC1) we find that 15% of the GP runtime is due to optimisation of hyperparameters and 85% is due to querying during the PC step (similar results for IC2). Hyperparameter optimisation is relatively cheap due to the low amount of acquisition data and because the process has been parallelised across the dimensions d, i.e. we optimise for each scalar output GP in parallel (recall Section 5.2.2). While querying the GP in the PC step, i.e. calculating the Gaussian posterior, is by itself computationally cheap, it requires the inversion (or rather Cholesky decomposition) of a covariance matrix each time—recall (5.5) and (5.6). This process is repeated sequentially on up to N time slices, d GPs, and k iterations, leading to the excessively large GP runtimes we see in Table 6.1(b).

One way to avoid the repeated inversion of the same covariance matrix at

each time slice t_n in (5.5) and (5.6) is to pre-compute its Cholesky decomposition immediately following the optimisation of the hyperparameters. We calculate and store the decomposition for each of the d GPs, ready to be called in the PC step[1]. This will reduce the total number of Cholesky decompositions required in the PC step from $\mathcal{O}(kdN)$ to $\mathcal{O}(k)$. In the first two rows of Table 6.1(c), we again solve Burgers' equation for IC1 and IC2 using this newly optimised version of GPararealGP. In both cases, we immediately see that T_{GP} decreases by a factor of four, leading to a dramatic 50% increase in speedup. The GP emulators now account for 15% and 18% of total runtime T_{GPara}, respectively. As discussed in Section 5.1.3, these results highlight just how important it is to include training runtimes when using any kind of learning method within a PinT simulation.

In the third rows of Table 6.1(b-c) are results from solving Burgers' equation for IC2 using the 506 legacy data points obtained from solving for IC1 (denoted "IC2(L)"). As before, the optimised version of GPararealGP cuts the GP runtime down by a factor of four compared to the non-optimised version, doubling the realised speedup. However, the use of the legacy data did not result in a reduction in the number of iterations ($k = 5$) and so speedup is actually lost—compare speedup in rows two and three in Table 6.1(c). This loss is directly attributed to the higher optimisation/conditioning costs incurred by using additional training data, indicating that legacy data is only really useful when it results in a reduction in k.

This effect can be seen when we use legacy data generated from Latin hypercube sampling[2] (LHS). To do this, we generate 506 points using the LHS scheme and propagate each one using both $\mathcal{F}_{\Delta T}$ and $\mathcal{G}_{\Delta T}$ (can be done in parallel). Note, however, that with only $N = 128$ processors available, this data generation step took approximately the same amount of time as four $\mathcal{F}_{\Delta T}$ runs to generate (i.e. $506/128 \approx 3.95$)—equivalent to ≈ 290 seconds of runtime. Although this cost is not accounted for in our runtime calculations, it is important to remember that data generation is not "free"—something that is often assumed in other learning-based PinT methods (Agboh et al., 2020; Nguyen and Tsai, 2022; Yalla and Engquist, 2018). In any case, the results (IC2(LHS)) are in fact an improvement over the IC2(L) experiment, with convergence occurring in 4 iterations instead of 5—see Table 6.1(b-c), final row. While this demonstrates the effectiveness of using legacy data to reduce k, the speedup generated is still less than that obtained without the legacy data due to the high training costs.

[1]This will of course require the storage of d lower triangular matrices each iteration, however, given that these matrices were previously being calculated and discarded each iteration anyway, it will not lead to an overall increase in memory requirements.

[2]Latin hypercube sampling is a method of generating near-random samples from a d-dimensional unit hypercube $[0, 1]^d$. If one partitions the unit hypercube into s^d smaller (equal size) hypercubes, then LHS will return s samples with each one lying in its own unique smaller hypercube (McKay et al., 1979). For example, if $d = 1$ then there should be one unique sample in each of the s subintervals $(0, 1/s), (1/s, 2/s), \ldots, (1 - 1/s, 1)$.

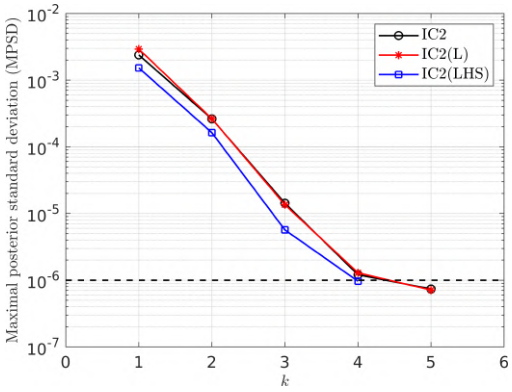


Figure 6.2: MPSD at each iteration k in simulations of GPareareal when solving Burgers' equation (6.1). Results obtained when solving problems IC2, IC2(L), and IC2(LHS) (see Table 6.1(c)). The dashed black line indicates the stopping tolerance $\varepsilon = 10^{-6}$.

Finally, we briefly examine the MPSD (recall Section 5.4) of the GP emulators used in the IC2, IC2(L), and IC2(LHS) experiments from Table 6.1(c). The MPSD at each iteration of these experiments is plotted in Figure 6.2, showing that the "accuracy" of the IC2 and IC2(L) GP emulators are very similar. We note that the MPSD is not a true measure of emulator accuracy but rather an indicator of accuracy close to the locations at which the emulators are queried. In the IC2(LHS) case, we see a slight improvement in accuracy due to the way that the data samples have been drawn near-uniformly across the unit hypercube, suggesting that the LHS legacy data is beneficial for GPareareal in this particular example.

6.2.2 Two-dimensional FitzHugh–Nagumo model

We now solve a larger system, examining how GPareareal performs as we increase the number of spatial points d used in the discretisation of our PDE system. We will use the optimised version of GPareareal discussed in previous section to solve the 2D FHN model:

$$
\begin{aligned}
\text{PDEs} \quad & v_t = a\nabla^2 v + v - v^3 - w - c \\
& w_t = \tau\left(b\nabla^2 w + v - w\right) && (\boldsymbol{x}, t) \in (-L, L)^2 \times (t_0, T] \\
\text{ICs} \quad & v(\boldsymbol{x}, t_0) = v_0(\boldsymbol{x}) \\
& w(\boldsymbol{x}, t_0) = w_0(\boldsymbol{x}) && \boldsymbol{x} \in [-L, L]^2 \\
\text{BCs} \quad & v\left((x, -L), t\right) = v\left((x, L), t\right) \\
& v\left((-L, y), t\right) = v\left((L, y), t\right) \\
& v_y\left((x, -L), t\right) = v_y\left((x, L), t\right) \\
& v_x\left((-L, y), t\right) = v_x\left((L, y), t\right) && t \in [t_0, T]
\end{aligned}
\tag{6.3}
$$

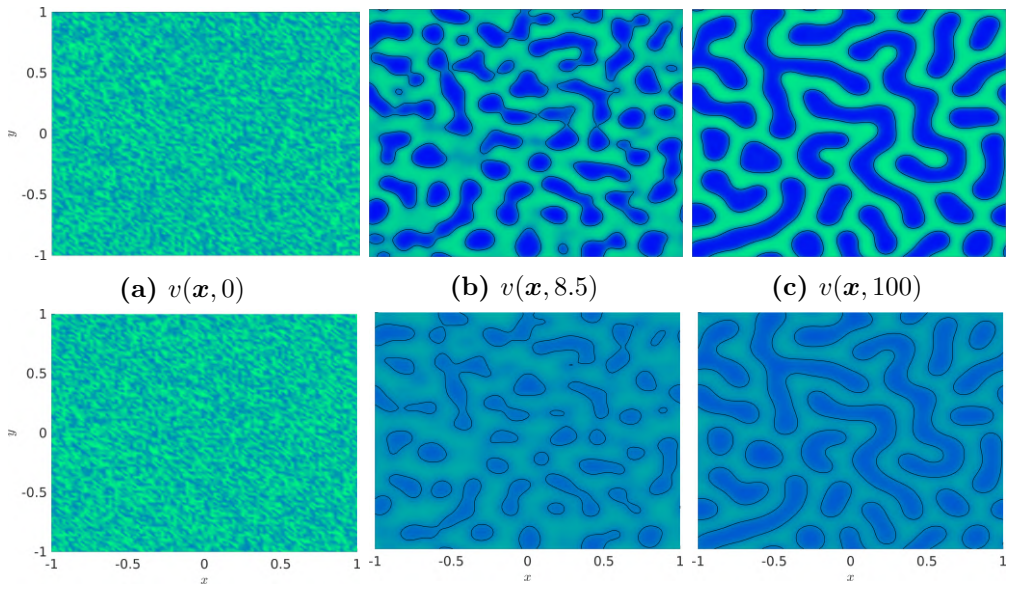
for $v(\boldsymbol{x}, t)$ and $w(\boldsymbol{x}, t)$ over a square domain. Note we have only listed the boundary conditions for $v(\boldsymbol{x}, t)$—identical conditions are required on $w(\boldsymbol{x}, t)$. This system is a spatially-dependent extension of the FHN model seen in (5.21). We denote the Laplacian $\nabla^2 v = v_{xx} + v_{yy}$ and fix parameters $(a, b, c, \tau) = (2.8\text{E}{-4}, 5\text{E}{-3}, 5\text{E}{-3}, 10)$ (Krämer et al., 2022).

We discretise both spatial dimensions using d equally spaced points, defining $\boldsymbol{v} = (v((x_0, y_0), t), \ldots, v((x_{d-1}, y_0), t), \ldots, v((x_0, y_{d-1}), t), \ldots, v((x_0, y_{d-1}), t))^\intercal$ (similarly for $\boldsymbol{w}$). We will denote the Kronecker product of matrices A and B, of dimensions $\ell \times m$ and $r \times s$, as

$$A \otimes B = \begin{bmatrix} a_{1,1}B & \cdots & a_{1,m}B \\ \vdots & \ddots & \vdots \\ a_{\ell,1}B & \cdots & a_{\ell,m}B \end{bmatrix},$$

which is a block matrix of dimension $\ell r \times ms$. Using this definition we can write the differentiation matrices as $D_{xx} = \mathbb{I}_d \otimes D_{xx}^{(2)}$ and $D_{yy} = D_{yy}^{(2)} \otimes \mathbb{I}_d$. This then allows us to write (6.3) as the following system of $\tilde{d} = 2d^2$ ODEs

$$\begin{aligned} \frac{\mathrm{d}\boldsymbol{v}}{dt} &= a(D_{xx} + D_{yy})\boldsymbol{v} + \boldsymbol{v} - \boldsymbol{v}^3 - \boldsymbol{w} - c\mathbf{1}, \\ \frac{\mathrm{d}\boldsymbol{w}}{dt} &= \tau\left(b(D_{xx} + D_{yy})\boldsymbol{w} + \boldsymbol{v} - \boldsymbol{w}\right), \end{aligned} \qquad t \in (t_0, T].$$



(a) $v(\boldsymbol{x}, 0)$ **(b)** $v(\boldsymbol{x}, 8.5)$ **(c)** $v(\boldsymbol{x}, 100)$

(d) $w(\boldsymbol{x}, 0)$ **(e)** $w(\boldsymbol{x}, 8.5)$ **(f)** $w(\boldsymbol{x}, 100)$

Figure 6.3: Numerical solutions obtained solving the 2D FHN system (6.3) over $\boldsymbol{x} \in [-1, 1]^2$ at times (a,d) $t = 0$, (b,e) $t = 8.5$, and (c,f) $t = 100$ using the fine solver with $N_{\mathcal{F}} = 51,200$. Solution values for $v(\boldsymbol{x}, t)$ (top panels) and $w(\boldsymbol{x}, t)$ (bottom panels) are coloured from blue (-1) to green $(+1)$ with contours representing zero values.

Table 6.2: Numerical wallclock time, speedup, and efficiency results obtained solving the 2D FHN system (6.3) with (a) Parareal, (b) optimised GParareal, and (c) further optimised GParareal for increasing spatial resolution $\tilde{d}$. Theoretical results calculated using (2.11)–(2.13) and (5.17)–(5.19) are shown in brackets. All timings are measured in seconds.

$\tilde{d}$	k_{para}	$T_{\mathcal{G}}$	$T_{\mathcal{F}}$	T_{GP}	T_{serial}	T_{para}	S_{para}	E_{para}
128	6	8.90E−5	200.08	—	1.02E5	1.51E3 (1.20E3)	67.93 (85.31)	0.13 (0.17)
200	5	1.13E−4	264.55	—	1.35E5	1.60E3 (1.32E3)	84.63 (102.37)	0.17 (0.20)
288	6	1.43E−4	342.73	—	1.75E5	2.42E3 (2.06E3)	72.51 (85.31)	0.14 (0.17)
392	5	1.78E−4	436.41	—	2.23E5	2.44E3 (2.18E3)	91.40 (102.37)	0.18 (0.20)
512	5	1.75E−4	539.54	—	2.76E5	2.97E3 (2.70E3)	92.93 (102.38)	0.18 (0.20)
20000	3	2.12E−1	2445.90	—	1.25E6	9.91E3 (7.77E3)	126.35 (161.15)	0.25 (0.31)

(a) Parareal results

$\tilde{d}$	k_{GPara}	$T_{\mathcal{G}}$	$T_{\mathcal{F}}$	T_{GP}	T_{serial}	T_{GPara}	S_{GPara}	E_{GPara}
128	3	1.64E−4	199.86	5.23E2	1.02E5	1.28E3 (1.12E3)	80.18 (91.11)	0.16 (0.18)
200	3	1.89E−4	264.09	8.39E2	1.35E5	1.80E3 (1.63E3)	75.27 (82.89)	0.15 (0.16)
288	4	2.22E−4	341.99	2.40E3	1.75E5	3.97E3 (3.77E3)	44.06 (86.41)	0.09 (0.09)
392	5	2.71E−4	434.38	5.63E3	2.22E5	8.06E3 (7.80E3)	27.61 (28.51)	0.05 (0.06)
512	4	2.85E−4	536.99	4.38E3	2.75E5	6.74E3 (6.52E3)	40.82 (42.12)	0.08 (0.08)

(b) Optimised GParareal results

$\tilde{d}$	k_{GPara}	$T_{\mathcal{G}}$	$T_{\mathcal{F}}$	T_{GP}	T_{serial}	T_{GPara}	S_{GPara}	E_{GPara}
128	3	1.54E−4	199.82	4.99E2	1.02E5	1.26E3 (1.10E3)	81.21 (93.13)	0.16 (0.18)
200	3	1.91E−4	265.17	7.08E2	1.36E5	1.66E3 (1.50E3)	81.76 (90.27)	0.16 (0.18)
288	4	2.27E−4	343.82	1.77E3	1.76E5	3.37E3 (3.15E3)	52.23 (55.95)	0.10 (0.11)
392	5	2.70E−4	437.42	4.02E3	2.24E5	6.49E3 (6.20E3)	34.53 (36.11)	0.07 (0.07)
512	4	2.94E−4	541.36	3.23E3	2.77E5	5.61E3 (5.40E3)	49.43 (51.35)	0.10 (0.10)

(c) Further optimised GParareal results

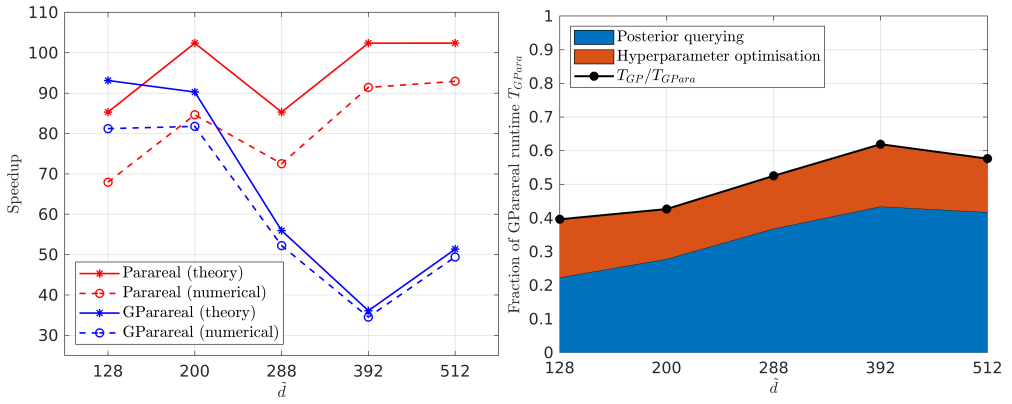
In the following experiments, we fix $N = 512$, $L = 1$, integrate over $t \in [0, 100]$ and use solvers $\mathcal{F}_{\Delta T}$ = RK8 and $\mathcal{G}_{\Delta T}$ = RK1 (the stopping tolerance is again $\varepsilon = 10^{-6}$). We use $N_{\mathcal{F}} = 5.12 \times 10^8$ fine and $N_{\mathcal{G}} = 2048$ coarse time steps with initial conditions chosen uniformly at random such that $\boldsymbol{v}(0), \boldsymbol{w}(0) \in [0, 1]^{d^2}$ (but fixed to be consistent across experiments). In Figure 6.3, we plot solutions to (6.3) using a total of $\tilde{d} = 20,000$ points in space ($d = 100$), illustrating how patterns begin to

form as time progresses.

Table 6.2(a-b) displays results obtained when solving the 2D FHN system (6.3) for increasing spatial resolution $\tilde{d}$ when using Parareal and GParareal. The first thing to notice is that $k_{\text{GPara}} \leqslant k_{\text{para}}$ for each $\tilde{d}$ (ignoring the $\tilde{d} = 20,000$ case for the moment), demonstrating once again that emulation can accelerate convergence. Only in the $\tilde{d} = 128$ case do we see this translate into improved speedup over Parareal. For $\tilde{d} > 128$, we observe that speedup drops off dramatically, driven purely by large values of T_{GP} which account for between $47 - 70\%$ of the total GParareal runtime in these experiments.

To again try and tackle the issue of high GP runtimes, we "further optimise" the implementation of GParareal by storing the distance matrices used in the calculation of the covariance matrices $K(\boldsymbol{x}, \boldsymbol{x})$. In short, calculating the (squared Euclidean) distance matrix (recall (5.2) for the 1D case) between points in the input dataset is computationally expensive once the dataset $\boldsymbol{x}$ (and d) becomes large. Therefore, we now pre-calculate these distances once per iteration to be used in the covariance kernel function whenever required during hyperparameter optimisation/querying. We can see in Table 6.2(c) that this helps reduce T_{GP} (more when $\tilde{d}$ is large), resulting in speedup increases of up to 25% in the best case ($\tilde{d} = 392$).

Even after making these further optimisations we can see that the speedup results from GParareal are still far below those attained by Parareal for increasing $\tilde{d}$—see Figure 6.4(a). We illustrate the (still) significant proportion of total GParareal runtime that posterior querying and hyperparameter optimisation take up in Figure 6.4(b). We can see that the hyperparameter optimisation takes around 20%



(a) **(b)**

Figure 6.4: (a) Numerical speedup results from Parareal (dashed red) and GParareal (dashed blue) plotted with theoretical results (solid), calculated using (2.12) and (5.18), respectively. (b) Fraction of GParareal runtime T_{GPara} taken up by querying the GP posteriors (blue), the hyperparameter optimisation (red), and their sum (solid black). Parareal results from Table 6.2(1) and both sets of GParareal results from Table 6.2(c).

of runtime and is (approximately) independent of $\tilde{d}$ because it is carried out in parallel for each GP. There does not seem to be much room for improvement in the current implementation without switching to alternate optimisation routines or perhaps increasing tolerances in the routine itself (which may lower the accuracy of the emulators). We can clearly see that fraction of time the posterior querying takes up does, however, increase with $\tilde{d}$ because the total number of times GPareareal queries the emulators is $\mathcal{O}(kdN)$. This includes the savings made when storing the Cholesky decompositions. In the worst case, this takes up over 40% of runtime and so finding a way to eliminate or reduce the dependence on $\tilde{d}$ is paramount, although unclear at present.

Notice also that GPareareal is quite severely hindered in terms of how large $\tilde{d}$ can be. The solutions in Figure 6.3 were calculated (using the fine solver) for $\tilde{d} = 20,000$, whereas the most GPareareal can solve for is $\tilde{d} \leqslant N = 512$. While we could increase N to increase $\tilde{d}$ (which would require more processors), GPareareal would still suffer from the extremely high GP runtimes we have seen here. While one can always interpolate lower spatial resolution solutions onto a higher resolution grid, any fine-scale solution behaviour will almost certainly be lost. The final row of Table 6.2(a) shows that Parareal faces no such restriction on the size of $\tilde{d}$, returning excellent speedup results for $\tilde{d} = 20,000$ (note that to account for the increase in spatial resolution, we had to increase the number of coarse steps to $N_{\mathcal{G}} = 1.024 \times 10^5$ for this particular simulation).

6.3 Discussion and further work

In this chapter, we extended our numerical investigation of GPareareal by applying it to two nonlinear PDE problems: the viscous Burgers' equation (6.1) and the 2D FHN system (6.3). When solving Burgers' equation for different initial conditions (using 129 spatial points), we found that GPareareal converged in at least half the number of iterations compared to Parareal, leading to additional numerical speedup. These speedup results were severely limited by the (serial) cost of training/querying the 129 GP emulators, which took up over 40% of the total GPareareal runtime. To optimise the process, we modified GPareareal so that it would store the Cholesky decomposition (one for each emulator) of each covariance matrix $K(\boldsymbol{x}, \boldsymbol{x})$ following hyperparameter optimisation—avoiding the need to recompute it each time the emulators are queried. While this did reduce GP runtimes by up to four times and boosted parallel speedup in all cases, the emulation process still accounted for over 15% of total GPareareal runtime.

In the cases where legacy data was used, coming from either a previous solve or from LHS, the GP training costs rose significantly—even when storing the Cholesky decompositions. We found that using legacy data (in addition to acquisition data)

each iteration, did not reduce the iteration count significantly enough (if at all) to reduce GParareal runtimes, leading to severe speedup degradation. This leads us to conclude that legacy data is most useful only when it will significantly reduce the iteration count (as we saw in Section 5.3), although this will be unknown *a priori* to simulation. One option to reduce training times could be to use the legacy data during only the first iteration to try to embed some of the valuable legacy information in the solution early on, without having to keep reusing it at high cost during future iterations. The idea is that wasteful training may be taking place each iteration if many of the legacy data points lay "far away" from the exact solution—recall the discussion in Section 5.2.4.

Using up to 512 spatial points, we found that, once again, GParareal can converge in fewer iterations than Parareal when solving the 2D FHN system but took a higher overall wallclock time to do so. We found that with increasing d, the number of posterior queries made to the GP increases and so even when storing Cholesky decompositions and distance matrices, the total emulation runtimes ran in excess of 60% of the total GParareal runtime. The main takeaway from these numerical experiments is that the importance of incorporating training runtimes is paramount when trying to quantify the performance of PinT methods that make use of learning-based methods. In retrospect, it would also make sense to go back and apply this (further) optimised version of GParareal to the experiments from Chapter 5 to observe their impact on simulation runtimes and speedup results.

A further feature hindering GParareal is that it is limited to solving systems of size $d \leqslant N$ in order to carry out hyperparameter optimisation for each of the GP emulators in parallel. This limited us to solving the 2D FHN system on a 16×16 grid, whereas Parareal could scale to grids of size 100×100 (and higher) without suffering such issues. While one could in theory solve systems of size $d \geqslant N$, it would require training the GP emulators in parallel batches, e.g. if $d = 2N$ then one trains d GPs then another d GPs straight after. However, this would most likely lead to excessively large runtimes as mentioned before.

The key to unlocking the true potential of GParareal is to identify an emulation method (not necessarily a GP) that is cheap, accurate, and scales well with high-dimensional datasets. This draws a striking similarity to the quest to find cheap and accurate coarse solvers for Parareal—recall the discussion in Section 5.1.3. In GParareal, the emulators are re-trained using all acquisition data (and legacy data if available) each iteration, leading to extremely high runtimes. A better solution would be to find an emulator that can avoid being re-trained on the entire dataset each iteration but rather can re-train itself (quickly) in light of only the new data (a more "online" type of learning-based method). In addition, one could try to further reduce GP costs by selecting a subset of the data that "best covers" the full dataset or perhaps filter out data points that lay "far away" from the exact

solution (particularly those that do not improve the accuracy of the emulator near the solution). This could be done using a clustering algorithm (or similar), however, the benefits of any faster emulation would of course need to outweigh any decrease in emulator accuracy that may be detrimental to the final iteration count in GPararealII.

Chapter 7

Discussion and outlook

The primary focus of this thesis has been the development and testing of probabilistic PinT algorithms for more efficiently solving computationally expensive IVPs. The aim was to accelerate the convergence of Parareal, a well-studied deterministic PinT algorithm, to solve IVPs faster and generate probabilistic solutions that capture numerical uncertainty. To do this, we used existing sampling- and learning-based techniques from PN to harness the valuable information contained within the fine and coarse solution (acquisition) data generated during a Parareal simulation. In this final chapter, we will discuss the contribution of these algorithms towards the aims set out in Chapter 1 and subsequently draw conclusions regarding their future viability as PinT algorithms.

7.1 Contribution toward original aims

7.1.1 SParareal

Inspired by the first PinT idea proposed by Nievergelt (1964), we developed SParareal, a sampling-based PinT method that randomly perturbs solutions in Parareal to try to more efficiently explore the solution space. It works by drawing M samples (at each time slice) from probability distributions, constructed using the most recently obtained acquisition data, and propagating them forward in time (in parallel) using the fine solver. The set of samples yielding the "most continuous" trajectory over time is then used in the correction term in Parareal's PC, with the hope of reducing the number of iterations k until convergence.

> **Conclusion 1.** *SParareal should always converge in the same number of iterations as Parareal or fewer (assuming a minimum level of sampling).*

In Chapter 3, we derived, analysed, and tested SParareal on a number of low-dimensional nonlinear ODEs (Aim I), observing that it could indeed converge in fewer iterations than Parareal (Aim II). For all IVPs tested we observed that as

the number of samples M increased, the expected number of iterations required to converge decreased (we took the expected value due to the stochastic nature of the algorithm). With a computational complexity almost identical to Parareal, this meant that SParareal could locate a solution in a shorter wallclock time compared to Parareal if it converged in fewer iterations.

The reason why we expect SParareal to perform no worse than Parareal is because it uses the deterministic PC solution (recall Section 3.2) as well as the random samples to explore the solution space. In the worst case, this means that even if none of the random samples are chosen as the optimal solution candidates, SParareal is able to fall back and utilise (values close to, but not exactly) the deterministic Parareal solutions. In the best case, a combination of the samples and PC solutions are used, leading to accelerated convergence. We note the minimum level of sampling due to the exceptional case in Appendix C.2 where SParareal took one more iteration to converge than Parareal when $M = 2$ samples were taken. The reason why this occurs in a small fraction (2.5%) of simulations is still unclear, though we speculate that perhaps the single random sample taken strays further from the true solution during early iterations, requiring an additional iteration to be corrected later on.

Conclusion 2. *The probabilistic solutions generated by SParareal are accurate (in mean-square) with respect to the serially obtained fine (exact) solution.*

In Chapter 4, we derived superlinear and linear mean-square error bounds to verify that the probabilistic solutions (Aim IV) obtained by SParareal are accurate (Aim III). We found that using additive noise, i.e. state-independent perturbations, in SParareal was not enough to guarantee accelerated convergence. In addition, this approach generated solutions with a hard lower bound on their numerical error, proportional to the bound on the absolute moments of the perturbations used—recall Remark 4.10. To obtain a higher degree of accuracy (as well as accelerated convergence), the results showed that the sampling rules outlined in Section 4.1.2, i.e. the state-dependent perturbations, were required. These rules ensured that the probability distributions contracted around the exact solution as the SParareal iterations progressed, ensuring increasing accuracy. This meant that following multiple simulations of SParareal, we could obtain a distribution of solutions to the IVP just as the sampling-based ODE solvers proposed by Conrad et al. (2017) did. While these distributions do not really have any real mathematical interpretation, they could be useful when exploring IVP solutions that exhibit chaotic behaviour or exist near stable/unstable manifolds.

With regard to which sampling rule should be used in SParareal, we found that as the level of sampling increased they all performed similarly. That being said, the choice of sampling rule is flexible with respect to the particular IVP being solved. For example, one could linearly combine different sampling rules or perhaps choose a particular distribution family that satisfies known properties of the IVP solution (e.g.

non-negativity). We do, however, suggest that the marginal standard deviations be chosen such that they contract as the iterations progress (for the reasons mentioned above).

Conclusion 3. *SParareal can be used to solve any large-scale IVP that Parareal is suitable for.*

The reasoning behind this final conclusion (Aim V) is that Parareal could be slightly modified to enable idle processors (following the first iteration) to carry out sampling and propagation at little to no extra computational cost. The performance of this SParareal-type scheme should be no worse than standard Parareal (ideally better with the additional sampling, recall Conclusion 1) and, even if there is no reduction in k, the algorithm will return a stochastic trajectory—beneficial for reasons mentioned above. If additional processors (beyond the one required for each time slice) are available, then even more sampling can be carried out and one can deploy standard SParareal.

We expect SParareal to be able to solve IVPs that Parareal is suited for and most likely face issues with those that Parareal struggles with (e.g. non-diffusive IVPs) due to its almost identical structure. As the size of the IVP (i.e. the number of ODEs being solved) becomes large, however, we expect the sampling to become less efficient due to the curse of dimensionality. Therefore to reduce the number of iterations, increasingly more samples will be required to explore the solution space effectively. As suggested before though, there is no harm (in terms of computational cost) in carrying out sampling using idle processors given there is a (small) chance of faster convergence.

With SParareal we have demonstrated that using (guided) random perturbations can indeed help accelerate the convergence of Parareal and return probabilistic solutions. While the sampling process may not scale well for large-scale IVPs, we believe there is no harm in putting idle processors in Parareal to good use by sampling and exploring more of the solution space. In the future, further experiments on how to determine which algorithm to use given a fixed number of, say NM, processors would be of interest. For a larger-scale IVP, how does Parareal perform when using NM processors (and therefore NM time slices) compared to SParareal, which will use M processors to carry out sampling in N (larger) time slices? Answering this would help determine which algorithm more efficiently uses each of its processors and possibly how to more optimally assign samples throughout the time interval.

7.1.2 GParareal

The GParareal algorithm is a learning-based PinT method that uses GP emulators to infer the correction term in Parareal's PC. Each iteration, the emulators are trained

(i.e. the kernel hyperparameters are optimised) using *all* available acquisition and legacy data from prior simulations (if any). Having avoided "throwing away" valuable solution data like Parareal (and SParareal[1]), the emulators are then ready to be queried in the PC step, ideally providing more accurate corrections than Parareal and enabling more rapid convergence.

Conclusion 4. *While GParareal did achieve accelerated convergence over Parareal for most of the IVPs tests, realising an increase in numerical speedup is contingent on the total GP emulator runtime being small compared to the fine solver.*

In Chapters 5 and 6, we derived and analysed GParareal (Aim I), demonstrating that it could converge in fewer iterations than Parareal using only acquisition data for a number of different nonlinear ODEs and PDEs (Aim II). Convergence in even fewer iterations could be achieved using legacy data (from prior simulations) in the ODE experiments, however, this proved less effective for the PDE problems (even using pseudo-randomly generated legacy data). Results suggested that the acquisition data rather than legacy data was most effective at accelerating convergence in GParareal (especially since additional training time was required for simulations using legacy data). In line with the complexity analysis, experiments verified that realising the maximum possible speedup with GParareal was contingent on the cost of training the GP emulators being small (compared to the cost of running the fine solver).

The PDE experiments in Section 6.2 showed that even when training the d emulators in parallel and storing distance matrices/Cholesky decompositions (used repeatedly during querying in the PC), the emulation process still took up large proportions of the total GParareal runtime. In addition, we found that the emulation process worked best (learned $\mathcal{F}_{\Delta T} - \mathcal{G}_{\Delta T}$ most effectively) for IVPs with more slowly varying/smoother dynamics (see Sections 5.3.1, 5.3.3, 6.2.1, and 6.2.2) and less well for IVPs that exhibited more fast moving/chaotic dynamics (recall Sections 5.3.2, 5.3.4, and 5.4). We will continue discussing the implications surrounding the emulation process in Conclusions 6 and 7.

Conclusion 5. *The deterministic solutions generated by GParareal are accurate with respect to the serially obtained fine (exact) solution. In fact, accuracy is proportional to the fill distance of the training dataset.*

In all numerical experiments, the solutions obtained by GParareal were accurate with respect to the fine solver solution and of a comparable order of accuracy to the solutions obtained by Parareal. Using an existing result on the consistency of

[1]Note that we did not directly compare SParareal and GParareal because we expected the performance of SParareal (for low sample numbers at least) to be similar to that of Parareal—recall Conclusion 1.

the GP posterior mean, we derived an error bound (in Section 5.2.4) showing that the accuracy of solutions is directly proportional to the fill distance of the training dataset (Aim III). This means that each iteration (as the size of the training set increases), the fill distance should decrease, thereby increasing the accuracy of the emulator and the GParareal solution—this agreed with numerical simulations.

While we did make use of learning-based probabilistic methods to develop GParareal, it did not return probabilistic solutions to the IVPs themselves (Aim IV). This is due to the fact that we approximate the GP posterior in (5.14) by its (deterministic) expected value, ignoring the posterior variance, in order to propagate a single value through the PC update in (5.16c). Ideally, we would be able to propagate the full Gaussian posterior through the PC update to return a probabilistic solution, however, we were unable to identify a (computationally efficient) method to do this. Instead of taking the mean, one way to (forcibly) obtain probabilistic solutions in GParareal could be to draw a random sample from the Gaussian posterior in (5.14), however, it is unclear how this scheme would perform or even if it would converge to the correct solution (especially if the posterior variance at the sampling location is large). Further investigation is required to identify how to embed a computationally efficient method for propagating uncertainty within the Parareal framework (or perhaps another PinT method).

Conclusion 6. *Accounting for data generation, hyperparameter optimisation, and querying costs is of the utmost importance when simulating IVPs using learning-based PinT methods.*

The purpose of the PDE experiments in Section 6.2 was to quantify the impact of GP emulation costs on realisable parallel speedup from GParareal—comments on which were notably absent in other learning-based Parareal variants (refer back to Section 5.1.3). Numerical experiments showed that even if GParareal converges in fewer iterations than Parareal, which saves a number of expensive $\mathcal{F}_{\Delta T}$ runs, the maximum obtainable speedup is drastically reduced by the cost of running the GP emulators. By including these costs in our computational complexity analysis, we hoped to give a more clear picture of the viability of using learning-based methods to accelerate PinT simulations.

These costs are so critical because they scale cubically (in our naive GP implementation) with the number of training data points (which increases each iteration, more so when using legacy data) and with the dimension of the system d (as more posterior querying required). One could try to exploit some well-known GP approximation methods to reduce computational runtimes (recall Section 5.2.3), however, care must be taken not to reduce the accuracy of the emulators too much or else GParareal will not achieve accelerated convergence. Alternatively, we could try to reduce the amount of training data used each iteration by somehow selecting a subset of the

acquisition data that is the "most informative". Again, further experimentation is required to determine whether or not this leads to a reduction in the accuracy of the emulators and therefore GParareal performance.

Conclusion 7. *Before GParareal can begin solving large-scale IVPs, further work is needed to find an emulation method that scales with the size of the system d and is cheap enough to train accurately as the simulation progresses.*

As we saw when solving the 2D FHN system in Section 6.2.2, GParareal is limited to solving systems of relatively small size ($d \leqslant 512$). In addition to the GP costs, it can struggle to accurately infer the correction term, which is a nonlinear d-dimensional function, because the fill distance is difficult to restrict when d is large. One can use more data, by increasing N or using legacy data, however, GParareal will then suffer the issues related to Conclusion 6. Although we did investigate using the Parareal "fallback" in Section 5.4 to aid GParareal when the emulators were insufficiently well-trained, questions still remain over how to set the fallback switching tolerance. To try to improve accuracy in the GP further, one could try using a non-zero mean in the GP prior, i.e. a function of the order of the error between $\mathcal{F}_{\Delta T}$ and $\mathcal{G}_{\Delta T}$—though this would likely come with additional hyperparameter optimisation costs.

The key restriction hindering GParareal is that it can only solve systems of size $d \leqslant N$ as we require a processor for each emulator in order to train them all in parallel. Clearly it is much more desirable to be able to solve systems of size $d \geqslant N$ just as Parareal can, however, it is unclear at present how to do this without the GP runtimes spiralling out of control. One avenue to explore would be to use dimensionality reduction techniques such as principal component analysis to reduce the dimension of the training data and therefore the number of emulators required in GParareal.

The GParareal algorithm has demonstrated that learning-based methods can indeed harness solution data to accelerate the convergence of Parareal as long as training costs remain small. The next natural step in its evolution is to assess how alternative emulators (which need to be cheaper to train and scale better with d) perform within the GParareal framework. The problem of finding the best emulators will be akin to finding the best coarse solvers to use within Parareal and may possibly require some process of trial and error. We believe, however, that with ever faster and more accurate learning-methods becoming available, the search will not be long.

7.2 Outlook for probabilistic PinT algorithms

In summary, the analytical and numerical work carried out in this thesis has demonstrated that probabilistic methods can indeed make use of existing fine and coarse solution data to improve the performance of the Parareal algorithm. We believe that

by tackling some of the aforementioned issues, the performance and scalability of these algorithms can certainly be improved. The hope is that this will lead to more efficient PinT algorithms that can avoid the need to re-simulate costly IVP solutions from scratch and instead re-use existing solution data to reduce IVP simulation runtimes. This should also help pave the way for a fully probabilistic PinT algorithm that can return a posterior distribution over the solution to an IVP at any given time. Having investigated using methods from PN within Parareal to achieve this goal, the next step could be to look into the converse—how can PinT techniques be embedded within existing sampling- or learning-based PN methods? In any case, we hope that the ideas put forth in this thesis will contribute to the successful development of future probabilistic PinT algorithms.

Appendices

A Rates of convergence

Here, we briefly recall some definitions on the rates of convergence of real sequences. These rates will be referred to frequently in discussions on the convergence of Parareal, SParareal, and GParareal.

Suppose $\{\boldsymbol{x}_k\}_{k\in\mathbb{N}_0}$ is a sequence, taking values in $\mathbb{R}^d$, that converges to $\boldsymbol{x}^* \in \mathbb{R}^d$. The convergence of this sequence is said to be *linear* if there exists $r \in (0,1)$ such that

$$\frac{\|\boldsymbol{x}_{k+1} - \boldsymbol{x}^*\|}{\|\boldsymbol{x}_k - \boldsymbol{x}^*\|} \leqslant r,$$

for all k sufficiently large. Similarly, the convergence is said to be *superlinear* if

$$\lim_{k\to\infty} \frac{\|\boldsymbol{x}_{k+1} - \boldsymbol{x}^*\|}{\|\boldsymbol{x}_k - \boldsymbol{x}^*\|} = 0.$$

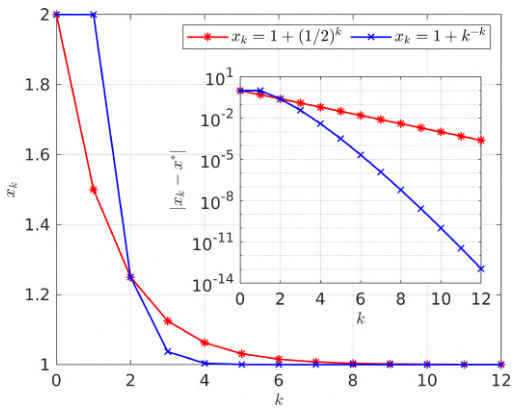


Figure A.1: The sequences x_k given in (A.1) plotted against k (sequence (i) in red and (ii) in blue). Inset: corresponding absolute errors between the values of each sequence and the limit $x^* = 1$. Sequence (i) converges linearly whilst sequence (ii) converges superlinearly.

Note that $\|\cdot\|$ is some norm on $\mathbb{R}^d$. For example, consider the sequences

$$(i)\ x_k = 1 + (1/2)^k \quad \text{and} \quad (ii)\ x_k = 1 + k^{-k}. \tag{A.1}$$

Both sequences converge to $x^* = 1$ with the first sequence converging linearly at rate $r = 1/2$ and the second sequence converging superlinearly at rate $(1/k+1)(k/k+1)^k$. In Figure A.1, we plot both sequences converging toward $x^* = 1$ and the corresponding error at each k, demonstrating linear and superlinear convergence (see inset plot). Further details on other rates of convergence can be found in Nocedal and Wright (2006).

B Proof of superlinear error bound for Parareal

Here, we state the proof of Theorem 2.2.

Proof. Define the error $e_n^k \coloneqq \|\boldsymbol{u}(t_n) - \boldsymbol{U}_n^k\|_\infty$. Then, using (2.9c), that $\mathcal{F}_{\Delta T}$ is the exact solver (4.5), and adding and subtracting $\mathcal{G}_{\Delta T}(\boldsymbol{u}(t_n))$, we obtain

$$\begin{aligned} e_{n+1}^{k+1} &= \|\mathcal{F}_{\Delta T}(\boldsymbol{u}(t_n)) - \big(\mathcal{G}_{\Delta T}(\boldsymbol{U}_n^{k+1}) + \mathcal{F}_{\Delta T}(\boldsymbol{U}_n^k) - \mathcal{G}_{\Delta T}(\boldsymbol{U}_n^k)\big) \pm \mathcal{G}_{\Delta T}(\boldsymbol{u}(t_n))\|_\infty \\ &\leqslant \|\big(\mathcal{F}_{\Delta T}(\boldsymbol{u}(t_n)) - \mathcal{G}_{\Delta T}(\boldsymbol{u}(t_n))\big) - \big(\mathcal{F}_{\Delta T}(\boldsymbol{U}_n^k) - \mathcal{G}_{\Delta T}(\boldsymbol{U}_n^k)\big)\|_\infty \\ &\qquad + \|\mathcal{G}_{\Delta T}(\boldsymbol{u}(t_n)) - \mathcal{G}_{\Delta T}(\boldsymbol{U}_n^{k+1})\|_\infty, \end{aligned}$$

where the second line follows by the triangle inequality. Applying (4.7) to the first term and the Lipschitz condition (4.8) to the second, we are left with the double recursion

$$e_{n+1}^{k+1} \leqslant A e_n^k + B e_n^{k+1}, \quad e_{n+1}^0 \leqslant D + B e_n^0, \tag{B.1}$$

where $A = C_1 \Delta T^{p+1}$, $B = L_{\mathcal{G}}$, and $D = C_2 A$ ($C_2 > 0$ constant). This recursion can be solved using the generating function method in Lemma D.3 (setting $\Lambda = 0$), giving us the desired result. □

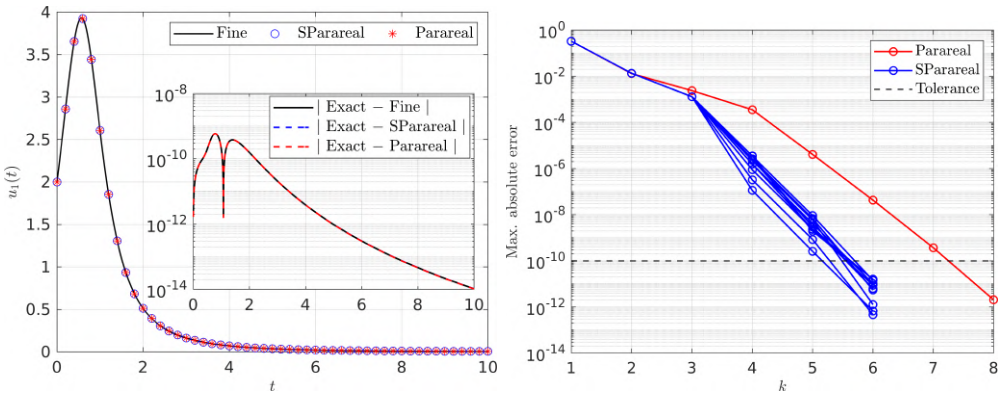
C Additional SParareal experiments

In the following sections, we present additional numerical experiments using SParareal.

C.1 Scalar Bernoulli equation

Consider the nonlinear nonautonomous Bernoulli equation

$$\frac{\mathrm{d}u_1}{\mathrm{d}t} = \frac{2}{1+t} u_1 - t^2 u_1^2, \tag{C.1}$$



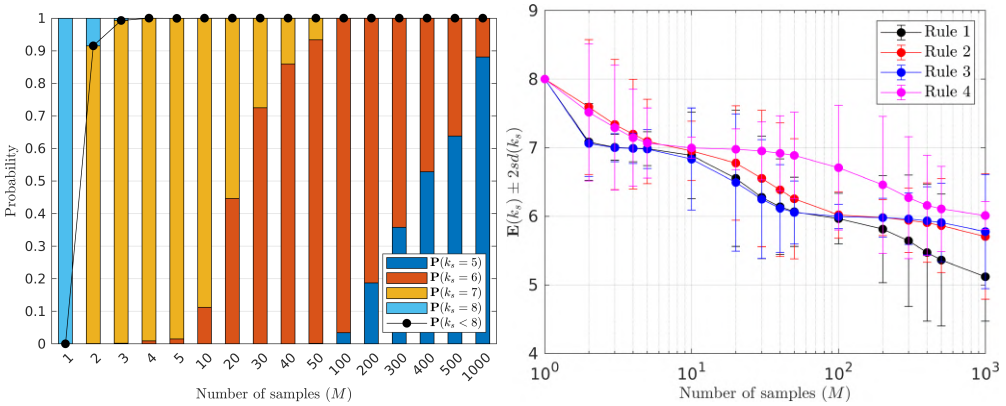
(a) **(b)**

Figure C.1: (a) Analytical solution u_1 of (C.1) plotted against the serial $\mathcal{F}_{\Delta T}$ solution and a single realisation of SParareal. Inset: numerical errors of $\mathcal{F}_{\Delta T}$, SParareal, and Parareal compared to u_1. (b) Errors at successive iterations of Parareal (red) and ten independent realisations of SParareal (blue). The dashed black line represents the tolerance $\varepsilon = 10^{-10}$. Both panels use SParareal with sampling rule 3 and $M = 100$.

with initial value $u_1(0) = 2$ on $t \in [0, 10]$. We discretise using $N = 20$ time slices, $N_{\mathcal{G}} = 20$ and $N_{\mathcal{F}} = 2000$ time steps. This equation (C.1) permits the analytical solution $u_1(t) = (1+t)^2/(t^5/5 + t^4/2 + t^3/3 + 1/2)$, tending to zero as $t \to \infty$. Observe the spacing between equidistant time steps of the true $\mathcal{F}_{\Delta T}$ solution and the SParareal solution to (C.1) in Figure C.1(a), highlighting the stiffness of the solution at early times. Given the stopping tolerance $\varepsilon = 10^{-10}$, we can see in Figure C.1(b) how Parareal converges in $k = 8$ iterations deterministically while SParareal converges in $k_s = 6$ iterations for each of the ten independent realisations shown.

Figure C.2(a) shows the estimated distributions of k_s using sampling rule 1. As expected, if $M = 1$, convergence is deterministic (i.e. Parareal is equivalent to SParareal) and hence $\mathbb{P}(k_s = 8) = 1$. As M increases however, $\mathbb{P}(k_s = 8)$ decreases rapidly to zero whilst $\mathbb{P}(k_s < 8)$ increases from zero to one with just $M \approx 5$ samples. This demonstrates that SParareal requires very few samples to begin converging in fewer iterations than Parareal, and that k_s assumes low values with increasing probabilities for increasing M. The stiffness of (C.1) appears to demand much larger values of M to continually reduce k_s when compared to the non-stiff scalar example in Section 3.3.1.

In Figure C.2(b) we report the expected values of k_s for each sampling rule. This shows that for low values of M, any of the sampling rules can be used to 'beat' Parareal. For larger M, however, rules 1 and 3 (centred around the fine solutions) outperform rules 2 and 4 (which are centred around the PC solution). Further testing revealed that varying the fine time steps within SParareal had little impact on



(a) **(b)**

Figure C.2: (a) Estimated discrete distributions of k_s as a function of M for sampling rule 1. (b) Estimated expectation of k_s as a function of M, calculated using estimated distributions of k_s for each sampling rule with error bars representing $\pm$ two standard deviations $sd(k_s)$. Distributions were estimated by simulating 2000 independent realisations of SParareal for each M.

performance. On the contrary, Figure C.3 shows how increasing the number of coarse steps from 20 to 60 drastically decreases the probability of SParareal converging sooner than Parareal. Increasing the number of coarse steps increases the accuracy of the $\mathcal{G}_{\Delta T}$ solver, hence Parareal reaches the stopping tolerance in fewer iterations k. This result suggests that whilst SParareal can still converge faster than Parareal by using more samples, it works more efficiently for particular problems where k is relatively large, i.e. problems in which a coarser $\mathcal{G}_{\Delta T}$ is used.

Finally, we calculated errors between the mean SParareal solution and the $\mathcal{F}_{\Delta T}$

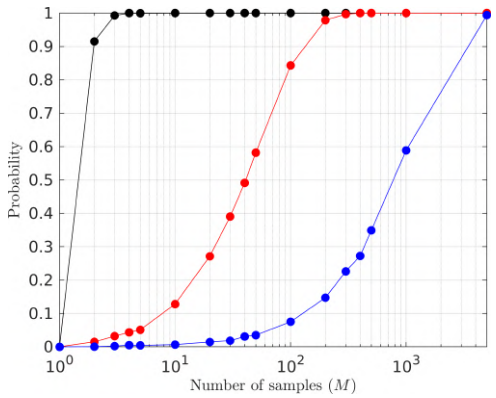


Figure C.3: Estimated probabilities that k_s is smaller than k as a function of M using sampling rule 3. Each curve shows $\mathbb{P}(k_s < 8)$ (black), $\mathbb{P}(k_s < 5)$ (red), and $\mathbb{P}(k_s < 4)$ (blue) using coarse steps $N_{\mathcal{G}} = 20, 40, 60$ respectively—noting that Parareal converges in $k = 8, 5, 4$ iterations for these coarse steps respectively. As before, 2000 independent realisations of SParareal were run for each M.

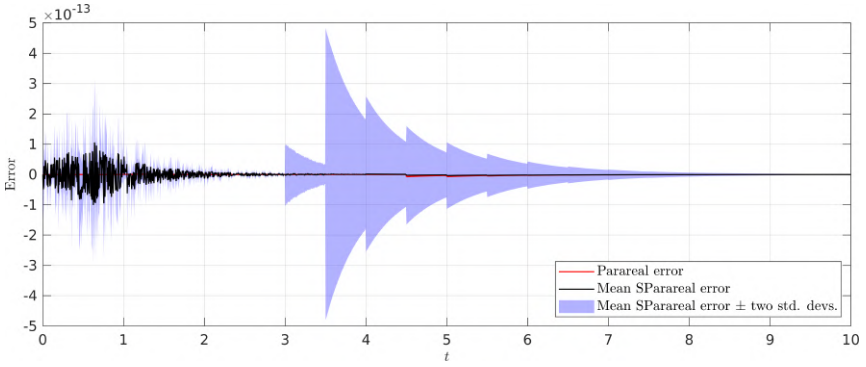


Figure C.4: Errors of Parareal (red) and mean SParareal (black) solutions compared to the $\mathcal{F}_{\Delta T}$ solution. The mean error is obtained by running 2000 independent realisations of SParareal with sampling rule 1 with $M = 10$—the confidence interval representing the mean ± two standard deviations is shown in light blue.

solution (Figure C.4). As expected, we observe that the mean solution attains comparable accuracy to Parareal and the fine solution.

C.2 Square limit cycle system

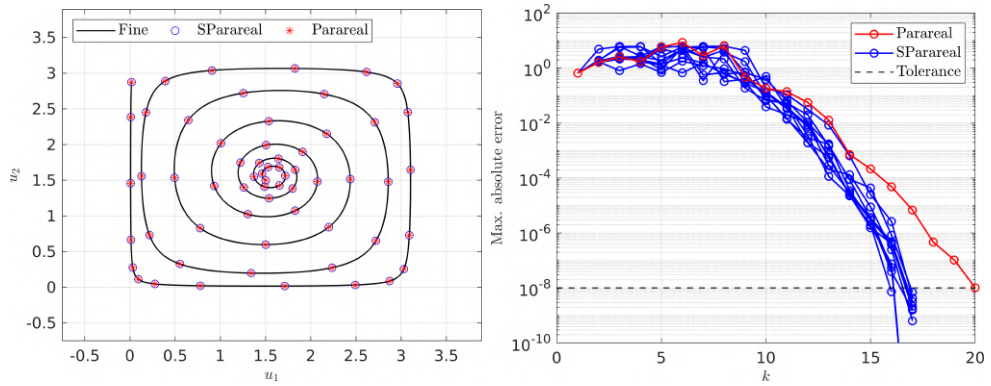
Consider the system

$$\frac{\mathrm{d}u_1}{\mathrm{d}t} = -\sin(u_1)\Big(\frac{\cos(u_1)}{10} + \cos(u_2)\Big), \tag{C.2a}$$

$$\frac{\mathrm{d}u_2}{\mathrm{d}t} = -\sin(u_2)\Big(\frac{\cos(u_2)}{10} - \cos(u_1)\Big), \tag{C.2b}$$

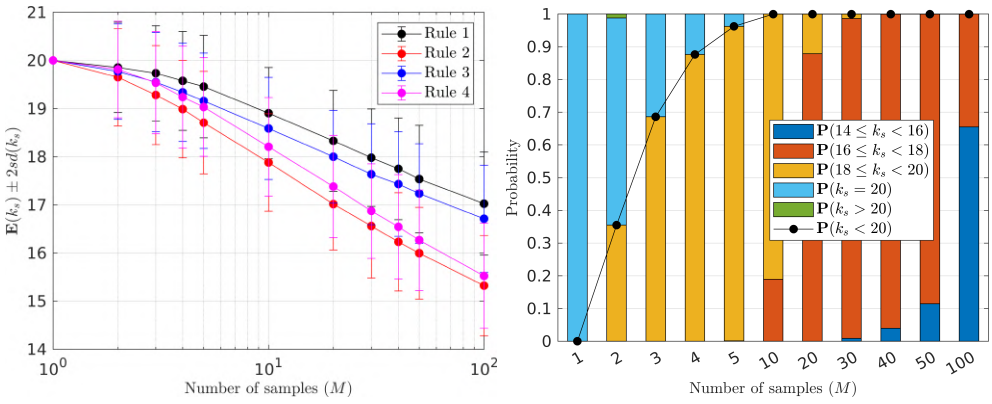
whose solutions, see Figure C.5(a), for initial values within the box $[0, \pi] \times [0, \pi]$, converge toward a square-shaped limit cycle on the edges of the box (Hirsch et al., 2013). The system is solved over $t \in [0, 60]$, starting at $\mathbf{u}(0) = (3/2, 3/2)^\intercal$, using $N = 30$ time slices and $N_{\mathcal{G}} = 30$ and $N_{\mathcal{F}} = 3000$ time steps. As shown in Figure C.5(b), Parareal takes $k = 20$ iterations to converge with tolerance $\varepsilon = 10^{-8}$ whereas ten realisations of SParareal take between $17 \leqslant k_s \leqslant 19$.

Contrary to the Brusselator system, Figure C.6(a) shows that sampling close to the PC values (rules 2 and 4) yield lower expected values of k_s. In this case, the bivariate Gaussian outperforms the t-copula, however the reverse is true for rules 1 and 3. Results generated using uncorrelated samples were found to yield inferior performance (not shown here). The detailed distributions of k_s in Figure C.6(b), using sampling rule 2, show a best performance of $k_s = 14$ with 100 samples and $\mathbb{P}(k_s < 20) = 1$ for $M \approx 10$. They do, however, reveal that in a limited number of cases (using two samples) convergence occurs in $k_s = 21$ iterations. This suggests there may be a minimum number of samples required to beat the convergence rate of Parareal for some systems—something to be mindful of in future experiments.



(a) **(b)**

Figure C.5: (a) Numerical solution of (C.2) over $[0, 60]$ using $\mathcal{F}_{\Delta T}$ (black), Parareal (red), and SParareal (blue). Note that only a subset of the solutions at times $\boldsymbol{t}$ are shown for clarity. (b) Errors at successive iterations of Parareal (red line) and ten independent realisations of SParareal (blue lines). Dashed black line represents the tolerance $\varepsilon = 10^{-8}$. Note that both panels use SParareal with sampling rule 2 and $M = 20$.



(a) **(b)**

Figure C.6: (a) Estimated expectation of k_s as a function of M, calculated using estimated distributions of k_s for each sampling rule with error bars representing $\pm$ two standard deviations $sd(k_s)$. (b) Estimated discrete distributions of k_s as a function of M for sampling rule 2. Distributions were calculated by simulating 2000 independent realisations of SParareal for each M.

D Technical results for SParareal error bounds

D.1 Standard results

Here we state some results that we make repeated use of.

Lemma D.1 (Peter-Paul Inequality)**.** *For any $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{R}^d$ and $\delta > 0$, we have that*

$$2\|\boldsymbol{u}\|\|\boldsymbol{v}\| \leqslant \delta\|\boldsymbol{u}\|^2 + \delta^{-1}\|\boldsymbol{v}\|^2. \tag{D.1}$$

Theorem D.2 (Binomial Theorem)**.** *For $|x| < 1$ and some $m \in \mathbb{N}$, we have that*

$$\frac{1}{(1-x)^m} = \sum_{i=0}^{\infty} \binom{i+m-1}{i} x^i. \tag{D.2}$$

D.2 Generating function method

Here, we solve two recurrence relations using generating functions. These two variable ("double") recurrences crop up often in convergence analysis involving Parareal (and other PinT) algorithms and have been used in a number of settings—refer to Gander and Hairer (2008), Carrel et al. (2022), and Gander et al. (2022) for examples.

Lemma D.3. *Let e_n^k be a non-negative sequence and $A, B, D, \Lambda \in \mathbb{R}$ be non-negative constants. If e_n^k satisfies*

$$e_{n+1}^{k+1} \leqslant Ae_n^k + Be_n^{k+1} + \Lambda, \quad e_{n+1}^1 \leqslant D + Be_n^1, \tag{D.3}$$

for $2 \leqslant k < n \leqslant N$ and $e_0^k = 0\ \forall k \geqslant 0$, then

$$e_n^k \leqslant DA^{k-1} \sum_{\ell=0}^{n-k} \binom{\ell+k-1}{\ell} B^\ell + \Lambda \sum_{j=0}^{k-2} \sum_{\ell=0}^{n-(j+1)} \binom{\ell+j}{\ell} A^j B^\ell.$$

Proof. For $k \geqslant 1$, define the generating function for e_n^k as

$$g_k(x) = \sum_{n=1}^{\infty} e_n^k x^n. \tag{D.4}$$

Multiply (D.3) by x^{n+1} and sum from n equals zero to infinity to obtain

$$\begin{aligned}
\sum_{n=0}^{\infty} e_{n+1}^{k+1} x^{n+1} &\leqslant A\sum_{n=0}^{\infty} e_n^k x^{n+1} + B\sum_{n=0}^{\infty} e_n^{k+1} x^{n+1} + \Lambda\sum_{n=0}^{\infty} x^{n+1}, \\
\sum_{n=0}^{\infty} e_{n+1}^{1} x^{n+1} &\leqslant D\sum_{n=0}^{\infty} x^{n+1} + B\sum_{n=0}^{\infty} e_n^{1} x^{n+1}.
\end{aligned}$$

Using (D.4), binomial theorem (D.2), and recalling that $e_0^k = 0\ \forall k \geqslant 0$, we can write

these expressions as

$$g_{k+1}(x) \leqslant \frac{Ax}{1-Bx} g_k(x) + \frac{\Lambda x}{(1-x)(1-Bx)}, \quad g_1(x) \leqslant \frac{Dx}{(1-x)(1-Bx)},$$

for $|x| < 1$, which can be solved iteratively to give

$$g_k(x) \leqslant \Big(\frac{Ax}{1-Bx}\Big)^{k-1} \frac{Dx}{(1-x)(1-Bx)} + \frac{\Lambda x}{(1-x)(1-Bx)} \sum_{j=0}^{k-2} \Big(\frac{Ax}{1-Bx}\Big)^j.$$

Re-arranging terms, this can be written as

$$g_k(x) \leqslant DA^{k-1}x^k \Big(\frac{1}{1-Bx}\Big)^k \frac{1}{(1-x)} + \Lambda \sum_{j=0}^{k-2} A^j x^{j+1} \Big(\frac{1}{1-Bx}\Big)^{j+1} \frac{1}{1-x}.$$

The first term can be expressed as

$$\begin{aligned} DA^{k-1}x^k \Big(\frac{1}{1-Bx}\Big)^k \frac{1}{(1-x)} &= DA^{k-1}x^k \left(\sum_{i=0}^{\infty} \binom{i+k-1}{i} (Bx)^i \right) \left(\sum_{i=0}^{\infty} x^i \right) \\ &= DA^{k-1}x^k \sum_{m=0}^{\infty} \left(\sum_{\ell=0}^{m} \binom{\ell+k-1}{\ell} B^\ell \right) x^m \\ &= DA^{k-1} \sum_{n=k}^{\infty} \left(\sum_{\ell=0}^{n-k} \binom{\ell+k-1}{\ell} B^\ell \right) x^n. \end{aligned}$$

The first line follows by applying (D.2) twice, the second line using the Cauchy product, and the third by setting $n = m + k$. The second term can be expressed as

$$\begin{aligned} \Lambda \sum_{j=0}^{k-2} A^j x^{j+1} \Big(\frac{1}{1-Bx}\Big)^{j+1} \frac{1}{1-x} &= \Lambda \sum_{j=0}^{k-2} A^j x^{j+1} \left(\sum_{i=0}^{\infty} \binom{i+j}{i} (Bx)^i \right) \left(\sum_{i=0}^{\infty} x^i \right) \\ &= \Lambda \sum_{j=0}^{k-2} A^j x^{j+1} \sum_{m=0}^{\infty} \left(\sum_{\ell=0}^{m} \binom{\ell+j}{\ell} B^\ell \right) x^m \\ &= \Lambda \sum_{n=j+1}^{\infty} \left(\sum_{j=0}^{k-2} \sum_{\ell=0}^{n-(j+1)} \binom{\ell+j}{\ell} A^j B^\ell \right) x^n. \end{aligned}$$

These steps follows as they did for the first term, except that we now set $n = m+j+1$ instead of $n = m + k$ in the final step. Combining these expressions we get

$$\begin{aligned} g_k(x) = \sum_{n=1}^{\infty} e_n^k x^n \leqslant DA^{k-1} \sum_{n=k}^{\infty} \left(\sum_{\ell=0}^{n-k} \binom{\ell+k-1}{\ell} B^\ell \right) x^n \\ + \Lambda \sum_{n=j+1}^{\infty} \left(\sum_{j=0}^{k-2} \sum_{\ell=0}^{n-(j+1)} \binom{\ell+j}{\ell} A^j B^\ell \right) x^n. \end{aligned}$$

By equating the coefficients in x^n on both sides we obtain the bound. □

The initial condition (D.3) can be written differently depending on the available information, i.e. one could instead use $e^1_{n+1} \leqslant D := \hat{e}^1$, slightly altering (D.3).

Lemma D.4. *Let $\hat{e}^k$ be a non-negative sequence and $\tilde{A}, \tilde{B} \in \mathbb{R}$ be non-negative constants. If $\hat{e}^k$ satisfies*

$$\hat{e}^{k+1} \leqslant \tilde{A}\hat{e}^k + \tilde{B}\hat{e}^{k-1}, \tag{D.5}$$

with initial conditions $\hat{e}^0$ and $\hat{e}^1$, then

$$\hat{e}^k \leqslant \hat{e}^0 \left[\frac{\tilde{A} + \sqrt{\tilde{A}^2 + 4\tilde{B}}}{2}\right]^k.$$

Proof. Define the following generating function for $\hat{e}^k$:

$$g(x) = \sum_{k=0}^{\infty} \hat{e}^k x^k.$$

Multiply (D.5) by x^{k+1} and sum from k equals one to infinity to obtain

$$\sum_{k=1}^{\infty} \hat{e}^{k+1}x^{k+1} \leqslant \tilde{A}\sum_{k=1}^{\infty} \hat{e}^k x^{k+1} + \tilde{B}\sum_{k=1}^{\infty} \hat{e}^{k-1}x^{k+1}.$$

Shifting indices, rearranging, and using the initial conditions we get

$$g(x) = \sum_{k=0}^{\infty} \hat{e}^k x^k \leqslant \frac{\hat{e}^0(1 - \tilde{A}x) + \hat{e}^1 x}{1 - \tilde{A}x - \tilde{B}x^2}.$$

Expanding the right hand side in powers of x^k, the coefficients give us

$$\hat{e}^k \leqslant \frac{1}{2\sqrt{\tilde{A}^2 + 4\tilde{B}}}\Big[(\tilde{A}\hat{e}^0 + \hat{e}^0\sqrt{\tilde{A}^2 + 4\tilde{B}} - 2\hat{e}^1)\lambda_1^k + \\ (-\tilde{A}\hat{e}^0 + \hat{e}^0\sqrt{\tilde{A}^2 + 4\tilde{B}} + 2\hat{e}^1)\lambda_2^k\Big],$$

where

$$\lambda_{1,2} = \frac{\tilde{A} \pm \sqrt{\tilde{A}^2 + 4\tilde{B}}}{2}.$$

Without loss of generality, we use that $\lambda_1 \geqslant \lambda_2$ to simplify the bound and obtain

$$\hat{e}^k \leqslant \hat{e}^0\lambda_1^k,$$

which yields the desired result. □

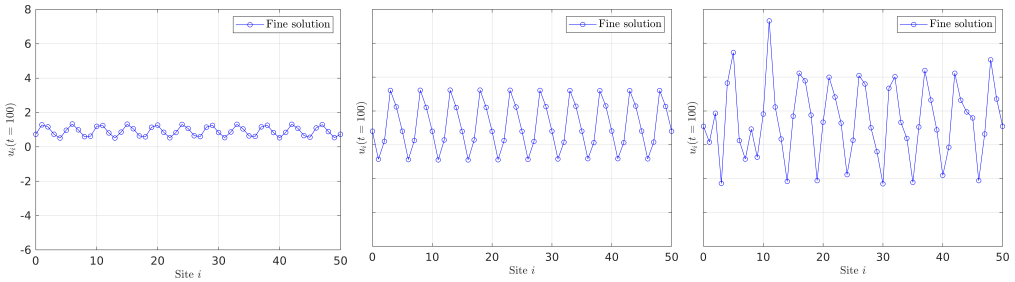
E The Lorenz96 system

The Lorenz96 system was proposed to study the predictability of a "toy" one-dimensional atmospheric model (Lorenz, 1995)[2]. It is a system of d ODEs that model the advection, dissipation, and external forcing of some scalar atmospheric quantity $u_i(t)$ across a periodic array of "sites", $i = 0, \ldots, d-1$ ($d \geqslant 4$), over time. Lorenz suggested thinking of these sites as being spatial locations on a line of latitude around the Earth. The periodicity means that the labelling of sites can be extended for $j \in \mathbb{Z}$ so that $u_j(t) = u_{d+j}(t)$. The non-dimensionalised ODEs are given by

$$\frac{\mathrm{d}u_i}{\mathrm{d}t} = u_{i-1}(u_{i+1} - u_{i-2}) - u_i + F \quad \text{over} \quad t \in [t_0, T], \quad \text{with} \quad u_i(t_0) = u_i^0. \tag{E.1}$$

The free parameter F is the forcing term, governing how difficult the system is to solve—higher F means more chaotic dynamics. Extensive bifurcation analysis of this forcing term F when fixed, or site-dependent, has been carried out by Kerin and Engler (2020).

In Figure E.1, we solve the Lorenz96 system (E.1) for $t \in [0, 100]$ and fixed initial conditions $u_i(0) \in [0, 1]$, $i = 0, \ldots, 49$ ($d = 50$). We do this for three different levels of forcing, $F \in \{1, 3, 5\}$, plotting the solution at each site at the final time $t = 100$. Over time, the solutions are roughly periodic (clearer when one sees the animated solutions[3]) for small F, however, chaos ensues when $F = 5$ and above. To examine the performance of GParareal + fallback in Section 5.4, we will solve system (E.1) for these three levels of forcing.



(a) $F = 1$ **(b)** $F = 3$ **(c)** $F = 5$

Figure E.1: Numerical solutions to the Lorenz96 system (E.1) with $d = 50$ sites and different levels of forcing: (a) $F = 1$, (b) $F = 3$, and (c) $F = 5$. Solutions are plotted at $t = 100$ and were obtained sequentially using the fine solver $\mathcal{F}_{\Delta T} = \text{RK8}$ with $N_{\mathcal{F}} = 10^5$ time steps.

[2]Edward Lorenz presented this model at an ECMWF workshop on predictability in Shinfield Park, Reading in 1995. The paper was not made public until sometime in 1996 and therefore adopted the name "Lorenz96".

[3]To see animated solutions of the Lorenz96 model over time, visit the public code repository.

Bibliography

A. Abdulle and G. Garegnani. Random time step probabilistic methods for uncertainty quantification in chaotic and geometric numerical integration. *Statistics and Computing*, 30(4):907–932, 2020. doi:10.1007/s11222-020-09926-w.

W. Agboh, O. Grainger, D. Ruprecht, and M. Dogar. Parareal with a learned coarse model for robotic manipulation. *Computing and Visualization in Science*, 23(1):8, 2020. doi:10.1007/s00791-020-00327-0.

K. Ait-Ameur, Y. Maday, and M. Tajchman. Multi-step Variant of the Parareal Algorithm. In *Domain Decomposition Methods in Science and Engineering XXV*, Lecture Notes in Computational Science and Engineering, pages 393–400. Springer International Publishing, 2020. doi:10.1007/978-3-030-56750-7_45.

K. Ait-Ameur, Y. Maday, and M. Tajchman. Time-Parallel Algorithm for Two Phase Flows Simulation. In *Numerical Simulation in Physics and Engineering: Trends and Applications: Lecture Notes of the XVIII 'Jacques-Louis Lions' Spanish-French School*, SEMA SIMAI Springer Series, pages 169–178. Springer International Publishing, 2021. doi:10.1007/978-3-030-62543-6_5.

M. A. Álvarez, L. Rosasco, and N. D. Lawrence. Kernels for vector-valued functions: A review. *Foundations and Trends in Machine Learning*, 4:195–266, 2011. doi:10.1561/2200000036.

R. F. Arenstorf. Periodic solutions of the restricted three body problem representing analytic continuations of keplerian elliptic motions. *American Journal of Mathematics*, 85(1):27–35, 1963. doi:10.2307/2373181.

M. Arioli, B. Codenotti, and C. Fassino. The padé method for computing the matrix exponential. *Linear Algebra and its Applications*, 240:111–130, 1996. doi:10.1016/0024-3795(94)00190-1.

E. Aubanel. Scheduling of tasks in the parareal algorithm. *Parallel Computing*, 37(3):172–182, 2011. doi:10.1016/j.parco.2010.10.004.

L. Baffico, S. Bernard, Y. Maday, G. Turinici, and G. Zérah. Parallel-in-time molecular-dynamics simulations. *Physical Review E - Statistical Physics, Plasmas, Fluids, and Related Interdisciplinary Topics*, 66:4–4, 2002. doi:10.1103/PhysRevE.66.057701.

G. Bal. On the convergence and the stability of the parareal algorithm to solve partial differential equations. *Lecture Notes in Computational Science and Engineering*, 40: 425–432, 2005. doi:10.1007/3-540-26825-1_43.

G. Bal. Parallelization in time of (stochastic) ordinary differential equations, 2006. Pre-print: www.stat.uchicago.edu/guillaumebal.

G. Bal and Y. Maday. A "Parareal" Time Discretization for Non-Linear PDE's with Application to the Pricing of an American Put. In *Recent Developments in Domain Decomposition Methods*, pages 189–202. Springer, Berlin, Heidelberg, 2002. doi:10.1007/978-3-642-56118-4_12.

G. Bal and Q. Wu. Symplectic Parareal. In *Lecture Notes in Computational Science and Engineering*, volume 60, pages 401–408. Springer, 2008. doi:10.1007/978-3-540-75199-1_51.

A.-M. Baudron, J.-J. Lautard, Y. Maday, and O. Mula. The parareal in time algorithm applied to the kinetic neutron diffusion equation. In *Domain Decomposition Methods in Science and Engineering XXI*, Lecture Notes in Computational Science and Engineering, pages 437–445. Springer International Publishing, 2014a. doi:10.1007/978-3-319-05789-7_41.

A.-M. Baudron, J.-J. Lautard, Y. Maday, M. K. Riahi, and J. Salomon. Parareal in time 3d numerical solver for the LWR benchmark neutron diffusion transient model. *Journal of Computational Physics*, 279:67–79, 2014b. doi:10.1016/j.jcp.2014.08.037.

J. Beck and S. Guillas. Sequential Design with Mutual Information for Computer Experiments (MICE): Emulation of a Tsunami Model. *SIAM/ASA Journal on Uncertainty Quantification*, 4:739–766, 2016. doi:10.1137/140989613.

A. Bellen and M. Zennaro. Parallel algorithms for initial-value problems for difference and differential equations. *Journal of Computational and Applied Mathematics*, 25:341–350, 1989. doi:10.1016/0377-0427(89)90037-X.

R. Bhatt, L. Debreu, and A. Vidard. Introducing time parallelisation within data assimilation using parareal method, 2022. HAL:03540480.

M. Bolten, S. Friedhoff, and J. Hahne. Task graph-based performance analysis of parallel-in-time methods, 2022. SSRN: 10.2139/ssrn.4201056.

N. Bosch, P. Hennig, and F. Tronarp. Calibrated adaptive probabilistic ODE solvers. In *Proceedings of the 24th International Conference on Artificial Intelligence and Statistics*, pages 3466–3474, 2021. URL http://proceedings.mlr.press/v130/bosch21a/bosch21a.pdf.

S. Brzychczy and R. R. Poznanski. *Mathematical Neuroscience*. Academic Press, 2013.

J. C. Butcher. *Numerical Methods for Ordinary Differential Equations*. John Wiley & Sons, Ltd, third edition, 2016.

T. Buvoli and M. L. Minion. Exponential Runge-Kutta parareal for non-diffusive equations,

2023. arXiv:2301.03764.

B. Carrel, M. J. Gander, and B. Vandereycken. Low-rank parareal: a low-rank parallel-in-time integrator, 2022. arXiv:2203.08455.

CCFE. Culham Centre for Fusion Energy, UK Atomic Energy Authority, 2023. URL https://ccfe.ukaea.uk/.

P. Chartier and B. Philippe. A parallel shooting technique for solving dissipative ODE's. *Computing*, 51:209–236, 1993. doi:10.1007/BF02238534.

O. A. Chkrebtii, D. A. Campbell, B. Calderhead, and M. A. Girolami. Bayesian solution uncertainty quantification for differential equations. *Bayesian Analysis*, 11(4):1239–1267, 2016. doi:10.1214/16-BA1017.

A. J. Christlieb, R. D. Haynes, and B. W. Ong. A parallel space-time algorithm. *SIAM Journal on Scientific Computing*, 34(5):C233–C248, 2012. doi:10.1137/110843484.

A. Clarke, C. Davies, D. Ruprecht, and S. Tobias. Parallel-in-time integration of kinematic dynamos. *Journal of Computational Physics: X*, 7, 2020. doi:10.1016/j.jcpx.2020.100057.

J. Cockayne, C. J. Oates, T. J. Sullivan, and M. Girolami. Bayesian probabilistic numerical methods. *SIAM Review*, 61(4):756–789, 2019. doi:10.1137/17M1139357.

P. R. Conrad, M. Girolami, S. Särkkä, A. Stuart, and K. Zygalakis. Statistical analysis of differential equations: introducing probability measures on numerical solutions. *Statistics and Computing*, 27(4):1065–1082, 2017. doi:10.1007/s11222-016-9671-0.

A. Corenflos, N. Chopin, and S. Särkkä. De-sequentialized monte carlo: a parallel-in-time particle smoother. *Journal of Machine Learning Research*, 23(283):1–39, 2022. URL http://jmlr.org/papers/v23/22-0140.html.

J. Cortial and C. Farhat. A time-parallel implicit method for accelerating the solution of non-linear structural dynamics problems. *International Journal for Numerical Methods in Engineering*, 77(4):451–470, 2009. doi:10.1002/nme.2418.

R. Courant, K. Friedrichs, and H. Lewy. Über die partiellen Differenzengleichungen der mathematischen Physik. *Mathematische Annalen*, 100(1):32–74, 1928. doi:10.1007/BF01448839.

N. Cressie. Spatial Prediction and Kriging. In *Statistics for Spatial Data*, chapter 3, pages 105–209. John Wiley & Sons, Ltd., 1993. doi:10.1002/9781119115151.ch3.

S. Cuomo, V. S. Di Cola, F. Giampaolo, G. Rozza, M. Raissi, and F. Piccialli. Scientific machine learning through physics-informed neural networks: Where we are and what's next. *Journal of Scientific Computing*, 92(3):88, 2022. doi:10.1007/s10915-022-01939-z.

X. Dai, C. Le Bris, F. Legoll, and Y. Maday. Symmetric parareal algorithms for Hamiltonian systems. *ESAIM Mathematical Modelling and Numerical Analysis*, 47:717–742, 2013.

doi:10.1051/m2an/2012046.

J. Danby. *Computer Modeling: From Sports to Spaceflight — From Order to Chaos*. Willmann-Bell, 1997.

V. Dolean, P. Jolivet, and F. Nataf. *An Introduction to Domain Decomposition Methods*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 2015. doi:10.1137/1.9781611974065.

B. D. Dudson, M. V. Umansky, X. Q. Xu, P. B. Snyder, and H. R. Wilson. BOUT++: A framework for parallel plasma fluid simulations. *Computer Physics Communications*, 180 (9):1467–1480, 2009. doi:10.1016/j.cpc.2009.03.008.

ECMWF. European Centre for Medium-Range Weather Forecasts, 2023. URL http://https://www.ecmwf.int/.

W. R. Elwasif, S. S. Foley, D. E. Bernholdt, L. A. Berry, D. Samaddar, D. E. Newman, and R. Sanchez. A dependency-driven formulation of parareal: Parallel-in-time solution of PDEs as a many-task application. In *MTAGS'11 - Proceedings of the 2011 ACM International Workshop on Many Task Computing on Grids and Supercomputers, Co-Located with SC'11*, pages 15–24, New York, NY, 2011. ACM Press. doi:10.1145/2132876.2132883.

S. Engblom. Parallel in time simulation of multiscale stochastic chemical kinetics. *Multiscale Modelling and Simulation*, 8:46–68, 2009. doi:10.1137/080733723.

C. Farhat and M. Chandesris. Time-decomposed parallel time-integrators: Theory and feasibility studies for fluid, structure, and fluid-structure applications. *International Journal for Numerical Methods in Engineering*, 58:1397–1434, 2003. doi:10.1002/nme.860.

P. F. Fischer, F. Hecht, and Y. Maday. A parareal in time semi-implicit approximation of the navier-stokes equations. *Lecture Notes in Compututational Science and Engineering*, 40:433–440, 2005. doi:10.1007/3-540-26825-1_44.

R. FitzHugh. Impulses and physiological states in theoretical models of nerve membrane. *Biophysical Journal*, 1:445–466, 1961. doi:10.1016/S0006-3495(61)86902-6.

B. Fornberg. Generation of finite difference formulas on arbitrarily spaced grids. *Mathematics of Computation*, 51(184):699–706, 1988. doi:10.1090/S0025-5718-1988-0935077-0.

M. J. Gander. Analysis of the parareal algorithm applied to hyperbolic problems using characteristics. *Boletin de la Sociedad Espanola de Matemática Aplicada*, 42:21–35, 2008.

M. J. Gander. 50 Years of Time Parallel Time Integration. In *Multiple Shooting and Time Domain Decomposition Methods*, pages 69–113. Springer, 2015. doi:10.1007/978-3-319-23321-5_3.

M. J. Gander and S. Güttel. Paraexp: A parallel integrator for linear initial-value problems. *SIAM Journal on Scientific Computing*, 35(2), 2013. doi:10.1137/110856137.

M. J. Gander and E. Hairer. Nonlinear convergence analysis for the parareal algorithm. In *Lecture Notes in Computational Science and Engineering*, volume 60, pages 45–56. Springer, 2008. doi:10.1007/978-3-540-75199-1_4.

M. J. Gander and E. Hairer. Analysis for parareal algorithms applied to hamiltonian differential equations. *Journal of Computational and Applied Mathematics*, 259:2–13, 2014. doi:10.1016/j.cam.2013.01.011.

M. J. Gander and M. Petcu. Analysis of a krylov subspace enhanced parareal algorithm for linear problems. *ESAIM: Proceedings*, 25:114–129, 2008. doi:10.1051/proc:082508.

M. J. Gander and S. Vandewalle. Analysis of the parareal time-parallel time-integration method. *SIAM Journal on Scientific Computing*, 29:556–578, 2007. doi:10.1137/05064607X.

M. J. Gander, Y.-L. Jiang, and R.-J. Li. Parareal schwarz waveform relaxation methods. In *Domain Decomposition Methods in Science and Engineering XX*, Lecture Notes in Computational Science and Engineering, pages 451–458. Springer, 2013a. doi:10.1007/978-3-642-35275-1_53.

M. J. Gander, Y. L. Jiang, B. Song, and H. Zhang. Analysis of two parareal algorithms for time-periodic problems. *SIAM Journal on Scientific Computing*, 35(5), 2013b. doi:10.1137/130909172.

M. J. Gander, F. Kwok, and J. Salomon. Paraopt: A parareal algorithm for optimality systems. *SIAM Journal on Scientific Computing*, 42(5):A2773–A28020, 2020. doi:10.1137/19M1292291.

M. J. Gander, T. Lunet, D. Ruprecht, and R. Speck. A unified analysis framework for iterative parallel-in-time algorithms, 2022. arXiv:2203.16069.

I. Garrido, M. S. Espedal, and G. E. Fladmark. A convergent algorithm for time parallelization applied to reservoir simulation. *Lecture Notes in Computational Science and Engineering*, 40:469–476, 2005. doi:10.1007/3-540-26825-1_48.

I. Garrido, B. Lee, G. E. Fladmark, and M. S. Espedal. Convergent iterative schemes for time-parallelization. *Mathematics of Computation*, 75(255):1403–1428, 2006. doi:10.1090/S0025-5718-06-01832-1.

R. J. Goldston. *Introduction to Plasma Physics*. CRC Press, first edition, 1995. doi:10.1201/9780367806958.

S. Götschel, M. Minion, D. Ruprecht, and R. Speck. Twelve ways to fool the masses when giving parallel-in-time results, 2021. arXiv:2102.11670.

T. Grafke, T. Schäfer, and E. Vanden-Eijnden. Long term effects of small random perturbations on dynamical systems: Theoretical and computational tools. In *Recent Progress and Modern Challenges in Applied Mathematics, Modeling and Computational Science*, Fields

Institute Communications, pages 17–55. Springer, New York, NY, 2017. doi:10.1007/978-1-4939-6969-2.

A. Gration and M. I. Wilkinson. Dynamical modelling of dwarf spheroidal galaxies using Gaussian-process emulation. *Monthly Notices of the Royal Astronomical Society*, 485: 4878–4892, 2019. doi:10.1093/mnras/stz605.

L. Grigori, S. A. Hirstoaga, V.-T. Nguyen, and J. Salomon. Reduced model-based parareal simulations of oscillatory singularly perturbed ordinary differential equations. *Journal of Computational Physics*, 436:110282, 2021. doi:10.1016/j.jcp.2021.110282.

E. Hairer, S. P. Nørsett, and G. Wanner. *Solving Ordinary Differential Equations I: Nonstiff Problems*. Springer Series in Computational Mathematics. Springer-Verlag, second edition, 1993. doi:10.1007/978-3-540-78862-1.

T. Haut and B. Wingate. An asymptotic parallel-in-time method for highly oscillatory PDEs. *SIAM Journal on Scientific Computing*, 36(2):A693–A713, 2014. doi:10.1137/130914577.

R. D. Haynes and B. W. Ong. MPI-OpenMP algorithms for the parallel space-time solution of time dependent PDEs. In *Domain Decomposition Methods in Science and Engineering XXI*, Lecture Notes in Computational Science and Engineering, pages 179–187. Springer International Publishing, 2014. doi:10.1007/978-3-319-05789-7_14.

P. Hennig and S. Hauberg. Probabilistic Solutions to Differential Equations and their Application to Riemannian Statistics. In *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics*, volume 33 of *Proceedings of Machine Learning Research*, pages 347–355, Reykjavik, Iceland, 22–25 Apr 2014. URL https://proceedings.mlr.press/v33/hennig14.html.

P. Hennig, M. A. Osborne, and M. Girolami. Probabilistic numerics and uncertainty in computations. *Proceedings of the Royal Society Mathematical, Physical and Engineering Sciences*, 471:20150142, 2015. doi:10.1098/rspa.2015.0142.

P. Hennig, M. A. Osborne, and H. P. Kersting. *Probabilistic Numerics: Computation as Machine Learning*. Cambridge University Press, 2022. doi:10.1017/9781316681411.

N. J. Higham. The scaling and squaring method for the matrix exponential revisited. *SIAM Journal on Matrix Analysis and Applications*, 26(4):1179–1193, 2005. doi:10.1137/04061101X.

M. W. Hirsch, S. Smale, and R. L. Devaney. *Differential Equations, Dynamical Systems, and an Introduction to Chaos*. Academic Press, third edition, 2013.

A. Q. Ibrahim, S. Götschel, and D. Ruprecht. Parareal with a physics-informed neural network as coarse propagator, 2023. arXiv:2303.03848.

M. Iizuka and K. Ono. Influence of the phase accuracy of the coarse solver calculation on

the convergence of the parareal method iteration for hyperbolic PDEs. *Computing and Visualization in Science*, 19(3):97–108, 2018. doi:10.1007/s00791-018-0299-9.

M. Kanagawa, P. Hennig, D. Sejdinovic, and B. K. Sriperumbudur. Gaussian processes and kernel methods: A review on connections and equivalences, 2018. arXiv:1807.02582.

T. Karvonen. Asymptotic bounds for smoothness parameter estimates in Gaussian process interpolation, 2022. arXiv:2203.05400.

T. Karvonen and C. J. Oates. Maximum likelihood estimation in Gaussian process regression is ill-posed, 2022. arXiv:2203.09179.

J. Kerin and H. Engler. On the Lorenz '96 model and some generalizations, 2020. arXiv:2005.07767.

H. Kersting and P. Hennig. Active uncertainty calibration in Bayesian ODE solvers. In *Proceedings of the Thirty-Second Conference on Uncertainty in Artificial Intelligence*, pages 309–318, 2016. doi:10.5555/3020948.3020981.

H. Kersting, T. J. Sullivan, and P. Hennig. Convergence rates of Gaussian ODE filters. *Statistics and Computing*, 30(6):1791–1816, 2020. doi:10.1007/s11222-020-09972-4.

P. E. Kloeden and E. Platen. *Numerical Solution of Stochastic Differential Equations*. Springer, 1992. doi:10.1007/978-3-662-12616-5.

N. Krämer, N. Bosch, J. Schmidt, and P. Hennig. Probabilistic ODE solutions in millions of dimensions. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 11634–11649, 2022. https://proceedings.mlr.press/v162/kramer22b/kramer22b.pdf.

W. Kutta. Beitrag zur näherungsweisen integration totaler differentialgleichungen. *Zeitschrift für Mathematik und Physik*, 46:435–453, 1901.

C.-O. Lee, Y. Lee, and J. Park. Parareal neural networks emulating a parallel-in-time algorithm. *IEEE Transactions on Neural Networks and Learning Systems*, 33(11):6089–6102, 2022. doi:10.1109/TNNLS.2021.3072209.

R. Lefever and G. Nicolis. Chemical instabilities and sustained oscillations. *Journal of Theoretical Biology*, 30:267–284, 1971. doi:10.1016/0022-5193(71)90054-3.

F. Legoll, T. Lelièvre, K. Myerscough, and G. Samaey. Parareal computation of stochastic differential equations with time-scale separation: A numerical convergence study. *Compututing and Visualisation in Science*, 23:1–18, 2020. doi:10.1007/s00791-020-00329-y.

F. Legoll, T. Lelièvre, and U. Sharma. An adaptive parareal algorithm: Application to the simulation of molecular dynamics trajectories. *SIAM Journal on Scientific Computing*, 44 (1):B146–B176, 2022. doi:10.1137/21M1412979.

H. C. Lie, A. M. Stuart, and T. J. Sullivan. Strong convergence rates of probabilistic integrators for ordinary differential equations. *Statistics and Computing*, 29(6):1265–1283, 2019. doi:10.1007/s11222-019-09898-6.

H. C. Lie, M. Stahn, and T. J. Sullivan. Randomised one-step time integration methods for deterministic operator differential equations. *Calcolo*, 59(1):13, 2022. doi:10.1007/s10092-022-00457-6.

J. L. Lions, Y. Maday, and G. Turinici. Résolution d'EDP par un schéma en temps ≪pararéel≫. *Comptes Rendus de l'Académie des Sciences - Series I - Mathematics*, 332 (7):661–668, 2001. doi:10.1016/S0764-4442(00)01793-6.

X. Liu and S. Guillas. Dimension reduction for gaussian process emulation: An application to the influence of bathymetry on tsunami heights. *SIAM/ASA Journal on Uncertainty Quantification*, 5(1):787–812, 2017. doi:10.1137/16M1090648.

E. Lorenz. Deterministic Nonperiodic Flow. *Journal of the Atmospheric Sciences*, 20:130–141, 1963. doi:10.1175/1520-0469(1963)020¡0130:dnf¿2.0.co;2.

E. Lorenz. Predictability: a problem partly solved. In *Seminar on Predictability, 4-8 September 1995*, volume 1, pages 1–18. ECMWF, 1995. URL https://www.ecmwf.int/node/10829.

Y. Maday and O. Mula. An adaptive parareal algorithm. *Journal of Computational and Applied Mathematics*, 377:112915–112915, 2020. doi:10.1016/j.cam.2020.112915.

Y. Maday and G. Turinici. A parareal in time procedure for the control of partial differential equations. *Comptes Rendus de l'Académie des Sciences Paris*, 335:387–392, 2002. doi:10.1016/S1631-073X(02)02467-6.

A. Mann. Core Concept: Nascent exascale supercomputers offer promise, present challenges. *Proceedings of the National Academy of Sciences*, 117:22623–22625, 2020. doi:10.1073/pnas.2015968117.

M. D. McKay, R. J. Beckman, and W. J. Conover. A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics*, 21(2):239–245, 1979. doi:10.2307/1268522.

X. Meng, Z. Li, D. Zhang, and G. E. Karniadakis. PPINN: Parareal physics-informed neural network for time-dependent PDEs. *Computer Methods in Applied Mechanics and Engineering*, 370:113250, 2020. doi:10.1016/j.cma.2020.113250.

G. E. Moore. Cramming more components onto integrated circuits. *Electronics*, 38(8):1–4, 1965.

K. P. Murphy. *Probabilistic Machine Learning: An Introduction*. MIT Press, 2022.

K. P. Murphy. *Probabilistic Machine Learning: Advanced Topics*. MIT Press, 2023.

J. D. Murray. *Mathematical Biology: I. An Introduction*, volume 17. Springer, 2002. doi:10.1007/b98868.

J. Nagumo, S. Arimoto, and S. Yoshizawa. An active pulse transmission line simulating nerve axon. *Proceedings of the IRE*, 50:2061–2070, 1962. doi:10.1109/JRPROC.1962.288235.

R. B. Nelsen. *An Introduction to Copulas*. Springer New York, 2006.

H. Nguyen and R. Tsai. Numerical wave propagation aided by deep learning, 2022. arXiv:2107.13184.

A. S. Nielsen. Feasibility study of the parareal algorithm, 2012. PhD Thesis, Technical University of Denmark.

J. Nievergelt. Parallel methods for integrating ordinary differential equations. *Communications of the ACM*, 7:731–733, 1964. doi:10.1145/355588.365137.

J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer Series in Operations Research and Financial Engineering. Springer New York, 2006. doi:10.1007/978-0-387-40065-5.

C. J. Oates and T. J. Sullivan. A modern retrospective on probabilistic numerics. *Statistics and Computing*, 29:1335–1351, 2019. doi:10.1007/s11222-019-09902-z.

A. O'Hagan. Curve fitting and optimal design for prediction. *Journal of the Royal Statistical Society: Series B - Methodology*, 40:1–24, 1978. doi:10.1111/j.2517-6161.1978.tb01643.x.

A. O'Hagan. Bayesian analysis of computer code outputs: A tutorial. *Reliability Engineering & System Safety*, 91:1290–1300, 2006. doi:10.1016/j.ress.2005.11.025.

B. Øksendal. *Stochastic Differential Equations: An Introduction with Applications*. Springer Science & Business Media, 2013.

B. W. Ong and J. B. Schroder. Applications of time parallelization. *Computing and Visualisation in Science*, 23, 2020. doi:10.1007/s00791-020-00331-4.

K. Pentland. PhD Thesis Code Repository, 2023. https://livewarwickac-my.sharepoint.com/personal/u1995041_live_warwick_ac_uk/_layouts/15/onedrive.aspx?id=.

K. Pentland, M. Tamborrino, D. Samaddar, and L. C. Appel. Stochastic parareal: An application of probabilistic methods to time-parallelization. *SIAM Journal on Scientific Computing*, 45(3):S82–S102, 2023a. doi:10.1137/21M1414231.

K. Pentland, M. Tamborrino, and T. J. Sullivan. Error bound analysis of the stochastic parareal algorithm. *SIAM Journal on Scientific Computing*, 45(5):A2657–A2678, 2023b. doi:10.1137/22M1533062.

K. Pentland, M. Tamborrino, T. J. Sullivan, J. Buchanan, and L. C. Appel. GParareal: a time-parallel ODE solver using Gaussian process emulation. *Statistics and Computing*, 33

(1):23, 2023c. doi:10.1007/s11222-022-10195-y.

A. Quarteroni and A. Valli. *Domain Decomposition Methods for Partial Differential Equations*. Oxford University Press, 1999.

J. Quiñonero Candela and C. E. Rasmussen. A unifying view of sparse approximate Gaussian process regression. *Journal of Machine Learning Research*, 6(65):1939–1959, 2005.

C. E. Rasmussen. Gaussian Processes in Machine Learning. In *Advanced Lectures on Machine Learning: ML Summer Schools 2003, Canberra, Australia, February 2–14, 2003, Tübingen, Germany, August 4 - 16, 2003, Revised Lectures*, Lecture Notes in Computer Science, pages 63–71. Springer, 2004. doi:10.1007/978-3-540-28650-9_4.

C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning*. Adaptive Computation and Machine Learning. MIT Press, 2006.

J. Rosemeier, T. Haut, and B. Wingate. Multi-level parareal algorithm with averaging, 2022. arXiv:2211.17239.

O. E. Rössler. An equation for continuous chaos. *Physics Letters A*, 57:397–398, 1976. doi:10.1016/0375-9601(76)90101-8.

C. Runge. Ueber die numerische auflösung von differentialgleichungen. *Mathematische Annalen*, 46(2):167–178, 1895. doi:10.1007/BF01446807.

K. Rupp. 50 years of microprocessor data, 2022. Raw data accessed on 25/10/2022 from https://github.com/karlrupp/microprocessor-trend-data.

D. Ruprecht. Convergence of Parareal with spatial coarsening. *Proceedings in Applied Mathematics & Mechanics*, 14:1031–1034, 2014. doi:10.1002/pamm.201410490.

D. Ruprecht. Shared memory pipelined parareal. In *Euro-Par 2017: Parallel Processing*, Lecture Notes in Computer Science, pages 669–681. Springer International Publishing, 2017. doi:10.1007/978-3-319-64203-1_48.

D. Ruprecht. Wave propagation characteristics of parareal. *Computing and Visualization in Science*, 19(1):1–17, 2018. doi:10.1007/s00791-018-0296-z.

D. Ruprecht and R. Krause. Explicit parallel-in-time integration of a linear acoustic-advection system. *Computers & Fluids*, 59:72–83, 2012. doi:10.1016/j.compfluid.2012.02.015.

P. Saha, J. Stadel, and S. Tremaine. A Parallel Integration Method for Solar System Dynamics. *The Astronomical Journal*, 114:409, 1997. doi:10.1086/118485.

D. Samaddar, D. E. Newman, and R. Sánchez. Parallelization in time of numerical simulations of fully-developed plasma turbulence using the parareal algorithm. *Journal of Computational Physics*, 229:6558–6573, 2010. doi:10.1016/j.jcp.2010.05.012.

D. Samaddar, D. P. Coster, X. Bonnin, L. A. Berry, W. R. Elwasif, and D. B. Batchelor. Application of the parareal algorithm to simulations of ELMs in ITER plasma. *Computer Physics Communications*, 235:246–257, 2019. doi:10.1016/j.cpc.2018.08.007.

H. Samuel. Time domain parallelization for computational geodynamics. *Geochemistry, Geophysics, Geosystems*, 13(1), 2012. doi:10.1029/2011GC003905.

S. Särkkä. *Bayesian Filtering and Smoothing*. Institute of Mathematical Statistics Textbooks. Cambridge University Press, 2013. doi:10.1017/CBO9781139344203.

S. Särkkä and A. F. Garciá-Fernández. Temporal parallelization of bayesian smoothers. *IEEE Transactions on Automatic Control*, 66(1):299–306, 2021. doi:10.1109/TAC.2020.2976316.

F. Schäfer, T. J. Sullivan, and H. Owhadi. Compression, inversion, and approximate PCA of dense kernel matrices at near-linear computational complexity. *Multiscale Modelling and Simulation*, 19(2):688–730, 2021. doi:10.1137/19M129526X.

M. Schober, D. K. Duvenaud, and P. Hennig. Probabilistic ODE solvers with Runge-Kutta means. *Advances in Neural Information Processing Systems*, 27, 2014a. URL https://papers.nips.cc/paper/2014/hash/59b90e1005a220e2ebc542eb9d950b1e-Abstract.html.

M. Schober, N. Kasenburg, A. Feragen, P. Hennig, and S. Hauberg. Probabilistic shortest path tractography in DTI using gaussian process ODE solvers. In *Medical Image Computing and Computer-Assisted Intervention (MICCAI 2014)*, Lecture Notes in Computer Science, pages 265–272. Springer International Publishing, 2014b. doi:10.1007/978-3-319-10443-0_34.

M. Schober, S. Särkkä, and P. Hennig. A probabilistic model for the numerical solution of initial value problems. *Statistics and Computing*, 29:99–122, 2019. doi:10.1007/s11222-017-9798-7.

H. A. Schwarz. Über einen grenzübergang durch alternierendes verfahren. *Vierteljahrsschrift der Naturforschenden Gesellschaft in Zürich*, pages 272–286, 1870.

J. Sevilla, L. Heim, A. Ho, T. Besiroglu, M. Hobbhahn, and P. Villalobos. Compute trends across three eras of machine learning, 2022. arXiv:2202.05924.

J. Shalf. The future of computing beyond Moore's law. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 378(2166):20190061, 2020. doi:10.1098/rsta.2019.0061.

J. Skilling. Bayesian solution of ordinary differential equations. In *Maximum Entropy and Bayesian Methods: Seattle, 1991*, Fundamental Theories of Physics, pages 23–37. Springer Netherlands, 1992. doi:10.1007/978-94-017-2219-3_2.

A. Sklar. Fonctions de Répartition à n Dimensions et Leurs Marges. *Publications de L'Institut de Statistique de L'Université de Paris*, 8:229–231, 1959.

E. Snelson and Z. Ghahramani. Sparse Gaussian processes using pseudo-inputs. In *Advances in Neural Information Processing Systems*, volume 18. MIT Press, 2006.

E. Snelson and Z. Ghahramani. Local and global sparse Gaussian process approximations. In *Proceedings of the Eleventh International Conference on Artificial Intelligence and Statistics*, pages 524–531. PMLR, 2007.

B. S. Southworth. Necessary conditions and tight two-level convergence bounds for parareal and multigrid reduction in time. *SIAM Journal on Matrix Analysis and Applications*, 40 (2):564–608, 2019. doi:10.1137/18M1226208.

G. A. Staff and E. M. Rønquist. Stability of the parareal algorithm. *Lecture Notes in Computational Science and Engineering*, 40:449–456, 2005. doi:10.1007/3-540-26825-1_46.

A. M. Stuart and A. L. Teckentrup. Posterior consistency for Gaussian process approximations of bayesian posterior distributions. *Mathematics of Computation*, 87(310):721–753, 2018. doi:10.1090/mcom/3244.

TOP500Project. TOP500 supercomputer performance development, 2022. Raw data accessed on 25/10/2022 from https://www.top500.org/statistics/perfdevel/.

A. Toselli and O. Widlund. *Domain Decomposition Methods — Algorithms and Theory*. Springer New York, 2005.

L. N. Trefethen. *Spectral Methods in MATLAB*. SIAM, 2000. doi:10.1137/1.9780898719598.

L. N. Trefethen, A. Birkisson, and T. Driscoll. *Exploring ODEs*. Society for Industrial and Applied Mathematics, Philadelphia, 2017. doi:10.1137/1.9781611975161.

J. M. F. Trindade and J. C. F. Pereira. Parallel-in-time simulation of two-dimensional, unsteady, incompressible laminar flows. *Numerical Heat Transfer, Part B: Fundamentals*, 50:25–40, 2006. doi:10.1080/10407790500459379.

R. Trobec, B. Slivnik, P. Bulić, and B. Robič. *Introduction to Parallel Computing: From Algorithms to Programming on State-of-the-Art Platforms*. Undergraduate Topics in Computer Science. Springer International Publishing, 2018. doi:10.1007/978-3-319-98833-7_1.

F. Tronarp, H. Kersting, S. Särkkä, and P. Hennig. Probabilistic solutions to ordinary differential equations as nonlinear Bayesian filtering: A new perspective. *Statistics and Computing*, 29:1297–1315, 2019. doi:10.1007/s11222-019-09900-1.

H. Wendland. *Scattered Data Approximation*. Cambridge Monographs on Applied and Computational Mathematics. Cambridge University Press, 2004. doi:10.1017/CBO9780511617539.

J. Wenger, N. Krämer, M. Pförtner, J. Schmidt, N. Bosch, N. Effenberger, J. Zenn, A. Gessner, T. Karvonen, F.-X. Briol, M. Mahsereci, and P. Hennig. ProbNum: Probabilistic numerics in python, 2021. arXiv:2112.02100.

Yale. Yale Centre for Research Computing, 2023. URL https://research.computing.yale.edu/.

G. R. Yalla and B. Engquist. Parallel in time algorithms for multiscale dynamical systems using interpolation and neural networks. In *Proceedings of the High Performance Computing Symposium*, HPC '18, pages 1–12. Society for Computer Simulation International, 2018.