CCFE
CULHAM CENTRE FOR FUSION ENERGY

UK Atomic Energy Authority

UKAEA-CCFE-PR(22)59

K. Pentland, M. Tamborrino, T. J. Sullivan, J. Buchanan, L. C. Appel

GParareal: A time-parallel ODE solver using Gaussian process emulation

GParareal: A time-parallel ODE solver using Gaussian process emulation

K. Pentland, M. Tamborrino, T. J. Sullivan, J. Buchanan, L. C. Appel

This is a preprint of a paper submitted for publication in Statistics and Computing

GParareal: A time-parallel ODE solver using Gaussian process emulation

Kamran Pentland[1] Massimiliano Tamborrino[2] T. J. Sullivan[1,3]
James Buchanan[4] L. C. Appel[4]

January 20, 2022

Abstract. Sequential numerical methods for integrating initial value problems (IVPs) can be prohibitively expensive when high numerical accuracy is required over the entire interval of integration. One remedy is to integrate in a parallel fashion, "predicting" the solution serially using a cheap (coarse) solver and "correcting" these values using an expensive (fine) solver that runs in parallel on a number of temporal subintervals. We propose an adaptive time-parallel algorithm that solves IVPs by modelling the correction term, i.e. the difference between fine and coarse solutions, using a Gaussian process emulator. This approach compares favourably with the classic *parareal* algorithm and has the additional advantage of being able to use archives of legacy solutions, i.e. solutions from prior runs of the IVP for different initial conditions, to further accelerate convergence of the method.

Keywords. Gaussian process emulation • parallel-in-time • parareal • initial value problems
2020 Mathematics Subject Classification. 65L05 • 65Y05 • 60G15

1. Introduction

1.1. Motivation and background

This paper is concerned with the numerical solution of a system of $d \in \mathbb{N}$ ordinary differential equations (ODEs) of the form

$$\frac{\mathrm{d}\boldsymbol{u}}{\mathrm{d}t} = \boldsymbol{f}\big(t, \boldsymbol{u}(t)\big) \quad \text{over} \quad t \in [t_0, T], \quad \text{with} \quad \boldsymbol{u}(t_0) = \boldsymbol{u}_0 \in \mathbb{R}^d, \tag{1.1}$$

where $\boldsymbol{f} \colon \mathbb{R} \times \mathbb{R}^d \to \mathbb{R}$ is a nonlinear function with sufficiently many continuous partial derivatives, $\boldsymbol{u} \colon [t_0, T] \to \mathbb{R}^d$ is the time-dependent solution, and $\boldsymbol{u}_0 \in \mathbb{R}^d$ is the initial value at t_0. We seek numerical solutions $\boldsymbol{U_j} \approx \boldsymbol{u}(t_j)$ to the initial value problem (IVP) in (1.1) on a pre-defined mesh $\boldsymbol{t} = (t_0, \ldots, t_J)$, where $t_{j+1} = t_j + \Delta T$ for fixed $\Delta T = (T - t_0)/J$.

More specifically, we are concerned with IVPs where: (i) the interval of integration, $[t_0, T]$; (ii) the number of mesh points, $J + 1$; or (iii) the wallclock time to evaluate the vector field, $\boldsymbol{f}$, is

[1] Mathematics Institute, University of Warwick, Coventry, CV4 7AL, United Kingdom (kamran.pentland@warwick.ac.uk, t.j.sullivan@warwick.ac.uk)
[2] Department of Statistics, University of Warwick, Coventry, CV4 7AL, United Kingdom (massimiliano.tamborrino@warwick.ac.uk)
[3] Alan Turing Institute, British Library, 96 Euston Road, London NW1 2DB, United Kingdom
[4] Culham Centre for Fusion Energy, Culham Science Centre, Abingdon, Oxfordshire, OX14 3DB, United Kingdom (james.buchanan@ukaea.uk, lynton.appel@ukaea.uk)

so large that the simulation of such numerical solutions takes hours, days, or even weeks using classical sequential integration methods, e.g. implicit/explicit Runge–Kutta methods (Hairer et al., 1993). This issue also arises when solving IVPs with spatial or other non-temporal dependencies in the sense that, even though highly efficient domain decomposition methods exist (Dolean et al., 2015), the parallel speed-up of such methods on high performance computers (HPCs) is still constrained by the serial nature of the time-stepping scheme. Therefore, with the advent of exascale HPCs on the horizon (Mann, 2020), there has been renewed interest in developing more efficient and robust *time-parallel* algorithms to reduce wallclock runtimes for IVP simulations in applications spanning numerical weather prediction (Hamon et al., 2020), kinematic dynamo modelling (Clarke et al., 2020), and plasma physics (Samaddar et al., 2010, 2019) to name but a few. In this work, we focus on the development of such a time-parallel method.

To solve (1.1) in parallel, one must overcome the causality principle of time: solutions at later times depend on solutions at earlier times. In recent years, a growing number of time-parallel algorithms, whereby one partitions $[t_0, T]$ into J 'slices' and attempts to solve J smaller IVPs using J processors, have been developed to speed-up IVP simulations — see Gander (2015) and Ong and Schroder (2020) for comprehensive reviews. We take inspiration from the *parareal* algorithm (Lions et al., 2001), a multiple shooting-type (or multigrid (Gander and Vandewalle, 2007)) method that uses a predictor-corrector update rule based on two numerical integrators, one coarse- and one fine-grained in time, to iteratively locate solutions $\boldsymbol{U}_j^k$ to (1.1) in parallel. At any iteration $k \in \{1, \ldots, J\}$ of parareal, the 'correction' is given by the residual between fine and coarse solutions obtained during iteration $k-1$ (further details are provided in Section 2). In a Markovian-like manner, all fine/coarse information about the solution obtained prior to iteration $k-1$ is ignored by the predictor-corrector rule, 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). Our goal is to demonstrate that such "acquisition" data, i.e. fine and coarse solution information accumulated up to iteration k, can be exploited using a statistical *emulator* in order to simulate a solution in faster wallclock time than parareal. Making use of acquisition data in parareal is mentioned briefly in the appendix of Maday and Mula (2020), in the context of spatial domain decomposition and high-order time-stepping, but has yet be investigated further.

In particular, we use a Gaussian process (GP) emulator (O'Hagan, 1978; Rasmussen, 2004) to rapidly infer the (expensive-to-simulate) multi-fidelity correction term in parareal. The emulator is trained using fine and coarse acquisition data from *all* prior iterations, with data from the fine integrator having been obtained in parallel. Similar to parareal, we derive a predictor-corrector scheme where the coarse integrator makes rapid low-accuracy predictions about the solutions which are subsequently refined using a correction, now inferred from the GP emulator. As well as using the emulator, the difference between this approach and parareal is that the new correction term is formed of integrated solutions values at the current iteration k, rather than $k-1$. Supposing the fine solver is of sufficient accuracy to exactly solve the IVP, the algorithm presented in this paper determines numerical solutions $\boldsymbol{U}_j^k$ that converge (assuming the emulator is sufficiently well trained) toward the exact solutions $\boldsymbol{U}_j$ over a number of refinement iterations k. This new approach is particularly beneficial if one wishes to fully understand and evaluate the dynamics of (1.1) by simulating solutions for a range initial values $\boldsymbol{u}_0$ or with different (bifurcating) input parameters. Firstly, if one can obtain extra parallel speedup, generating such a sequence of independent simulations becomes more computationally tractable in feasible time. Secondly, the "legacy" data, i.e. solution information accumulated between independent runs, can also be used to inform future simulations, avoiding the need to repeat expensive calculations. Being able to re-use (expensive) acquisition or legacy data to integrate IVPs such as (1.1) in parallel is not something, to the best of our knowledge, that existing time-parallel algorithms

currently do.

In recent years there has been a surge in interest in the field of *probabilistic numerics* (Hennig et al., 2015; Oates and Sullivan, 2019), where "ODE filters" have been developed to solve ODEs using GP regression techniques. Instead of simulating a numerical solution on the mesh $\boldsymbol{t}$, as classical integration methods do, ODE filters return a probability measure over the solution at any $t \in [t_0, T]$ (Bosch et al., 2021; Schober et al., 2019; Tronarp et al., 2019). Such methods solve sequentially in time, conditioning the GP on acquisition data, i.e. solution and derivative evaluations, at competitive computational cost (compared to classical methods) (Kersting et al., 2020; Krämer et al., 2021). However, integrating IVPs with large time intervals or expensive vector fields using such filters is still a computationally intractable process. As such, our aim is to fuse aspects of time-parallelism with the Bayesian methods showcased in ODE filters — something briefly mentioned in Kersting and Hennig (2016) and Pentland et al. (2021), but not yet explored. Whereas ODE filters use GPs to explicitly model the *solution* to an IVP, we instead use them to model the *residual* between approximate solutions provided by the deterministic fine and coarse solvers, i.e. the parareal correction. While the method proposed in this paper does not return a probabilistic solution to (1.1), we believe it contributes to a positive step in this direction.

In this paper, we present an adaptive time-parallel algorithm that iteratively locates a numerical solution to an IVP (1.1) using a Gaussian process emulator conditioned on acquisition and legacy data. The aim is that this algorithm makes more efficient use of the simulation data it generates and therefore provide additional parallel speed-up compared to parareal, from which it takes inspiration.

1.2. Contributions and outline

The rest of this paper is outlined as follows. In Section 2 we introduce parareal, providing an overview of the algorithm and its computational complexity. In Section 3 we present our algorithm, henceforth referred to as GParareal, in which we describe how the GP emulator, conditioned on acquisition data obtained in parallel throughout the simulation, is used to refine coarse numerical solutions to an IVP. Numerical experiments are performed using HPC facilities in Section 4. We demonstrate good performance of GParareal against parareal in terms of convergence, wallclock time, and solution accuracy on a number of low-dimensional ODE problems using just acquisition data. Furthermore, we demonstrate how the GP emulator enables convergence in cases where the coarse solver is too inaccurate for parareal to converge and show that GParareal can use legacy simulation data to solve a given system for alternate initial values even faster, retaining comparable numerical accuracy. We discuss the benefits, drawbacks, and open questions surrounding GParareal in Section 5.

2. Parareal

Here we briefly recall the parareal algorithm (Lions et al., 2001), first describing the fine- and coarse-grained numerical solvers it uses, then the algorithm itself, and finally some remarks on complexity, numerical speed-up, and choice of solvers. For a full mathematical derivation and exposition of parareal, refer to Gander and Vandewalle (2007). To simplify notation, we describe parareal for solving a scalar-valued ODE, i.e. $\boldsymbol{f}(t, \boldsymbol{u}(t)) := f(t, u(t))$ in (1.1), without loss of generality.

2.1. The solvers

Denote the parareal approximation of the solution to (1.1) at time t_j as $U_j \approx u(t_j), j = 0, \dots, J$. To calculate these solutions, parareal uses two one-step[1] numerical integrators. The first, referred to as the *fine solver* $\mathcal{F}$, is a computationally expensive integrator that propagates an initial value at t_j, over an interval of length ΔT, and returns a solution with high numerical accuracy at t_{j+1}. In this paper, we assume that $\mathcal{F}$ provides sufficient numerical accuracy to solve (1.1) for the solution to be considered 'exact'. The objective is to calculate the exact solutions

$$U_{j+1} = \mathcal{F}(U_j) \quad \text{for} \quad j = 0, \dots, J-1, \quad \text{where} \quad U_0 = u_0, \tag{2.1}$$

without running $\mathcal{F}$ J times sequentially, as this calculation is assumed to be computationally intractable. To avoid this, parareal locates iteratively improved approximations U_j^k, where $k = 0, 1, 2, \dots$ is the iteration number, that converge toward U_j as k increases; note that $U_0^k = U_0 = u_0$ for all k. To do this, parareal uses a second numerical integrator $\mathcal{G}$, referred to as the *coarse solver*. $\mathcal{G}$ propagates an initial value at t_j over an interval of length ΔT, however, it has lower numerical accuracy and is computationally inexpensive to run compared to $\mathcal{F}$. This means that $\mathcal{G}$ can be run serially across a number of time slices to provide relatively cheap low accuracy solutions whilst $\mathcal{F}$ is permitted only to run in parallel over multiple slices.

2.2. The algorithm

To begin (iteration $k = 0$), approximate solutions to (1.1) are calculated sequentially using $\mathcal{G}$, on a single processor, such that

$$U_{j+1}^0 = \mathcal{G}(U_j^0) \quad j = 0, \dots, J-1. \tag{2.2}$$

Following this (iteration $k = 1$), the fine solver propagates each approximation in (2.2) *in parallel*, on J processors, to obtain $\mathcal{F}(U_j^0)$ for $j = 0, \dots, J-1$. These values are then used in the predictor-corrector

$$U_{j+1}^k = \underbrace{\mathcal{G}(U_j^k)}_{\text{predict}} + \underbrace{\mathcal{F}(U_j^{k-1}) - \mathcal{G}(U_j^{k-1})}_{\text{correct}} \quad \text{for} \quad j = 0, \dots, J-1. \tag{2.3}$$

Here, $\mathcal{G}$ is applied sequentially to predict the solution at the next time step, before being corrected by the residual between coarse and fine values found during the previous iteration (note that (2.3) cannot be run in parallel). This is a discretised approximation of the Newton–Raphson method for locating the true roots U_j with initial guess (2.2) (Gander and Vandewalle, 2007). The final step is to evaluate the stopping criteria

$$|U_j^k - U_j^{k-1}| < \varepsilon \quad \forall j \leqslant I, \tag{2.4}$$

for some pre-defined tolerance ε, stopping once $I = J$. This criterion is standard for parareal (Gander and Hairer, 2008; Garrido et al., 2006), however, other criteria such as taking the average relative error between fine solutions over a time slice (Samaddar et al., 2010, 2019) or measuring the total energy of the system, could be used instead. Unconverged solution values, i.e. U_j^k for $j > I$, are improved in later iterations ($k > 1$) by initiating further parallel $\mathcal{F}$ runs on each U_j^k, followed by an update using (2.3). The version of parareal implemented in Section 4

[1] Multi-step numerical integrators have been tested with parareal (Ait-Ameur et al., 2020, 2021). However, they require multiple initial values to begin integration in each time slice and are not compatible with the proposed method in Section 3.

does not iterate over solutions that have already converged, avoiding the waste of computational resources (Elwasif et al., 2011; Garrido et al., 2006; Pentland et al., 2021). Extending parareal to the multivariate case in (1.1) is straightforward: see Gander and Vandewalle (2007) for notation and Pentland et al. (2021) for pseudocode.

2.3. Convergence and complexity

After k iterations, the first k time slices (at minimum) are converged, as the exact initial condition (u_0) has been propagated by $\mathcal{F}$ at least k times. Therefore, in the worst case, parareal converges in $k = J$ iterations, equivalent to calculating (2.1) serially, at an even higher computational cost. Convergence[2] in $k \ll J$ iterations is necessary if parallel speed-up is to be realised.

Without loss of generality, assume running $\mathcal{F}$ over any $[t_j, t_{j+1}]$, $j \in \{0, \dots, J-1\}$, takes wallclock time $T_\mathcal{F}$ (denote time $T_\mathcal{G}$ similarly for $\mathcal{G}$). Therefore, calculating (2.1) using $\mathcal{F}$ serially, takes approximately $T_{\text{serial}} = JT_\mathcal{F}$ seconds. Using parareal, the total wallclock time (in the worst case, excluding any serial overheads) can be approximated by

$$T_{\text{para}} \approx \underbrace{JT_\mathcal{G}}_{\text{Iteration } 0} + \sum_{j=1}^{k} \underbrace{\left(T_\mathcal{F} + (J-j)T_\mathcal{G}\right)}_{\text{Iterations } 1 \text{ to } k} = kT_\mathcal{F} + (k+1)\left(J - \frac{k}{2}\right)T_\mathcal{G}. \tag{2.5}$$

The approximate parallel speed-up is therefore

$$S_{\text{para}} \approx \frac{T_{\text{serial}}}{T_{\text{para}}} = \left[\frac{k}{J} + (k+1)\left(1 - \frac{k}{2J}\right)\frac{T_\mathcal{G}}{T_\mathcal{F}}\right]^{-1}. \tag{2.6}$$

To maximise (2.6), both the convergence rate k and the ratio $T_\mathcal{G}/T_\mathcal{F}$ should be as small as possible. In practice, however, there is a trade-off between these two quantities as fast $\mathcal{G}$ solvers (with sufficient accuracy to still guarantee convergence) typically require more iterations to converge, increasing k. An illustration of the computational task scheduling during the first few iterations of parareal vs. a full serial integration is given in Figure 2.1.

Selecting a fast but accurate coarse solver remains a trial and error process, entirely dependent on the system being solved. Typically, $\mathcal{G}$ is chosen such that it has a coarser temporal resolution/lower numerical accuracy (Baffico et al., 2002; Farhat and Chandesris, 2003; Samaddar et al., 2010; Trindade and Pereira, 2006), a coarser spatial resolution (when solving PDEs) (Ruprecht, 2014; Samaddar et al., 2019), and/or uses simplified model equations (Engblom, 2009; Legoll et al., 2020; Meng et al., 2020) compared to $\mathcal{F}$. In Section 3, we aim to widen the pool of choices for $\mathcal{G}$ by using a GP emulator to capture variability in the residual $\mathcal{F} - \mathcal{G}$ and showcase its effectiveness by demonstrating that GPareareal can converge to a solution in cases where parareal cannot in Section 4.

3. GPareareal

In this section, we present our GPareareal algorithm, in which a Gaussian process (GP) emulator is used in the analogue of parareal's predictor-corrector step. Suppose we seek the same high resolution numerical solutions to (1.1) as expressed in (2.1), denoted now as V_j instead of U_j. Furthermore, we denote the iteratively improved approximations in GPareareal at iteration k as V_j^k (as before, $V_0^k = V_0 = u_0$).

[2] For parareal to converge, the solvers $\mathcal{F}$ and $\mathcal{G}$ must satisfy specific mathematical conditions (Bal, 2005; Maday and Turinici, 2005).

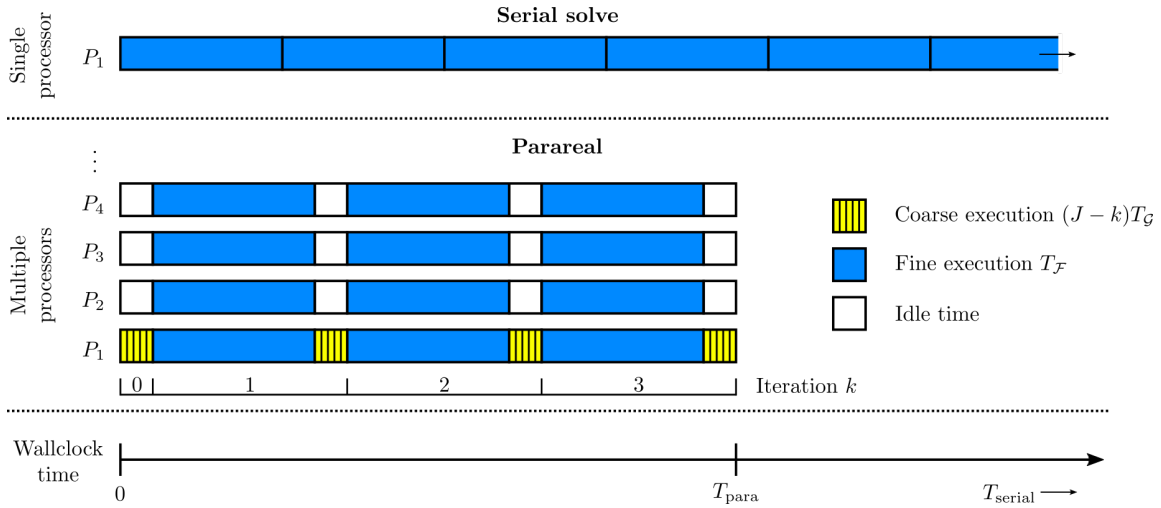


Figure 2.1 Computational task scheduling during three iterations of parareal as compared with a full serial integration. The coloured blocks represent the wallclock time any given processor spends on a task. Coarse runs are shown in yellow, fine runs in blue, and any idle time in white. The wallclock time is given on the axis at the bottom, indicating both T_{para} and T_{serial}.

In parareal, the predictor-corrector (2.3) updates the numerical solutions at iteration k using a correction term based on information calculated during the *previous* iteration $k-1$. We propose the following refinement rule, again based on a coarse prediction and multi-fidelity correction, that instead refines solutions using information from the *current* iteration k, rather than $k-1$:

$$V_{j+1}^k = \mathcal{F}(V_j^k) = (\mathcal{F}-\mathcal{G}+\mathcal{G})(V_j^k) = \underbrace{(\mathcal{F}-\mathcal{G})(V_j^k)}_{\text{correction}} + \underbrace{\mathcal{G}(V_j^k)}_{\text{prediction}} \quad k \geqslant 1,\ j = 0,\dots,J-1. \tag{3.1}$$

Given V_j^k is known, the prediction is rapidly calculable, however the correction is not known explicitly without running $\mathcal{F}$ at expensive cost. We propose using a GP emulator to model this correction term, trained on *all* previously obtained $\mathcal{F}$ and $\mathcal{G}$ evaluations. The emulator returns a Gaussian distribution over $(\mathcal{F}-\mathcal{G})(V_j^k)$ from which we can extract an explicit value and carry out the refinement in (3.1).

In Section 3.1, we present the algorithm, giving an explanation of the kernel hyperparameter optimisation process in Section 3.2 and convergence process in Section 3.3. In Section 3.4, we detail the computational complexity, remarking that given large enough runtimes for the fine solver, an iteration of GParareal runs in approximately the same wallclock time as parareal. Again, to simplify notation, we first detail GParareal for an autonomous scalar-valued ODE, i.e. $\boldsymbol{f}(t,\boldsymbol{u}(t)) \coloneqq f(u(t))$ in (1.1). The extension to the multivariate non-autonomous case is described in Section 3.5.

3.1. The algorithm

Gaussian process emulator

Before solving (1.1), we define a GP prior (Kennedy and O'Hagan, 2001; Rasmussen, 2004) to emulate the unknown correction function $\mathcal{F}-\mathcal{G}$. This function maps an initial value $x_j \in \mathbb{R}$ at time t_j to the residual difference between $\mathcal{F}(x)$ and $\mathcal{G}(x)$ at time t_{j+1}. More formally, we define the GP prior

$$\mathcal{F}-\mathcal{G} \sim \mathcal{GP}(m,\kappa), \tag{3.2}$$

with mean function $m\colon \mathbb{R} \to \mathbb{R}$ and covariance kernel $\kappa\colon \mathbb{R} \times \mathbb{R} \to \mathbb{R}$. Given some vectors of initial values, $\boldsymbol{x}, \boldsymbol{x}' \in \mathbb{R}^J$, the corresponding vector of means 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, depending on the accuracy of both $\mathcal{F}$ and $\mathcal{G}$, hence we define a zero-mean process, i.e. $m(x_j) = 0$. Ideally, the covariance kernel will be chosen based on any prior knowledge of the solution to (1.1), e.g. regularity/smoothness. If no information is available *a priori* to simulation, we are free to select any appropriate kernel. In this work we use the square exponential (SE) kernel

$$\kappa(x, x') = \sigma^2 \exp\left(-\frac{(x - x')^2}{2\ell^2}\right), \quad \text{for some} \quad x, x' \in \mathbb{R}. \tag{3.3}$$

The kernel hyperparameters, denoting the output length scale σ^2 and input length scale ℓ^2, are referred to collectively in the vector $\boldsymbol{\theta}$ and need to be initialised prior to simulation. The algorithm proceeds as follows; see Appendix A for pseudocode.

Iteration $k = 0$

Firstly, run $\mathcal{G}$ sequentially from the exact initial value, on a single processor, to locate the coarse solutions

$$V_{j+1}^0 = \mathcal{G}(V_j^0) \quad j = 0, \dots, J-1. \tag{3.4}$$

Iteration $k = 1$

Use $\mathcal{F}$ to propagate these values (3.4) on each time slice in *parallel*, on J processors, to obtain the following values at t_{j+1}

$$\mathcal{F}(V_j^0) \quad j = 0, \dots, J-1. \tag{3.5}$$

At this stage, we diverge from the parareal method. Given the initially propagated values $\boldsymbol{x} = (V_0^0, \dots, V_{J-1}^0)^\intercal$, evaluate $\mathcal{F} - \mathcal{G}$ using (3.4) and (3.5) such that

$$\boldsymbol{y} \coloneqq \big((\mathcal{F} - \mathcal{G})(x_j)\big)_{j=0,\dots,J-1}. \tag{3.6}$$

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 3.2. Conditioning the prior (3.2) using the acquisition data $\boldsymbol{x}$ and $\boldsymbol{y}$, the GP posterior over $(\mathcal{F} - \mathcal{G})(x')$, where $x' \in \mathbb{R}$ is some initial value in the state space, is given by

$$(\mathcal{F} - \mathcal{G})(x') \mid \boldsymbol{x}, \boldsymbol{y} \sim \mathcal{N}\big(\hat{\mu}(x'), \hat{K}(x', x')\big), \tag{3.7}$$

with mean

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

and variance

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

Now we wish to determine refined solutions V_j^1 at each mesh point. Given $\mathcal{F}$ has been run once, the exact solution is known up to time t_1. Specifically, at t_0 we know $V_0^1 = V_0$ and at t_1 we know $V_1^1 = V_1 = \mathcal{F}(V_0^1)$. At t_2, the exact solution $V_2 = \mathcal{F}(V_1^1)$ is unknown, hence we need to calculate its value without running $\mathcal{F}$ again. To do this, we re-write the exact solution using (3.1):

$$V_2^1 = \mathcal{F}(V_1^1) = (\mathcal{F} - \mathcal{G} + \mathcal{G})(V_1^1) = \underbrace{(\mathcal{F} - \mathcal{G})(V_1^1)}_{\text{correction}} + \underbrace{\mathcal{G}(V_1^1)}_{\text{prediction}}. \tag{3.8}$$

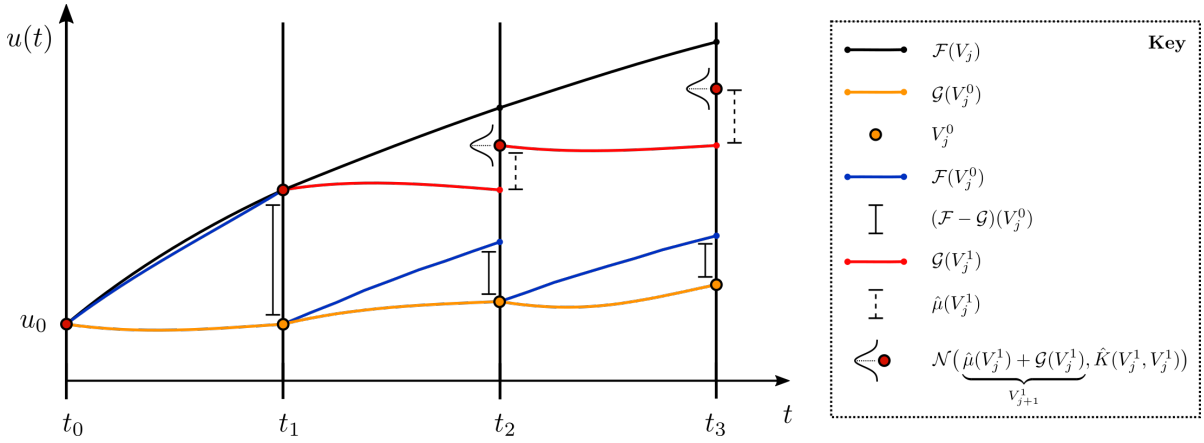


Figure 3.1 Schematic of the first iteration of GPareareal. The 'exact' solution over $[t_0, t_3]$ is shown in black with the first coarse and fine (parallel) runs given in yellow and blue respectively. Solid bars represent the residual between these solutions (3.6). The predictions, i.e. the second coarse runs, are shown in red and the corresponding corrections from the GP emulator are represented by the dashed bars. The refined solutions (3.12) at the end of the iteration are represented by the red dots.

Both terms in (3.8) are initially unknown, but the prediction can be calculated rapidly at low computational cost while the correction can be inferred using the GP posterior (3.7) with $x' = V_1^1$. Therefore, we obtain a Gaussian distribution over the solution

$$V_2^1 \sim \mathcal{N}\big(\hat{\mu}(V_1^1) + \mathcal{G}(V_1^1), \hat{K}(V_1^1, V_1^1)\big), \tag{3.9}$$

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 V_2^1 using $\mathcal{G}$ is computationally infeasible for nonlinear IVPs. To tackle this and be able to propagate V_2^1, we approximate the distribution by taking its mean value,

$$V_2^1 = \hat{\mu}(V_1^1) + \mathcal{G}(V_1^1). \tag{3.10}$$

This approximation is a convenient way of minimising computational cost, at the price of ignoring uncertainty in the GP emulator — see Section 5 for a discussion of possible alternatives.

The refinement process, applying (3.1) and then approximating its Gaussian distribution by taking its expectation, is repeated sequentially for later t_j, yielding the approximate solutions

$$V_{j+1}^1 = \hat{\mu}(V_j^1) + \mathcal{G}(V_j^1) \quad \text{for} \quad j = 2, \ldots, J-1. \tag{3.11}$$

This process is illustrated in Figure 3.1. Finally, we impose stopping criteria (2.4), identifying which V_j^1 for $j \leqslant I$ have converged. Using the same stopping criteria as parareal will allow us to compare the performance of both algorithms in Section 4.

Iteration $k \geqslant 2$

If the stopping criteria is not met, i.e. $I < J$, we can iteratively refine any unconverged solutions by re-running the previous steps. This means calculating $\mathcal{F}(V_j^{k-1})$, $j = I, \ldots, J-1$, in parallel and generating new evaluations $\hat{\boldsymbol{y}} = \big((\mathcal{F}-\mathcal{G})(V_j^{k-1})\big)_{j=I,\ldots,J-1}^{\intercal}$, with corresponding inputs $\hat{\boldsymbol{x}} = (V_I^{k-1}, \ldots, V_{J-1}^{k-1})^{\intercal}$. 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. The refinement step is then applied such that we obtain

$$V_{j+1}^{k} = \hat{\mu}(V_j^k) + \mathcal{G}(V_j^k) \quad \text{for} \quad j = I+1, \ldots, J-1. \tag{3.12}$$

Once $I = J$, the solution, the number of iterations k taken to converge, and the acquisition data $\boldsymbol{x}$ and $\boldsymbol{y}$ are returned.

3.2. 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 inputs $\boldsymbol{x}$). We optimise each $\theta_i \in \boldsymbol{\theta}$ such that it maximises its (log) marginal likelihood (Rasmussen, 2004). To do this, first define $h(x) \coloneqq (\mathcal{F} - \mathcal{G})(x)$ and $\boldsymbol{h} \coloneqq (h(x_j))_{j=0,\ldots,N-1}$, where N is the current length of $\boldsymbol{x}$. Given the evaluations $\boldsymbol{y}$ are noise-free, the likelihood of obtaining such data is $p(\boldsymbol{y}|\boldsymbol{h}, \boldsymbol{x}, \boldsymbol{\theta}) = \delta(\boldsymbol{y} - \boldsymbol{h})$, where $\delta(\cdot)$ is the multidimensional Dirac delta function. The marginal likelihood, given $\boldsymbol{x}$ and $\boldsymbol{\theta}$, is therefore

$$p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{\theta}) = \int \underbrace{p(\boldsymbol{y}|\boldsymbol{h}, \boldsymbol{x}, \boldsymbol{\theta})}_{\text{likelihood}} \underbrace{p(\boldsymbol{h}|\boldsymbol{x}, \boldsymbol{\theta})}_{\text{prior}} \, \mathrm{d}\boldsymbol{h} = \int \delta(\boldsymbol{y} - \boldsymbol{h}) \mathcal{N}(\boldsymbol{h}|\boldsymbol{0}, K(\boldsymbol{x}, \boldsymbol{x})) \, \mathrm{d}\boldsymbol{h} = \mathcal{N}(\boldsymbol{y}|\boldsymbol{0}, K(\boldsymbol{x}, \boldsymbol{x})). \tag{3.13}$$

Taking the logarithm, we want to locate the maximum *a posteriori* (MAP) estimate of the log-marginal likelihood, i.e.

$$\max_{\boldsymbol{\theta}} \left[\log p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{\theta}) \right] = \max_{\boldsymbol{\theta}} \left[-\frac{1}{2} \boldsymbol{y}^{\mathrm{T}} K^{-1} \boldsymbol{y} - \frac{1}{2} \log |K| - \frac{J}{2} \log 2\pi \right]. \tag{3.14}$$

These hyperparameters can be estimated numerically using any gradient-based optimiser at low computational cost, when compared to the cost of a single $\mathcal{F}$ run.

3.3. Convergence

We make the following two remarks on convergence of GParareal:

- In the limit of infinite evaluations, the expected value of the emulator will return the exact value of $\mathcal{F} - \mathcal{G}$. Therefore, the refinement step (3.11) will return the exact value of V_j.
- Similar to the worst-case scenario in parareal, should GParareal fail to locate a solution in parallel and take $k = J$ iterations to converge, it will return the sequentially calculated fine solution (at extra computational cost).

Just as parareal does, GParareal calculates a deterministic set of numerical solutions to (1.1) that converge toward the exact solutions, assuming sufficiently many evaluations of $\mathcal{F} - \mathcal{G}$ are collected (or available via a legacy archive). It should also be noted that solutions obtained with GParareal are deterministic and will not change upon multiple simulations of the algorithm (unless one uses different sets of legacy data).

3.4. Computational complexity

The computational cost of running an iteration in GParareal is approximately the same as running one in parareal. The cost of running either solver is identical, recall $T_{\mathcal{G}}$ and $T_{\mathcal{F}}$, and any serial overheads will too be of similar cost. Additional costs for GParareal arise when (serially) optimising hyperparameters and conditioning the emulator on acquisition/legacy data. During the kth iteration, approximately kJ evaluations have been collected, hence standard

GP optimisation costs $\mathcal{O}(k^2J^2)$ per hyperparameter and $\mathcal{O}(k^3J^3)$ to condition the emulator. We work under the assumption that these serial costs are dwarfed by a single run of the $\mathcal{F}$ solver, considering the optimisation/conditioning wallclock times to be small in comparison. This assumption is reasonable because if these costs begin to overtake those of an $\mathcal{F}$ run, then the problem being solved probably does not require a time-parallel algorithm in the first place. Parallel speed-up can therefore be calculated the same way as parareal — recall (2.6) — and it should be stressed that $T_{\mathcal{F}}$ is large, so any reduction in the number of iterations taken until convergence (k) can result in a large decrease in wallclock time. This means that if GParareal converges in fewer iterations than parareal for a given IVP, it will locate a solution in faster wallclock time — see Section 4.

To highlight the effectiveness of GParareal, we implement a standard, cubic complexity, GP emulator in GParareal. It should therefore be noted that when solving systems of ODEs (Section 3.5), the serial costs associated with this GP implementation will inevitably increase (relative to an $\mathcal{F}$ run), calling for more efficient, less expensive, emulation methods. These could make use of more efficient (even parallelisable) matrix inversion or alternatively one could use a Kalman filter instead — as is done in the ODE filters mentioned previously. One could even optimise each of the hyperparameters in parallel, thereby making use of the pool of idle processors. We aim to highlight the effectiveness of using just a basic GP emulator within GParareal, without incurring the additional "theoretical cost" of implementing more advanced methods just yet.

3.5. Generalisation to ODE systems

The methodology in Section 3.1 can be generalised to solve systems of d autonomous ODEs. Accordingly, the correction term we wish to emulate is now vector-valued, $\mathcal{F} - \mathcal{G} \colon \mathbb{R}^d \to \mathbb{R}^d$, hence we require a vector-valued (or multi-output) GP, rather than a scalar GP.

The simplest approach is to model each output of $\mathcal{F} - \mathcal{G}$ 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 $\boldsymbol{y}_i$, $i = 1, \ldots, d$. If datasets are large, optimising and conditioning of these GPs can be run on the idle processors in parallel to reduce the serial runtime.

The more complex approach is to jointly emulate the outputs of $\mathcal{F} - \mathcal{G}$ 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 the joint, block-diagonal, covariance prior using a linear combination of the separate kernels κ_i. Prior testing revealed that using this method did not improve performance enough to justify the added complexity, $\mathcal{O}(d^3k^3J^3)$ vs. $\mathcal{O}(dk^3J^3)$ in the independent setting (results not reported). Some applications may require correlated output dimensions, hence we note the methodology here for any interested readers.

As a final note, to solve non-autonomous systems of equations, i.e. (1.1), there are two possible approaches. One way is to include the time variable as an extra input to each of the d scalar GPs — this requires a more carefully selected covariance kernel. The other way, is to re-write the d-dimensional non-autonomous system as a system of $d + 1$ autonomous equations and solve as described above — this is the method we use in Section 4.

4. Numerical experiments

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

For simplicity, $\mathcal{F}$ and $\mathcal{G}$ are chosen to be explicit Runge–Kutta (RK) methods of order $p, q \in \{1, 2, 4\}$ respectively ($p \geqslant q$). Let $N_{\mathcal{F}}$ and $N_{\mathcal{G}}$ denote the number of time steps each solver uses over $[t_0, T]$. All aspects relating to the GP emulator are carried out using the `GPstuff` package by Vanhatalo et al. (2013). Each output dimension of $\mathcal{F} - \mathcal{G}$ is modelled independently using its own GP and isotropic SE covariance kernel. Note that all experiments are run on up to 40 CPUs[3].

4.1. FitzHugh–Nagumo model

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

$$\frac{\mathrm{d}u_1}{\mathrm{d}t} = c\big(u_1 - \frac{u_1^3}{3} + u_2\big), \quad \frac{\mathrm{d}u_2}{\mathrm{d}t} = -\frac{1}{c}(u_1 - a + bu_2), \tag{4.1}$$

where we fix parameters $(a, b, c) = (0.2, 0.2, 3)$. We integrate (4.1) over $t \in [0, 40]$, dividing the interval into $J = 40$ slices, and set the tolerance for both GParareal and parareal to $\varepsilon = 10^{-6}$. We use solvers $\mathcal{G} = \text{RK2}$ and $\mathcal{F} = \text{RK4}$ with $N_{\mathcal{G}} = 160$ and $N_{\mathcal{F}} = 1.6 \times 10^8$ steps respectively. Note that the large value of $N_{\mathcal{F}}$ is required so that parallel speedup can be realised (as both algorithms require $T_{\mathcal{G}}/T_{\mathcal{F}} \ll 1$).

In Figure 4.1(a), we solve (4.1) 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 4.1(b)). Note, however, that in Figure 4.1(c), GParareal takes six fewer iterations to converge to these solutions than parareal does. As a result, GParareal locates a solution in a faster wallclock time than parareal, see Figure 4.1(d), with a speedup of 4.3× vs. the serial solver — almost twice the 2.4× speedup obtained by parareal.

To compare the convergence of both methods more broadly, we solve (4.1) for a range of initial values. The heatmap in Figure 4.2(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 the low accuracy of $\mathcal{G}$. In direct contrast, see Figure 4.2(b), GParareal converges more quickly 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 $\mathcal{G}$ has poor accuracy.

Until now, GParareal simulations have been carried out using only acquisition data. In Figure 4.3, we demonstrate how GParareal can use both acquisition and legacy data to converge in fewer iterations than without the legacy data. Approximately $kJ = 5 \times 40 = 200$ legacy data points, obtained solving (4.1) for $\boldsymbol{u}_0 = (-1, 1)^\intercal$, are stored and used to condition the GP emulator prior to solving (4.1) for alternate initial values $\boldsymbol{u}_0 = (0.75, 0.25)^\intercal$. In Figure 4.3(a), we can see that convergence takes two fewer iterations with the legacy data than without. 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 4.3(b). Repeating this experiment with the same legacy data

[3] `MATLAB` code for both GParareal and parareal can be found at: https://github.com/kpentland/GParareal.

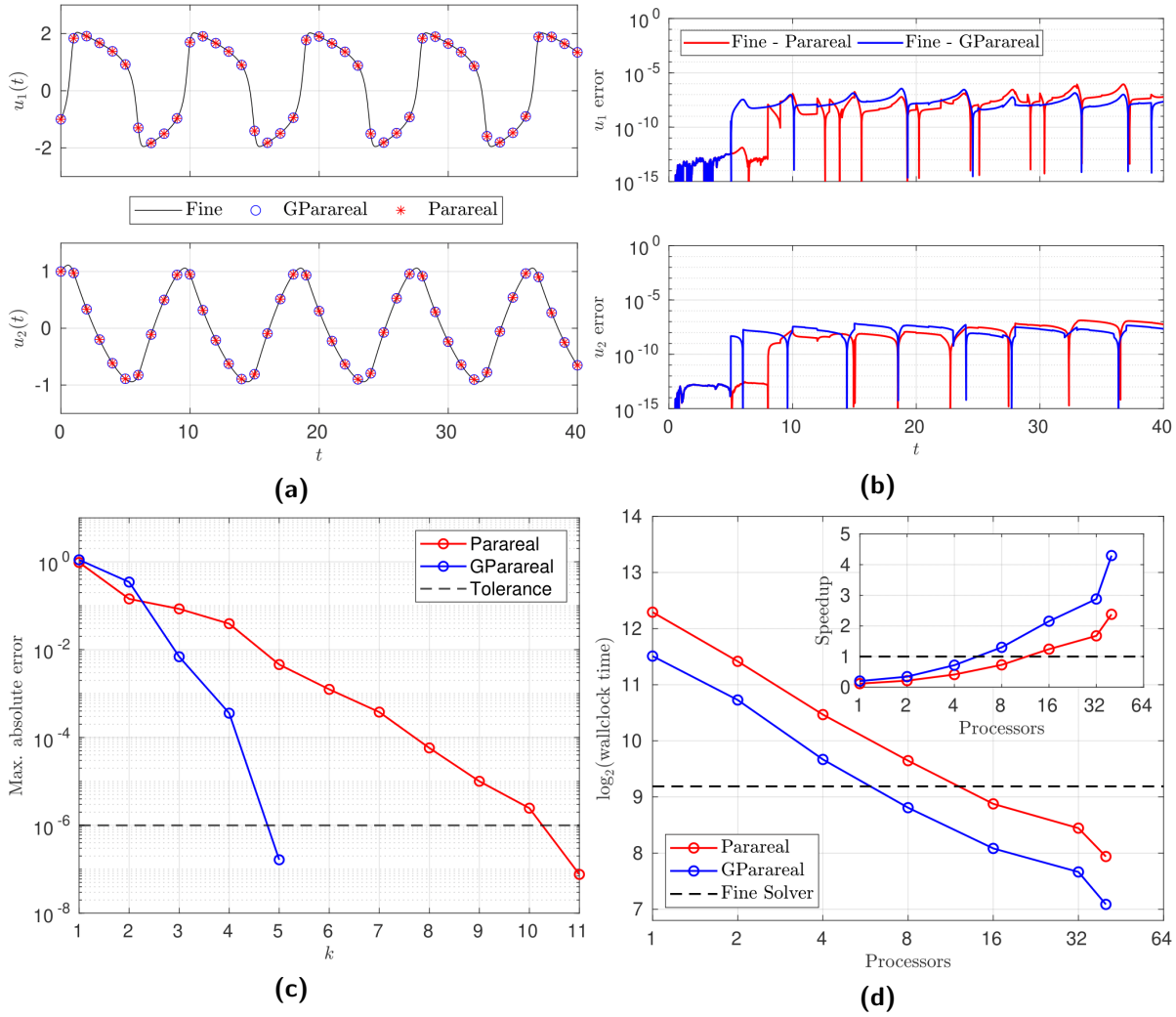
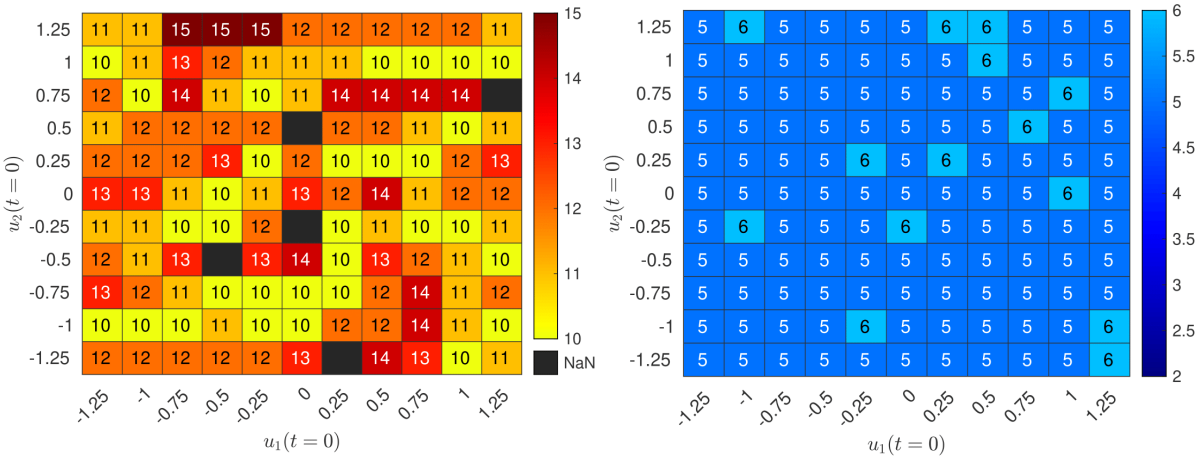


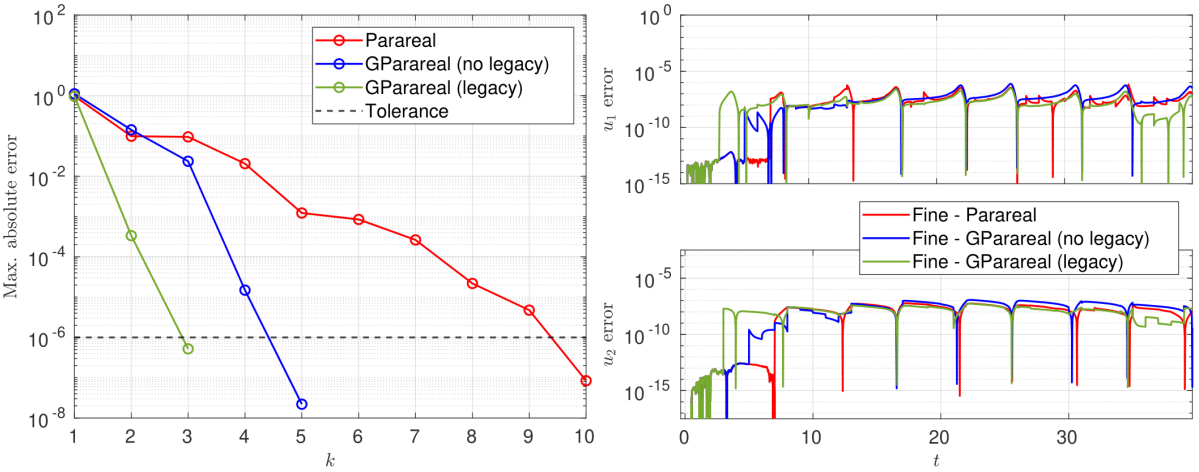
Figure 4.1 Numerical results obtained solving (4.1) for $\boldsymbol{u}_0 = (-1, 1)^\intercal$. (a) Time-dependent solutions using the fine solver, GPararealal, and parareal — both GPararealal and parareal plotted only at time slice boundaries t_i for clarity. (b) The corresponding absolute errors between solutions from GPararealal and parareal vs. the fine solution. (c) Maximum absolute errors from (2.4) of each algorithm at successive iterations k until tolerance ε 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 4.2 Heat maps displaying the number of iterations taken until convergence k of (a) parareal and (b) GParareal when solving the FitzHugh–Nagumo system (4.1) for different initial values $\boldsymbol{u}_0 \in [-1.25, 1.25]^2$. Black boxes indicate where the algorithm returned a `NaN` value during simulation.

for a range of initial values we see that convergence is either unchanged or improved in all cases, see Figure 4.4. It should be noted that conditioning the GP and optimising hyperparameters using the legacy data comes at extra (serial) computational cost and checks should be made to ensure the parallel computations, i.e. $T_{\mathcal{F}}$, still dominate all serial parts of the simulation. These results show that using GParareal (with or without legacy data) we can solve and evaluate the dynamics of the FHN model in significantly lower wallclock time than parareal.



(a) **(b)**

Figure 4.3 Numerical simulations solving (4.1) for $\boldsymbol{u}(0) = (0.75, 0.25)^\intercal$ using GParareal with and without access to legacy data, i.e. data obtained solving (4.1) for $\boldsymbol{u}(0) = (-1, 1)^\intercal$. The parareal simulation of the same problem is also shown for comparison. (a) Maximum absolute errors from (2.4) against iteration number k until tolerance ε met. (b) Time-dependent errors of the corresponding numerical solutions from each simulation vs. the fine solution.

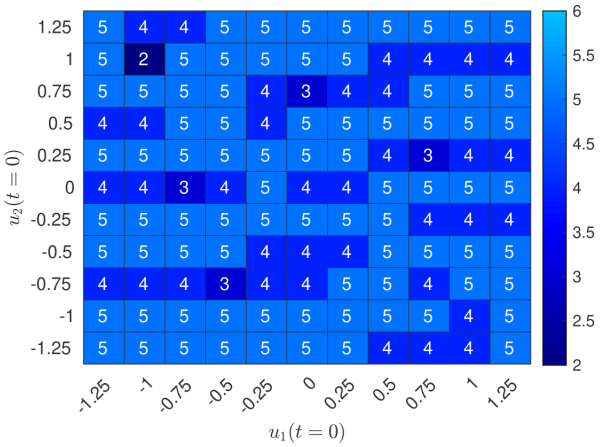


Figure 4.4 Heat map displaying the number of iterations taken until convergence k of GParareal when solving (4.1) for different initial values $\boldsymbol{u}_0 \in [-1.25, 1.25]^2$ — this time using legacy data obtained by solving (4.1) for $\boldsymbol{u}_0 = (-1, 1)^\intercal$.

4.2. Rössler system

Next we solve the Rössler system,

$$\frac{\mathrm{d}u_1}{\mathrm{d}t} = -u_2 - u_3, \quad \frac{\mathrm{d}u_2}{\mathrm{d}t} = u_1 + \hat{a}u_2, \quad \frac{\mathrm{d}u_3}{\mathrm{d}t} = \hat{b} + u_3(u_1 - \hat{c}), \tag{4.2}$$

with parameters $(\hat{a}, \hat{b}, \hat{c}) = (0.2, 0.2, 5.7)$ that cause the system to exhibit chaotic behaviour (Rössler, 1976). Suppose we wish to integrate (4.2) over $t \in [0, 340]$ with initial values $\boldsymbol{u}_0 = (0, -6.78, 0.02)^\intercal$ and solvers $\mathcal{G} = \mathrm{RK1}$ and $\mathcal{F} = \mathrm{RK4}$. The interval is divided into $J = 40$ 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 (4.2) using alternate initial values (as we did in Section 4.1), we instead generate such 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, many dynamical 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., 2021). Such rare metastability may not be revealed/observed until the system has been evolved 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 (4.2) over $t \in [0, 170]$, then we need to reduce J, $N_\mathcal{G}$, and $N_\mathcal{F}$ by a factor of two, i.e. use $J^{(2)} = J/2$, $N_\mathcal{G}^{(2)} = N_\mathcal{G}/2$, 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 $kJ^{(2)} = 9 \times 20 = 180$ legacy evaluations of $\mathcal{F} - \mathcal{G}$ (results not shown). Integrating (4.2) over the full interval $[0, 340]$, GParareal converges in four iterations sooner with the legacy data than without — refer to Figure 4.5(c). In Figure 4.5(d) we can see that using the legacy data attains higher numerical speedup ($3\times$) compared to parareal ($1.6\times$). In addition we have generated even more acquisition data that can be stored in the legacy data archive. In Figure 4.5(a) we see the trajectories from each simulation converging

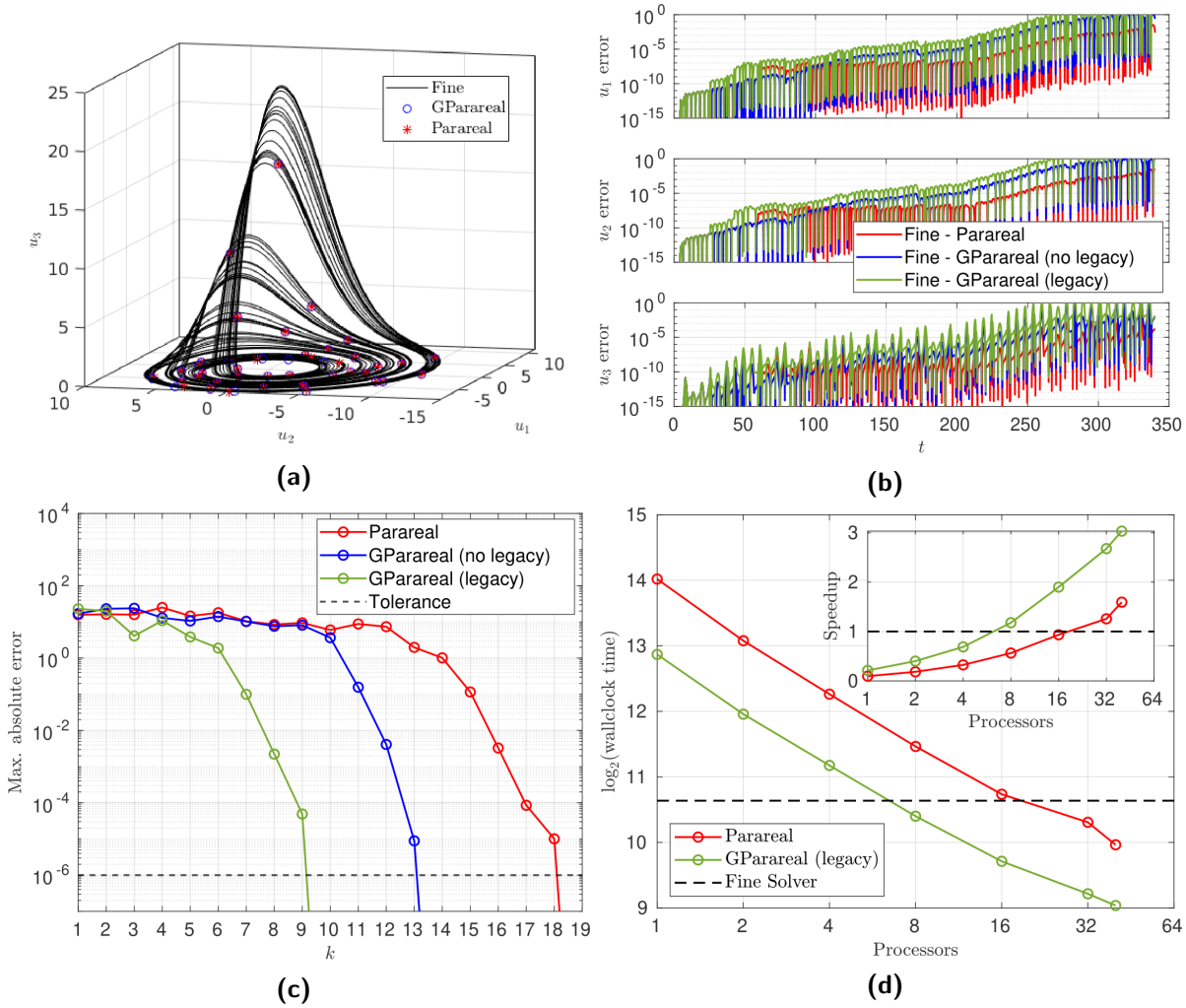


Figure 4.5 Numerical results obtained solving the Rössler system (4.2) over $t \in [0, 340]$. (a) Phase space solutions using the fine solver, GParareal (with legacy data), and parareal — both GParareal and parareal plotted only at times $\boldsymbol{t}$ for clarity. (b) The corresponding absolute errors between solutions from GParareal and parareal vs. the fine solution. (c) Maximum absolute errors from (2.4) of each algorithm at successive iterations k until tolerance ε 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 vs. the serial wallclock time.

toward the Rössler attractor and Figure 4.5(b) illustrates GParareal retaining 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. Discussion

In this paper we present an adaptive time-parallel algorithm (GParareal) that iteratively locates a numerical solution to a system of ODEs. It does so using a predictor-corrector, comprised of numerical solutions from coarse ($\mathcal{G}$) and fine ($\mathcal{F}$) integrators, however, unlike the classical parareal algorithm, it uses a Gaussian process (GP) emulator to infer the correction term

$\mathcal{F} - \mathcal{G}$. The numerical experiments reported in Section 4 demonstrate that GParareal performs favourably compared to parareal, converging in fewer iterations and achieving increased parallel speedup for a number of low-dimensional ODE systems. We also demonstrate how GParareal can make use of legacy data, i.e. prior $\mathcal{F}$ and $\mathcal{G}$ 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 — something that existing time-parallel methods do not do.

In Section 4.1, using just the data obtained during simulation (acquisition data), GParareal achieves an almost two-fold increase in speedup over parareal when solving the FitzHugh-Nagumo 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 was shown to converge in even fewer iterations. Similar results were illustrated for the Rössler system in Section 4.2 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 all cases, the solutions generated by GParareal were of a numerical accuracy comparable to those found using parareal.

In its current implementation, GParareal may, however, suffer from the curse of dimensionality as an increasing number of evaluations will be required to learn $\mathcal{F} - \mathcal{G}$ sufficiently well when solving higher-dimensional systems. One option to remedy this is to model $\mathcal{F} - \mathcal{G}$ using more sophisticated (non-cubic complexity) emulation techniques. Another way could be to obtain more acquisition data, i.e. launch more $\mathcal{F}$ and $\mathcal{G}$ runs using the idle processors during later iterations, to further train the emulator at no additional computational cost. It should also be noted that GParareal may not always provide faster convergence using legacy data if such legacy evaluations of $\mathcal{F} - \mathcal{G}$ lay 'far away', i.e. over one or two input length-scales away, from the initial values of interest in the current simulation. In this case, GParareal would rely more heavily on its acquisition data. There is no immediate remedy for such a situation, but using a fallback parareal correction, as suggested in the next paragraph, could be an option.

In (3.9) we approximate a Gaussian distribution over the exact solution by taking its expected value, ignoring uncertainty in the GP prediction of $\mathcal{F} - \mathcal{G}$. During early iterations, when little acquisition data may be available, this uncertainty (i.e. the variance) could be large. Ideally, one would propagate the full uncertainty using the coarse solver to the next time step, however this is a computationally expensive process when solving nonlinear ODE problems. Instead of taking the expected value, one could approximate the numerical solution by taking a random sample from (3.9). A sampling-based solver such as this would return a stochastic solution to the ODE, much like the stochastic parareal algorithm presented in Pentland et al. (2021). It is unclear how this algorithm would perform vs. parareal (or even stochastic parareal) however it could still make use of legacy data following successive independent simulations. Another possible alternative to approximating (3.9) with the expected value arises if the input initial value is at least one or two length-scale distances away from any other known input value in our acquired dataset. In this case we then might expect the GP prediction of the mean in (3.9) to be poor. In this case, a fallback to the deterministic parareal correction for $\mathcal{F} - \mathcal{G}$ (see (2.3)) could be built in as a next best correction to the coarse solution. Among others, these are two alternative formulations of GParareal that are worth investigating in the future.

Follow-up work will focus on extending GParareal, using some of the methods suggested above, to solve higher-dimensional systems of ODEs in parallel. In particular, we aim to develop a truly probabilistic numerical method that can account for the inherent uncertainty in the GP emulator, returning a probability distribution rather than point estimates over the solution.

Acknowledgements

KP is funded by the Engineering and Physical Sciences Research Council through the MathSys II CDT (grant EP/S022244/1) as well as the Culham Centre for Fusion Energy. TJS is partially supported by the Deutsche Forschungsgemeinschaft through project 415980428. This work has partly been carried out within the framework of the EUROfusion Consortium and has received funding from the Euratom research and training programme 2014–2018 and 2019–2020 under grant agreement No. 633053. The authors would also like to acknowledge the University of Warwick Scientific Computing Research Technology Platform for assistance in the research described in this paper. The views and opinions expressed herein do not necessarily reflect those of any of the above-named institutions or funding agencies.

References

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. *Found. Trends Mach. Learn.*, 4:195–266, 2011. doi:10.1561/2200000036.

L. Baffico, S. Bernard, Y. Maday, G. Turinici, and G. Zérah. Parallel-in-time molecular-dynamics simulations. *Phys. Rev. E - Stat. Phys. Plasmas Fluids Relat. Interdiscip. Top.*, 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. *Lect. Notes Comput. Sci. Eng.*, 40:425–432, 2005. doi:10.1007/3-540-26825-1_43.

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.

A. Clarke, C. Davies, D. Ruprecht, and S. Tobias. Parallel-in-time integration of kinematic dynamos. *J. Comput. Phys. X*, 7, 2020. doi:10.1016/j.jcpx.2020.100057.

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.

X. Dai, C. Le Bris, F. Legoll, and Y. Maday. Symmetric parareal algorithms for Hamiltonian systems. *ESAIM Math. Model. Numer. Anal.*, 47:717–742, 2013. doi:10.1051/m2an/2012046.

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.

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, New York, USA, 2011. ACM Press. doi:10.1145/2132876.2132883.

S. Engblom. Parallel in time simulation of multiscale stochastic chemical kinetics. *Multiscale Model. Simul.*, 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. *Int. J. Numer. Methods Eng.*, 58: 1397–1434, 2003. doi:10.1002/nme.860.

R. FitzHugh. Impulses and Physiological States in Theoretical Models of Nerve Membrane. *Biophys. J.*, 1:445–466, 1961. doi:10.1016/S0006-3495(61)86902-6.

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 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 S. Vandewalle. Analysis of the parareal time-parallel time-integration method. *SIAM J. Sci. Comput.*, 29:556–578, 2007. doi:10.1137/05064607X.

I. Garrido, B. Lee, G. E. Fladmark, and M. S. Espedal. Convergent iterative schemes for time-parallelization. *Math. Comput.*, 75(255):1403–1428, 2006. doi:10.1090/S0025-5718-06-01832-1.

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.

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.

F. P. Hamon, M. Schreiber, and M. Minion. Parallel-in-time multi-level integration of the shallow-water equations on the rotating sphere. *J. Comput. Phys.*, 407:109210, 2020. doi:10.1016/j.jcp.2019.109210.

P. Hennig, M. A. Osborne, and M. Girolami. Probabilistic numerics and uncertainty in computations. *Proc. R. Soc. Math. Phys. Eng. Sci.*, 471:20150142, 2015. doi:10.1098/rspa.2015.0142.

M. C. Kennedy and A. O'Hagan. Bayesian calibration of computer models. *J. R. Stat. Soc. Ser. B Stat. Methodol.*, 63:425–464, 2001. doi:10.1111/1467-9868.00294.

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. *Stat. Comput.*, 30(6):1791–1816, 2020. doi:10.1007/s11222-020-09972-4.

N. Krämer, N. Bosch, J. Schmidt, and P. Hennig. Probabilistic ODE Solutions in Millions of Dimensions, 2021. arXiv:2110.11812.

F. Legoll, T. Lelièvre, K. Myerscough, and G. Samaey. Parareal computation of stochastic differential equations with time-scale separation: A numerical convergence study. *Comput. Vis. Sci.*, 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, 2021. URL https://hal.archives-ouvertes.fr/hal-03189428.

J. L. Lions, Y. Maday, and G. Turinici. Résolution d'EDP par un schéma en temps ≪pararéel≫. *Comptes Rendus Acad. Sci. Ser. I Math.*, 332(7):661–668, 2001. doi:10.1016/S0764-4442(00)01793-6.

Y. Maday and O. Mula. An adaptive parareal algorithm. *J. Comput. Appl. Math.*, 377: 112915–112915, 2020. doi:10.1016/j.cam.2020.112915.

Y. Maday and G. Turinici. The parareal in time iterative solver: A further direction to parallel implementation. *Lect. Notes Comput. Sci. Eng.*, 40:441–448, 2005. doi:10.1007/3-540-26825-1_45.

A. Mann. Core Concept: Nascent exascale supercomputers offer promise, present challenges. *PNAS*, 117:22623–22625, 2020. doi:10.1073/pnas.2015968117.

X. Meng, Z. Li, D. Zhang, and G. E. Karniadakis. PPINN: Parareal physics-informed neural network for time-dependent PDEs. *Comput. Methods Appl. Mech. Eng.*, 370:113250, 2020. doi:10.1016/j.cma.2020.113250.

J. Nagumo, S. Arimoto, and S. Yoshizawa. An active pulse transmission line simulating nerve axon. *Proc. IRE*, 50:2061–2070, 1962. doi:10.1109/JRPROC.1962.288235.

C. J. Oates and T. J. Sullivan. A modern retrospective on probabilistic numerics. *Stat. Comput.*, 29:1335–1351, 2019. doi:10.1007/s11222-019-09902-z.

A. O'Hagan. Curve fitting and optimal design for prediction. *J. R. Stat. Soc. Ser. B Methodol.*, 40:1–24, 1978. doi:10.1111/j.2517-6161.1978.tb01643.x.

B. W. Ong and J. B. Schroder. Applications of time parallelization. *Comput. Vis. Sci.*, 23, 2020. doi:10.1007/s00791-020-00331-4.

K. Pentland, M. Tamborrino, D. Samaddar, and L. C. Appel. Stochastic parareal: An application of probabilistic methods to time-parallelisation, 2021. arXiv:2106.10139.

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.

O. E. Rössler. An equation for continuous chaos. *Physics Letters A*, 57:397–398, 1976. doi:10.1016/0375-9601(76)90101-8.

D. Ruprecht. Convergence of Parareal with spatial coarsening. *PAMM*, 14:1031–1034, 2014. doi:10.1002/pamm.201410490.

D. Samaddar, D. E. Newman, and R. Sánchez. Parallelization in time of numerical simulations of fully-developed plasma turbulence using the parareal algorithm. *J. Comput. Phys.*, 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. *Comput. Phys. Commun.*, 235:246–257, 2019. doi:10.1016/j.cpc.2018.08.007.

M. Schober, S. Särkkä, and P. Hennig. A probabilistic model for the numerical solution of initial value problems. *Stat. Comput.*, 29:99–122, 2019. doi:10.1007/s11222-017-9798-7.

J. M. F. Trindade and J. C. F. Pereira. Parallel-in-time simulation of two-dimensional, unsteady, incompressible laminar flows. *Numer. Heat Transf. Part B Fundam.*, 50:25–40, 2006. doi:10.1080/10407790500459379.

F. Tronarp, H. Kersting, S. Särkkä, and P. Hennig. Probabilistic solutions to ordinary differential equations as nonlinear Bayesian filtering: A new perspective. *Stat Comput*, 29:1297–1315, 2019. doi:10.1007/s11222-019-09900-1.

J. Vanhatalo, J. Riihimäki, J. Hartikainen, P. Jylänki, V. Tolvanen, and A. Vehtari. GPstuff: Bayesian Modeling with Gaussian processes. *J. Mach. Learn. Res.*, 14:1175–1179, 2013. URL https://www.jmlr.org/papers/volume14/vanhatalo13a/vanhatalo13a.pdf.

A. Psuedocode for GParareal

Algorithm 1: GParareal

Initialise: Set counters $k = I = 0$ and define V_j^k, $\hat{V}_j^k$ and $\tilde{V}_j^k$ as the refined, coarse, and fine solutions at the jth mesh point and kth iteration respectively (note $V_0^k = \hat{V}_0^k = \tilde{V}_0^k = u^0 \ \forall k$). If known, initialise any legacy input data $\boldsymbol{x}$, output data $\boldsymbol{y}$, and hyperparameters $\boldsymbol{\theta}$.

%Calculate approximate initial values at each t_j by running $\mathcal{G}$ serially.

for $j = 1$ **to** J **do**
  $\hat{V}_j^0 = \mathcal{G}(\hat{V}_{j-1}^0)$;
  $V_j^0 = \hat{V}_j^0$;
end
for $k = 1$ **to** J **do**
  %Propagate refined solutions (from iteration $k-1$) on unconverged sub-intervals by running $\mathcal{F}$ in parallel.
  for $j = I+1$ **to** J **do**
    $\tilde{V}_j^{k-1} = \mathcal{F}(V_{j-1}^{k-1})$;
  end
  $I = I + 1$;
  $V_I^k = \tilde{V}_I^{k-1}$ for all k ; %copy converged solution at t_I to future k.
  $\boldsymbol{x} = \text{append}(\boldsymbol{x}, (V_I^{k-1}, \dots, V_{J-1}^{k-1})^{\mathrm{T}})$; %collect new input data.
  $\boldsymbol{y} = \text{append}(\boldsymbol{y}, (\tilde{V}_{I+1}^{k-1} - \hat{V}_{I+1}^{k-1}, \dots, \tilde{V}_J^{k-1} - \hat{V}_J^{k-1})^{\mathrm{T}})$; %collect new output data.
  $\hat{\boldsymbol{\theta}} = \text{GPoptimise}(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\theta})$; %optimise hyperparameters.
  %Propagate refined solution (at iteration k) with $\mathcal{G}$, then correct using the expected value of the GP prediction (3.7) (this step *cannot* be carried out in parallel).
  for $j = I+1$ **to** J **do**
    $x^\star = V_{j-1}^k$;
    $\hat{V}_j^k = \mathcal{G}(x^\star)$;
    $y^\star = \text{GPpredict}(\boldsymbol{x}, \boldsymbol{y}, \hat{\boldsymbol{\theta}}, x^\star)$; %returns Gaussian random variable
    $V_j^k = \mathbb{E}(y^\star) + \hat{V}_j^k$;
  end
  %Evaluate the stopping criterion, saving all solutions up to t_I.
  $I = \max_{n \in \{I, \dots, N\}} |V_i^k - V_i^{k-1}| < \varepsilon \quad \forall i < n$;
  if $I = N$ **then**
    return k, V^k, $\boldsymbol{X}$, $\boldsymbol{Y}$; %if tolerance met for all time steps, stop.
  end
end