



# GParareal: a time-parallel ODE solver using Gaussian process emulation

Kamran Pentland<sup>1</sup> · Massimiliano Tamborrino<sup>1</sup> · T. J. Sullivan<sup>1,2</sup> · James Buchanan<sup>3</sup> · L. C. Appel<sup>3</sup>

Received: 1 February 2022 / Accepted: 7 December 2022 / Published online: 21 December 2022  
© The Author(s) 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. In this work, we propose a time-parallel algorithm (*GParareal*) 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 we demonstrate, on a number of IVPs, that *GParareal* can converge in fewer iterations than *parareal*, leading to an increase in parallel speed-up. *GParareal* also manages to locate solutions to certain IVPs where *parareal* fails and has the additional advantage of being able to use archives of legacy solutions, e.g. solutions from prior runs of the IVP for different initial conditions, to further accelerate convergence of the method — something that existing time-parallel methods do not do.

**Keywords** Gaussian process emulation · Parallel-in-time · Parareal · Initial value problems

**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

$$\frac{du}{dt} = f(t, u(t)) \quad \text{over } t \in [t_0, T], \quad \text{with} \\ u(t_0) = u_0 \in \mathcal{U} \subset \mathbb{R}^d, \quad (1.1)$$

where  $f: [t_0, T] \times \mathcal{U} \rightarrow \mathbb{R}^d$  is a nonlinear function with sufficiently many continuous partial derivatives,  $u: [t_0, T] \rightarrow \mathcal{U}$  is the time-dependent solution, and  $u_0$  is the initial value at time  $t_0$ . We seek numerical solutions  $U_j \approx u(t_j)$  to the initial value problem (IVP) in (1.1) on a pre-defined mesh  $\mathbf{t} = (t_0, \dots, t_J)$ , 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,  $f$ , is so large that such numerical solutions take hours, days, or even weeks to obtain using classical sequential integration methods, e.g. implicit/explicit Runge–Kutta methods (Hairer et al. 1993). Expensive vector fields  $f$  can, for example, arise when (spatially) discretising partial differential equations (PDEs) into a system of ODEs. Runtime

✉ Kamran Pentland  
kamran.pentland@warwick.ac.uk

Massimiliano Tamborrino  
massimiliano.tamborrino@warwick.ac.uk

T. J. Sullivan  
t.j.sullivan@warwick.ac.uk

James Buchanan  
james.buchanan@ukaea.uk

L. C. Appel  
lynton.appel@ukaea.uk

<sup>1</sup> University of Warwick, Coventry CV4 7AL, United Kingdom

<sup>2</sup> Alan Turing Institute, British Library, 96 Euston Road, London NW1 2DB, United Kingdom

<sup>3</sup> Culham Centre for Fusion Energy, Culham Science Centre, Abingdon Oxfordshire OX14 3DB, United Kingdom

issues also arise when solving IVPs with spatial or other non-temporal dependencies in 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  $U_j^k$  to (1.1) in parallel. At any iteration  $k \in \{1, \dots, J\}$  of parareal, the ‘correction’ is given by the residual between fine and coarse solutions obtained during iteration  $k - 1$  (further details are provided in Sect. 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 (Elwasif et al. 2011; Ait-Ameur et al. 2020; Maday and Mula 2020; Dai et al. 2013; Pentland et al. 2022). 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 determine 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 to be investigated further.

In particular, we use a Gaussian process (GP) emulator (O’Hagan 1978; Rasmussen 2003) to rapidly infer the (expensive-to-simulate) multi-fidelity correction term in parareal. The emulator is trained using 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-type 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. In addition to using an emula-

tor, 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 that the fine solver is of sufficient accuracy to exactly solve the IVP, the algorithm presented in this paper determines numerical solutions  $U_j^k$  that converge (assuming the emulator is sufficiently well trained) toward the exact solutions  $U_j$  over a number of iterations. 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 of initial values  $u_0$  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. solution information accumulated between independent simulations, can be used to inform future simulations by increasing the size of the dataset available to the emulator. 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. 2022; Oates and Sullivan 2019), where ‘ODE filters’ have been developed to solve ODEs using GP regression techniques. Instead of calculating a numerical solution on the mesh  $t$ , as classical integration methods do, ODE filters return a probability measure over the solution at any  $t \in [t_0, T]$  (Schober et al. 2019; Tronarp et al. 2019; Bosch et al. 2021; Wenger et al. 2021). 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. 2022). 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. (2022), 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 that it constitutes a positive step in this direction.

## 1.2 Contributions and outline

The rest of this paper is structured as follows. In Sect. 2, we introduce parareal, providing an overview of the algorithm and its computational complexity for a scalar ODE. In Sect. 3, we present our algorithm, henceforth referred to as

GParareal, in which we describe how a GP emulator, conditioned on acquisition data obtained in parallel throughout the simulation, is used to refine coarse numerical solutions to a scalar ODE. In addition, we detail the computational complexity of GParareal, provide a bound for its numerical error at a given iteration, and describe the extension for solving systems of ODEs. Numerical experiments are performed using HPC facilities in Sect. 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 legacy simulation data can be used to obtain solutions even faster, retaining comparable numerical accuracy. We discuss the benefits, drawbacks, and open questions surrounding GParareal in Sect. 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 autonomous ODE, i.e.  $f(t, u(t)) := f(u(t))$  in (1.1), without loss of generality.

### 2.1 The solvers

To calculate a solution to (1.1), parareal uses two one-step<sup>1</sup> numerical integrators. The first, referred to as the *fine solver*  $\mathcal{F}_{\Delta T}$ , is a computationally expensive integrator that propagates an initial value at  $t_j$ , over an interval of length  $\Delta T$ , and returns a solution with high numerical accuracy at  $t_{j+1}$ . In this paper, we assume that  $\mathcal{F}_{\Delta T}$  provides sufficient numerical accuracy to solve (1.1) for the solution to be considered ‘exact’, i.e.  $U_j = u(t_j)$ . The objective is to calculate the exact solutions

$$U_j = \mathcal{F}_{\Delta T}(U_{j-1}), \quad j = 1, \dots, J, \tag{2.1}$$

where  $U_0 := u_0$ , without running  $\mathcal{F}_{\Delta T}$   $J$  times sequentially, as this calculation is assumed to be computationally intractable. To avoid this, parareal locates iteratively

<sup>1</sup> 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 Sect. 3.

improved approximations  $U_j^k$ , where  $k = 0, 1, 2, \dots$  is the iteration number, that converge toward  $U_j$  (note that  $U_0^k = U_0 = u_0 \forall k \geq 0$ ). To do this, parareal uses a second numerical integrator  $\mathcal{G}_{\Delta T}$ , referred to as the *coarse solver*.  $\mathcal{G}_{\Delta T}$  propagates an initial value at  $t_j$  over an interval of length  $\Delta T$ , however, it has lower numerical accuracy and is computationally inexpensive to run compared to  $\mathcal{F}_{\Delta T}$ . This means that  $\mathcal{G}_{\Delta T}$  can be run serially across a number of time slices to provide relatively cheap low accuracy solutions whilst  $\mathcal{F}_{\Delta T}$  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  $\mathcal{G}_{\Delta T}$ , on a single processor, such that

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

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

$$U_j^k = \underbrace{\mathcal{G}_{\Delta T}(U_{j-1}^k)}_{\text{predict}} + \underbrace{\mathcal{F}_{\Delta T}(U_{j-1}^{k-1}) - \mathcal{G}_{\Delta T}(U_{j-1}^{k-1})}_{\text{correct}}, \tag{2.3}$$

for  $j = 1, \dots, J$ . Here,  $\mathcal{G}_{\Delta T}$  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 calculated 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). For a pre-defined tolerance  $\varepsilon > 0$ , the parareal solution  $U_j^k$  is deemed to have converged up to time  $t_I$  if

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

This criterion is standard for parareal (Garrido et al. 2006; Gander and Hairer 2008), however, other criteria, e.g. 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 updated in future iterations ( $k > 1$ ) by initiating further parallel  $\mathcal{F}_{\Delta T}$  runs on each  $U_j^k$ , followed by an update using (2.3). The algorithm stops once  $I = J$ , converging in  $k$  (out of  $J$ ) iterations. The version of parareal described here and implemented in Sect. 4 does not iterate over solutions that have already converged, avoiding the waste of computational resources (Elwasif et al. 2011; Pentland et al. 2022; Garrido et al. 2006). Extending

parareal to the full nonautonomous system in (1.1) is straightforward: see Gander and Vandewalle (2007) for notation and Pentland et al. (2022) for pseudocode.

### 2.3 Convergence and computational complexity

After  $k$  iterations, the solution states up to time  $t_k$  (at minimum) have converged, as the exact initial condition ( $u_0$ ) has been propagated by  $\mathcal{F}_{\Delta T}$  at least  $k$  times. Therefore, if parareal converges in  $k = J$  iterations, the solution will be equal to the one found by calculating (2.1) serially, at an even higher computational cost. Convergence<sup>2</sup> in  $k \ll J$  iterations is necessary if significant parallel speed-up is to be realised. Refer to Gander and Vandewalle (2007); Gander and Hairer (2008) for derivations of explicit parareal error bounds.

Without loss of generality, assume running  $\mathcal{F}_{\Delta T}$  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  $\mathcal{G}_{\Delta T}$ ). Therefore, calculating (2.1) using  $\mathcal{F}_{\Delta T}$  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

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

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  $\mathcal{G}_{\Delta T}$  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 Fig. 1—optimised scheduling of parareal is studied in Elwasif et al. (2011).

Selecting a fast but accurate coarse solver remains a trial and error process, entirely dependent on the system being solved. Typically,  $\mathcal{G}_{\Delta T}$  is chosen such that it has a coarser temporal resolution/lower numerical accuracy (Samaddar et al. 2010; Farhat and Chandesris 2003; Baffico et al. 2002; Trindade and Pereira 2006), a coarser spatial resolution (when solving PDEs) (Samaddar et al. 2019; Ruprecht

2014), and/or uses simplified model equations (Engblom 2009; Legoll et al. 2020; Meng et al. 2020) compared to  $\mathcal{F}_{\Delta T}$ . In Sect. 3, we aim to widen the pool of choices for  $\mathcal{G}_{\Delta T}$  by using a GP emulator to capture variability in the residual  $\mathcal{F}_{\Delta T} - \mathcal{G}_{\Delta T}$  and showcase its effectiveness by demonstrating that GParareal can converge to a solution in cases where parareal cannot in Sect. 4.

### 3 GParareal

In this section, we present the GParareal algorithm, in which a 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 from GParareal as  $V_j^k$  (as before,  $V_0^k = V_0 = u_0 \forall k \geq 0$ ).

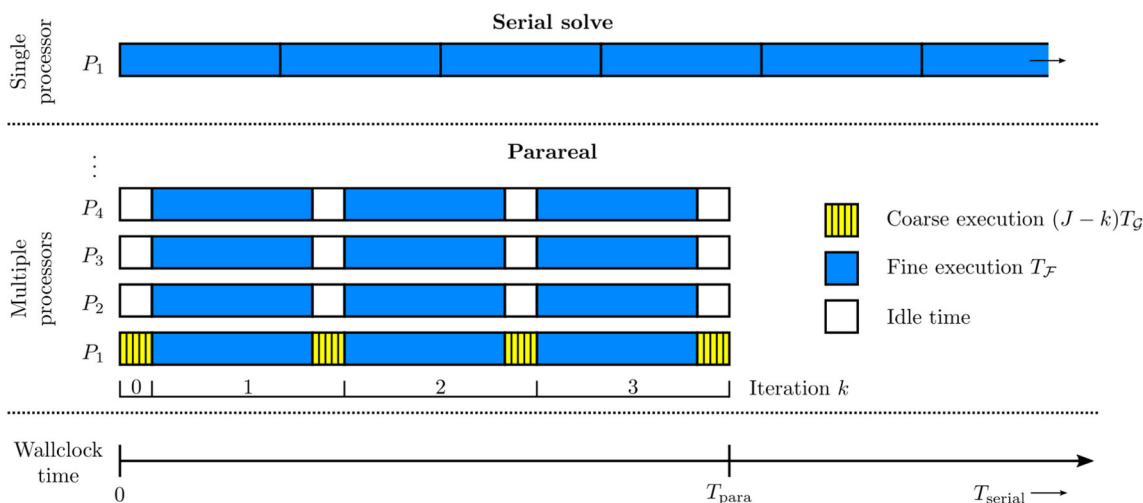
In parareal, the predictor-corrector (2.3) updates the numerical solutions at iteration  $k$  using a correction term based on information calculated during the *previous* iteration  $k - 1$ . We propose the following update rule, again based on a coarse prediction and multi-fidelity correction, that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

$$\begin{aligned}
 V_j^k &= \mathcal{F}_{\Delta T}(V_{j-1}^k) = (\mathcal{F}_{\Delta T} - \mathcal{G}_{\Delta T} + \mathcal{G}_{\Delta T})(V_{j-1}^k) \\
 &= \underbrace{(\mathcal{F}_{\Delta T} - \mathcal{G}_{\Delta T})(V_{j-1}^k)}_{\text{correction}} + \underbrace{\mathcal{G}_{\Delta T}(V_{j-1}^k)}_{\text{prediction}}, \tag{3.1}
 \end{aligned}$$

for  $1 \leq k < j \leq J$ . If  $V_{j-1}^k$  is known, the prediction 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 model this correction term, trained on *all* previously obtained evaluations of  $\mathcal{F}_{\Delta T}$  and  $\mathcal{G}_{\Delta T}$ . The emulator returns a Gaussian distribution over  $(\mathcal{F}_{\Delta T} - \mathcal{G}_{\Delta T})(V_{j-1}^k)$  from which we can extract an explicit value and carry out the refinement in (3.1).

In Sect. 3.1, we present the algorithm, giving an explanation of the kernel hyperparameter optimisation process in Sect. 3.2 and providing error analysis in Sect. 3.3. In Sect. 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 wall-clock time as parareal. Again, to simplify notation, we first detail GParareal for an autonomous scalar-valued ODE, i.e.  $f(t, \mathbf{u}(t)) := f(\mathbf{u}(t))$  in (1.1). The extension to the multivariate nonautonomous case is described in Sect. 3.5.

<sup>2</sup> For parareal to converge, the solvers  $\mathcal{F}_{\Delta T}$  and  $\mathcal{G}_{\Delta T}$  must satisfy specific mathematical conditions (Maday and Turinici 2005; Bal 2005).



**Fig. 1** 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  $T_{serial}$

### 3.1 The algorithm

#### Gaussian process emulator

Before solving (1.1), we define a GP prior (Rasmussen and Williams 2006) over the unknown correction function  $\mathcal{F}_{\Delta T} - \mathcal{G}_{\Delta T}$ . This function maps an initial value  $x_j \in \mathcal{U}$  at time  $t_j$  to the residual difference between  $\mathcal{F}_{\Delta T}(x_j)$  and  $\mathcal{G}_{\Delta T}(x_j)$  at time  $t_{j+1}$ . More formally, we define the GP prior

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

with mean function  $m: \mathcal{U} \rightarrow \mathbb{R}$  and covariance kernel  $\kappa: \mathcal{U} \times \mathcal{U} \rightarrow \mathbb{R}$ . Given some vectors of initial values,  $\mathbf{x}, \mathbf{x}' \in \mathcal{U}^J$ , the corresponding vector of means is denoted  $\mu(\mathbf{x}) = (m(x_j))_{j=0, \dots, J-1}$  and the covariance matrix  $K(\mathbf{x}, \mathbf{x}') = (\kappa(x_i, x'_j))_{i, j=0, \dots, J-1}$ . The correction term is expected to be small, depending on the accuracy of both  $\mathcal{F}_{\Delta T}$  and  $\mathcal{G}_{\Delta T}$ , 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

$$\kappa(x, x') = \sigma^2 \exp\left(-\frac{(x-x')^2}{2\ell^2}\right), \tag{3.3}$$

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

#### Iteration $k = 0$

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

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

Store these solutions in the vector  $\mathbf{x} := (V_0^0, \dots, V_{J-1}^0)^\top$  for use in the GP emulator.

#### Iteration $k = 1$

Use  $\mathcal{F}_{\Delta T}$  to propagate the values in (3.4) on each time slice in *parallel*, on  $J$  processors, to obtain the following values at  $t_j$

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

At this stage, we diverge from the parareal method. Given  $\mathbf{x}$ , store the values of  $\mathcal{F}_{\Delta T} - \mathcal{G}_{\Delta T}$ , using (3.4) and (3.5), in the vector

$$\mathbf{y} := ((\mathcal{F}_{\Delta T} - \mathcal{G}_{\Delta T})(x_j))_{j=0, \dots, J-1}. \tag{3.6}$$

At this point, the inputs  $\mathbf{x}$  and evaluations  $\mathbf{y}$  are used to optimise the kernel hyperparameters  $\theta$  via maximum likelihood estimation—see Sect. 3.2. Conditioning the prior (3.2) using the acquisition data  $\mathbf{x}$  and  $\mathbf{y}$ , 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

$$(\mathcal{F}_{\Delta T} - \mathcal{G}_{\Delta T})(x') \mid \mathbf{x}, \mathbf{y} \sim \mathcal{N}(\hat{\mu}(x'), \hat{K}(x', x')), \tag{3.7}$$

with mean

$$\hat{\mu}(x') = \underbrace{\mu(x')}_{=0} + K(x', \mathbf{x})[K(\mathbf{x}, \mathbf{x})]^{-1}(\mathbf{y} - \underbrace{\mu(\mathbf{x})}_{=0}) \quad (3.8)$$

and variance

$$\hat{K}(x', x') = K(x', x') - K(x', \mathbf{x})[K(\mathbf{x}, \mathbf{x})]^{-1}K(\mathbf{x}, x'). \quad (3.9)$$

Now we wish to determine updated solutions  $V_j^1$  at each mesh point. Given  $\mathcal{F}_{\Delta T}$  has been run once, the exact solution is known at time  $t_1$ . Specifically, at  $t_0$  we know  $V_0^k = V_0 \forall k \geq 0$  and at  $t_1$  we know  $V_1^k = V_1 = \mathcal{F}_{\Delta T}(V_0^k) \forall k \geq 1$ . At  $t_2$ , the exact solution  $V_2 = \mathcal{F}_{\Delta T}(V_1^1)$  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 (3.1):

$$\begin{aligned} V_2^1 &= \mathcal{F}_{\Delta T}(V_1^1) = (\mathcal{F}_{\Delta T} - \mathcal{G}_{\Delta T} + \mathcal{G}_{\Delta T})(V_1^1) \\ &= \underbrace{(\mathcal{F}_{\Delta T} - \mathcal{G}_{\Delta T})(V_1^1)}_{\text{correction}} + \underbrace{\mathcal{G}_{\Delta T}(V_1^1)}_{\text{prediction}}. \end{aligned} \quad (3.10)$$

Both terms in (3.10) 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

$$V_2^1 \sim \mathcal{N}(\hat{\mu}(V_1^1) + \mathcal{G}_{\Delta T}(V_1^1), \hat{K}(V_1^1, V_1^1)), \quad (3.11)$$

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  $\mathcal{G}_{\Delta T}$  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,

$$V_2^1 = \hat{\mu}(V_1^1) + \mathcal{G}_{\Delta T}(V_1^1).$$

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

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

$$V_j^1 = \hat{\mu}(V_{j-1}^1) + \mathcal{G}_{\Delta T}(V_{j-1}^1) \quad \text{for } j = 3, \dots, J. \quad (3.12)$$

This process is illustrated in Fig. 2. Finally, we impose stopping criteria (2.4), identifying which  $V_j^1$  for  $j \leq I$  have converged. Using the same stopping criteria as parareal will

allow us to compare the performance of both algorithms in Sect. 4.

### Iteration $k \geq 2$

If the stopping criteria is not met, i.e.  $I < J$ , we can iteratively update any unconverged solutions by re-applying the previous steps. This means calculating  $\mathcal{F}_{\Delta T}(V_j^{k-1})$ ,  $j = I, \dots, J - 1$ , in parallel and then storing new evaluations  $\hat{\mathbf{y}} = ((\mathcal{F}_{\Delta T} - \mathcal{G}_{\Delta T})(V_j^{k-1}))_{j=I, \dots, J-1}^\top$ , with corresponding inputs  $\hat{\mathbf{x}} = (V_I^{k-1}, \dots, V_{J-1}^{k-1})^\top$ . Hyperparameters are then re-optimised and the GP is re-conditioned using *all* prior acquisition data, i.e.  $\mathbf{x} = [\mathbf{x}; \hat{\mathbf{x}}]$  and  $\mathbf{y} = [\mathbf{y}; \hat{\mathbf{y}}]$ , generating an updated posterior. Here,  $[\mathbf{a}; \mathbf{b}]$  denotes the vertical concatenation of column vectors  $\mathbf{a}$  and  $\mathbf{b}$ . The update rule is then applied such that we obtain

$$V_j^k = \hat{\mu}(V_{j-1}^k) + \mathcal{G}_{\Delta T}(V_{j-1}^k) \quad \text{for } j = I + 2, \dots, J.$$

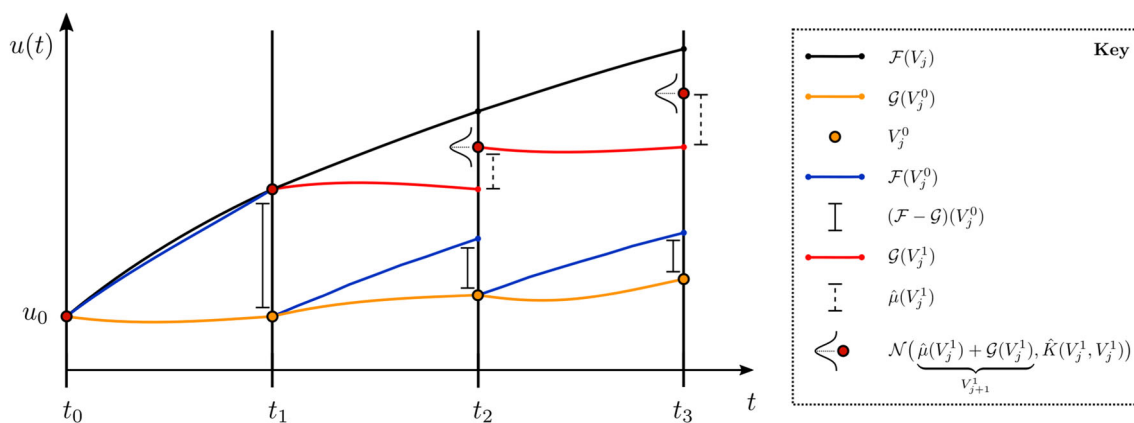
Once  $I = J$ , the solution, the number of iterations  $k$  taken to converge, and the acquisition data  $\mathbf{x}$  and  $\mathbf{y}$  are returned. A key advantage of GParareal is that the acquisition data can be used in future GParareal simulations (as “legacy data”) to provide the GP emulator with more data and therefore exploit additional speedup—this will be demonstrated in Sect. 4.

### 3.2 Kernel hyperparameter optimisation

The hyperparameters  $\theta$  of the kernel  $\kappa$  will need to be optimised in light of the acquisition data  $\mathbf{y}$  (and corresponding input data  $\mathbf{x}$ ). We optimise each element of  $\theta$  such that it maximises its (log) marginal likelihood (Rasmussen 2003). To do this, first define  $g(x) := (\mathcal{F}_{\Delta T} - \mathcal{G}_{\Delta T})(x)$  and  $\mathbf{g} := (g(x_j))_{j=0, \dots, N-1}$ , where  $N$  is the current length of  $\mathbf{x}$  (and  $\mathbf{y}$ ). Given the evaluations  $\mathbf{y}$  are noise-free, the likelihood of obtaining such data is  $p(\mathbf{y}|\mathbf{g}, \mathbf{x}, \theta) = \delta(\mathbf{y} - \mathbf{g})$ , where  $\delta(\cdot)$  is the multidimensional Dirac delta function. The marginal likelihood, given  $\mathbf{x}$  and  $\theta$ , is therefore

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

where  $\mathcal{N}(\mathbf{y}|\mathbf{0}, K(\mathbf{x}, \mathbf{x}))$  denotes the probability density function of a multivariate Gaussian distribution evaluated at  $\mathbf{y}$ , with mean vector  $\mathbf{0}$  and covariance matrix  $K(\mathbf{x}, \mathbf{x})$  that depends on  $\theta$ , see (3.3). The hyperparameters in  $\theta$  can then be estimated numerically by maximising the log marginal likelihood using any gradient-based optimiser. Optimisation is



**Fig. 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 (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 updated solutions (3.12) 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 made not to for clarity

carried out once per iteration (up until the hyperparameters do not change significantly between iterations) and hyperparameters from the prior iteration are used as to start the optimisation at the current iteration.

### 3.3 Error Analysis

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

$$e_j^k := |V_j - V_j^k|, \tag{3.13}$$

between the exact and GParareal solution at iteration  $k$  and time  $t_j$ . We show that this error has an upper bound proportional to the *fill distance* (defined below) of the dataset at iteration  $k$ . To do this, we now denote the input dataset at iteration  $k$  as  $\mathbf{x}_k$  rather than  $\mathbf{x}$  (because the dataset size strictly increases with each iteration of GParareal) and, similarly, denote the output dataset  $\mathbf{y}$  as  $\mathbf{y}_k$ . We also introduce some assumptions on the solvers  $\mathcal{F}_{\Delta T}$  and  $\mathcal{G}_{\Delta T}$ , and a known result on the consistency of the GP posterior mean  $\hat{\mu}$  (3.8) to the true correction function  $g = \mathcal{F}_{\Delta T} - \mathcal{G}_{\Delta T}$ . For clarity, we re-state the GParareal update rule

$$V_j^k = \mathcal{G}_{\Delta T}(V_{j-1}^k) + \hat{\mu}(V_{j-1}^k), \quad 1 \leq k < j \leq J. \tag{3.14}$$

#### 3.3.1 Preparatory assumptions and results

First, we state some assumptions on  $\mathcal{F}_{\Delta T}$  and  $\mathcal{G}_{\Delta T}$ , as in Gander and Hairer (2008).

**Assumption 3.1** [Exact fine solver  $\mathcal{F}_{\Delta T}$ ]  $\mathcal{F}_{\Delta T}$  solves (1.1) exactly such that

$$V_j = \mathcal{F}_{\Delta T}(V_{j-1}), \quad j = 1, \dots, J. \tag{3.15}$$

**Assumption 3.2** [One-step coarse solver  $\mathcal{G}_{\Delta T}$ ]  $\mathcal{G}_{\Delta T}$  is a one-step numerical solver with uniform local truncation error  $\mathcal{O}(\Delta T^{p+1})$ , for  $p \geq 1$ , such that

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

for  $u \in \mathbb{R}$  and continuously differentiable functions  $c_i(u)$ ,  $i = 1, 2, \dots$ . For  $u, v \in \mathbb{R}$ , we can then write

$$|(\mathcal{F}_{\Delta T}(u) - \mathcal{G}_{\Delta T}(u)) - (\mathcal{F}_{\Delta T}(v) - \mathcal{G}_{\Delta T}(v))| \leq C_1 \Delta T^{p+1} |u - v|, \tag{3.16}$$

where  $C_1 > 0$  is the Lipschitz constant for  $c_1(u)$ .

**Assumption 3.3** [Lipschitz coarse solver  $\mathcal{G}_{\Delta T}$ ]  $\mathcal{G}_{\Delta T}$  satisfies the Lipschitz condition

$$|\mathcal{G}_{\Delta T}(u) - \mathcal{G}_{\Delta T}(v)| \leq L_G |u - v|, \tag{3.17}$$

for  $u, v \in \mathbb{R}$  and some  $L_G > 0$ .

Next, we define the concepts required to state a result on the consistency of the GP posterior mean. Firstly, we define the *fill distance*  $h_{\mathbf{x}_k}$  to be the largest smallest distance between any point  $v \in \mathcal{U}$  and any point  $v_i \in \mathbf{x}_k$ , i.e.

$$h_{\mathbf{x}_k} := \sup_{v \in \mathcal{U}} \inf_{v_i \in \mathbf{x}_k} |v - v_i|.$$

It should be clear that each data point  $v_i \in \mathbf{x}_k$  is also contained in  $\mathcal{U}$ . Secondly, we define the *reproducing kernel*

*Hilbert space (RKHS)*, a Hilbert space  $H_\kappa(\mathcal{U})$  of functions  $g: \mathcal{U} \rightarrow \mathbb{R}$  with inner product  $\langle \cdot, \cdot \rangle_{H_\kappa(\mathcal{U})}$ . See Stuart and Teckentrup (2018) for a more formal definition and conditions on the inner product itself. We can now state the following result on the GP posterior mean consistency, adapted from Wendland (2004, Theorem 11.14).

**Theorem 3.4** (*GP posterior mean consistency*) *Suppose  $\mathcal{U} \subset \mathbb{R}$  is a bounded interval and let  $\kappa$  be the SE kernel. Denote the GP posterior mean, built using  $\mathbf{x}_k, \mathbf{y}_k$ , and  $\kappa$  (3.8) 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*

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

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

See Wendland (2004, Theorem 11.14) for a more general version of this result that holds when  $\mathcal{U} \subset \mathbb{R}^d$  and for derivatives of both  $g$  and  $\hat{\mu}$ . It should be noted that Theorem 3.4 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. For consistency results involving Matérn kernels, see Stuart and Teckentrup (2018).

### 3.3.2 Error bound for GParareal solutions

**Theorem 3.5** (*GParareal error bound*) *Suppose the solvers used in GParareal satisfy Assumptions 3.1, 3.2, and 3.3, and that the conditions required for Theorem 3.4 hold. Then, the absolute error (3.13) of the GParareal solution to the autonomous scalar-valued ODE, i.e.  $\mathbf{f}(t, \mathbf{u}(t)) := f(u(t))$  in (1.1), at iteration  $k$  and time  $t_j$  satisfies*

$$e_j^k \leq \begin{cases} \Lambda_k \sum_{i=0}^{j-(k+1)} A^i & 1 \leq k < j \leq J, \\ 0 & 0 \leq j \leq k \leq J. \end{cases}$$

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

**Proof** First, consider the case  $0 \leq j \leq k \leq J$ . For  $j = 0$ , recall that  $V_0^k = V_0 \forall k \geq 0$  by definition, hence  $e_0^k = 0 \forall k \geq 0$ . For  $j = 1$ , we seek  $V_1^1 = \mathcal{F}_{\Delta T}(V_0^1)$  which we in fact know from applying  $\mathcal{F}_{\Delta T}$  to  $V_0^0$  during the prior iteration (i.e.  $k = 0$ ). Therefore, we have that

$$\begin{aligned} V_1^1 &= \mathcal{F}_{\Delta T}(V_0^1) = \mathcal{F}_{\Delta T}(V_0^0) = \mathcal{F}_{\Delta T}(V_0) = V_1 \\ &\Rightarrow V_1^k = V_1 \quad \forall k \geq 1 \\ &\Rightarrow e_1^k = 0 \quad \forall k \geq 1. \end{aligned}$$

We can repeat this process iteratively up to  $j = J$  to show that

$$\begin{aligned} V_j^J &= \mathcal{F}_{\Delta T}(V_{j-1}^J) = \mathcal{F}_{\Delta T}(V_{j-1}^{J-1}) = \mathcal{F}_{\Delta T}(V_{j-1}) = V_j \\ &\Rightarrow V_j^k = V_j \quad \forall k \geq J \\ &\Rightarrow e_j^k = 0 \quad \forall k \geq J. \end{aligned}$$

Now, consider the case  $1 \leq k < j \leq J$ . Using the update rule (3.14), that  $\mathcal{F}_{\Delta T}$  is the exact solver (3.15), and adding and subtracting the terms  $g(V_j^k)$  and  $\mathcal{G}_{\Delta T}(V_j)$ , we can write

$$\begin{aligned} e_{j+1}^k &= |V_{j+1} - V_{j+1}^k| = |\mathcal{F}_{\Delta T}(V_j) - (\mathcal{G}_{\Delta T}(V_j^k) + \hat{\mu}(V_j^k))| \\ &= |\mathcal{F}_{\Delta T}(V_j) - (\mathcal{G}_{\Delta T}(V_j^k) + \hat{\mu}(V_j^k)) \pm g(V_j^k) \pm \mathcal{G}_{\Delta T}(V_j)|. \end{aligned}$$

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

$$\begin{aligned} e_{j+1}^k &\leq |(\mathcal{F}_{\Delta T}(V_j) - \mathcal{G}_{\Delta T}(V_j))| \\ &\quad - |(\mathcal{F}_{\Delta T}(V_j^k) - \mathcal{G}_{\Delta T}(V_j^k))| \\ &\quad + |\mathcal{G}_{\Delta T}(V_j) - \mathcal{G}_{\Delta T}(V_j^k)| + |g(V_j^k) - \hat{\mu}(V_j^k)|. \end{aligned}$$

On the right hand side, the first term can be bounded using (3.16), the second by (3.17), and the third using Theorem 3.4, yielding the recursion

$$e_{j+1}^k \leq A e_j^k + \Lambda_k,$$

where  $A = C_1 \Delta T^{p+1} + L_G$  and  $\Lambda_k = C_\tau h_{x_k}^\tau |g|_{H_\kappa(\mathcal{U})}$ . This recursion can be solved using the initial condition  $e_j^k = 0 \forall k \geq j$  to obtain the desired result.  $\square$

Theorem 3.5 shows that the error is proportional to the fill distance at iteration  $k$  and that GParareal will recover the exact solution at time  $t_j$  after  $k = j$  iterations.

Note that 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 would 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 GParareal scheme because the mean will never be evaluated outside of  $\mathcal{V}$ . Also note, the result will generalise for GParareal applied to systems of ODEs by using norms and the generalised version of Theorem 3.4 in Wendland (2004).

### 3.4 Computational complexity

The complexity of GParareal can be calculated similarly to that of parareal—refer back to Sect. 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  $k$ th iteration, up to  $kJ$  evaluations of  $\mathcal{F}_{\Delta T} - \mathcal{G}_{\Delta T}$  have been collected, hence standard cubic complexity GP conditioning scales like  $\mathcal{O}(k^3 J^3)$  in terms of floating point operations (and  $\mathcal{O}(k^2 J^2)$  per hyperparameter). Given a fixed number of time slices  $J$ , let  $T_{GP}(k)$  represent the total wallclock time taken to condition and optimise hyperparameters of the GP (using up to  $kJ$  observations) at iteration  $k$ —note this is a strictly increasing function of  $k$ . Ignoring serial overheads, we can write down the total wallclock time for GParareal as

$$T_{GPara} \approx JT_G + \sum_{i=1}^k (T_{\mathcal{F}} + (J - i)T_G + T_{GP}(i))$$

$$= kT_{\mathcal{F}} + (k + 1) \left( J - \frac{k}{2} \right) T_G + T_{GP}, \quad (3.18)$$

where  $T_{GP} := \sum_{i=1}^k T_{GP}(i)$ . The approximate parallel speed-up is then given by

$$S_{GPara} \approx \left[ \frac{k}{J} + (k + 1) \left( 1 - \frac{k}{2J} \right) \frac{T_G}{T_{\mathcal{F}}} + \frac{1}{J} \frac{T_{GP}}{T_{\mathcal{F}}} \right]^{-1}. \quad (3.19)$$

Therefore, in addition to the parareal requirements that  $k$  be as small as possible and  $T_G \ll T_{\mathcal{F}}$ , GParareal requires that  $T_{GP} \ll T_{\mathcal{F}}$  in order to maximise parallel speedup. If this is the case, the complexity of GParareal is approximately the same as parareal.

This suggests that if  $k$  and/or  $J$  are large, then the cost of the emulation may begin to dominate that of the fine solver, limiting the parallel speedup from GParareal, see Sect. 4. 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. spatio-temporal PDEs. This means that if  $P$  processors are required for the space-parallel computations of the PDE and  $J$  processors for the time-parallel computations, then  $J P$  processors are required in total. For moderate to large values of  $P$ , only leftover HPC resources are available to exploit time-parallelism and so  $J$  typically cannot be chosen very large, somewhat limiting how large  $T_{GP}$  can 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  $J$ , thereby increasing  $T_{\mathcal{F}}$  at the same time.

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 complex-

ity, 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  $K(\mathbf{x}, \mathbf{x})^{-1}$ . 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, often significant, cost of hyperparameter optimisation, one may deploy parallel optimisation routines if available or, as we implement in Sect. 4, stop the optimisation once additional data no longer improves the hyperparameter estimates.

### 3.5 Generalisation to ODE systems

The methodology in Sect. 3.1 can be generalised to solve systems of  $d$  autonomous ODEs. Accordingly, the correction term we wish to emulate is now 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  $\kappa_i$  (usually the same for consistency) and corresponding hyperparameters  $\theta_i$ —to be optimised independently using their own respective observation datasets  $\mathbf{y}^{(i)}$ ,  $i = 1, \dots, d$ . In this case, the  $d$  GP emulators can be conditioned/optimised independently of one another and so we make use of the idle processors to carry out these computations in an embarrassingly parallel fashion to reduce the total GP complexity from  $\mathcal{O}(dk^3 J^3)$  to  $\mathcal{O}(k^3 J^3)$  each iteration—the same as the scalar case.

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 using a linear combination of the separate kernels  $\kappa_i$ . Prior testing revealed that using this method did not improve performance enough to justify the added complexity,  $\mathcal{O}(d^3 k^3 J^3)$  vs.  $\mathcal{O}(dk^3 J^3)$  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 nonautonomous 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 covari-

ance kernel. The other way is to re-write the  $d$ -dimensional nonautonomous system as a system of  $d + 1$  autonomous equations and solve as described above—this is the method we use in Sect. 4.

## 4 Numerical experiments

In this section, we present numerical experiments to compare the performance of GParareal and parareal on a number of low-dimensional ODE systems, namely the FitzHugh–Nagumo model, the chaotic Rössler system, a nonautonomous system, and the double pendulum system. MATLAB code for GParareal, parareal, and the GP emulator as used in the experiments of this section can be found at <https://github.com/kpentland/GParareal>.

For simplicity,  $\mathcal{F}_{\Delta T}$  and  $\mathcal{G}_{\Delta T}$  are chosen to be explicit Runge–Kutta methods (RK) of order  $q$ ,  $p \in \{1, 2, 4, 8\}$ , respectively ( $q \geq p$ ). Let  $N_{\mathcal{F}}$  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 Sect. 3.5), 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, reducing the GP emulation costs by a factor of  $d$ . Hyperparameter optimisation is carried out at each iteration, stopping when the (maximal) absolute difference between hyperparameters is larger than  $10^{-2}$ . The experiments are run on up 512 CPUs.

### 4.1 FitzHugh–Nagumo model

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

$$\frac{du_1}{dt} = c\left(u_1 - \frac{u_1^3}{3} + u_2\right), \quad \frac{du_2}{dt} = -\frac{1}{c}(u_1 - a + bu_2), \quad (4.1)$$

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  $\mathcal{G}_{\Delta T} = \text{RK2}$  and  $\mathcal{F}_{\Delta T} = \text{RK4}$  with  $N_{\mathcal{G}} = 160$  and  $N_{\mathcal{F}} = 1.6 \times 10^5$  steps respectively.

In Fig. 3a, we solve (4.1) with initial condition  $\mathbf{u}_0 = (-1, 1)^\top$  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 (Fig. 3b). Note, however, that

in Fig. 3c, 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 Fig. 3d, with an almost 5-fold speedup vs. the serial solver—over twice the 2.4-fold speedup obtained by parareal. Note that we increase  $N_{\mathcal{F}}$  to  $1.6 \times 10^8$  to ensure  $\mathcal{F}_{\Delta T}$  is expensive to run and realise parallel speedup in Equation (4.1)(d) (as both algorithms require  $T_{\mathcal{G}}/T_{\mathcal{F}} \ll 1$ ).

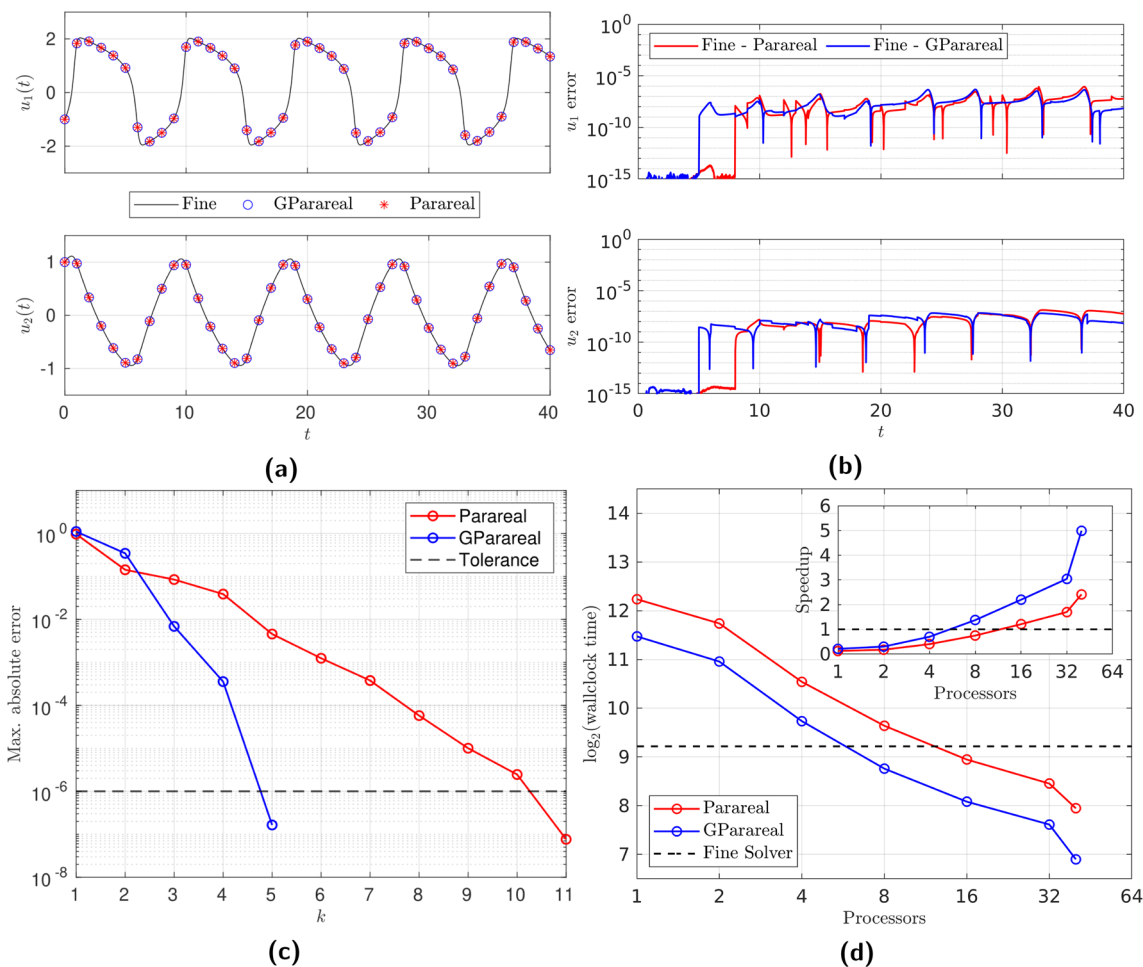
To compare the convergence of both methods more broadly, we solve (4.1) for a range of initial values. The heatmap in Fig. 4a illustrates how the convergence of parareal is highly dependent, not just on the solvers in use, but also the initial values at  $t = 0$ , taking anywhere from 10 to 15 iterations 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  $\mathcal{G}_{\Delta T}$ . In direct contrast, see Fig. 4b, 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  $\mathcal{G}_{\Delta T}$  has poor accuracy.

Until now, GParareal simulations have been carried out using only acquisition data. In Fig. 5, 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  $\mathbf{u}_0 = (-1, 1)^\top$ , are stored and made available to the GP emulator when solving (4.1) for alternate initial values  $\mathbf{u}_0 = (0.75, 0.25)^\top$ . In Fig. 5a, 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 Fig. 5b. Repeating the experiment from Fig. 4b with the same legacy data for a range of initial values we see that  $k$  is either unchanged or improved in all cases, see Fig. 6. 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 that  $T_{\mathcal{F}} \gg T_{\text{GP}}$ . These results illustrate 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.

### 4.2 Rössler system

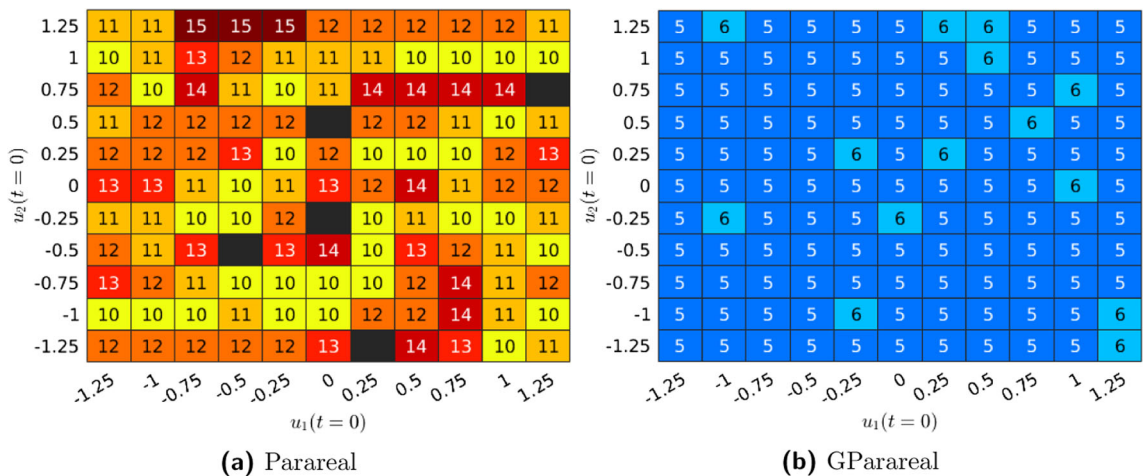
Next we solve the Rössler system,

$$\begin{aligned} \frac{du_1}{dt} &= -u_2 - u_3, \\ \frac{du_2}{dt} &= u_1 + \hat{a}u_2, \quad \frac{du_3}{dt} = \hat{b} + u_3(u_1 - \hat{c}), \end{aligned} \quad (4.2)$$

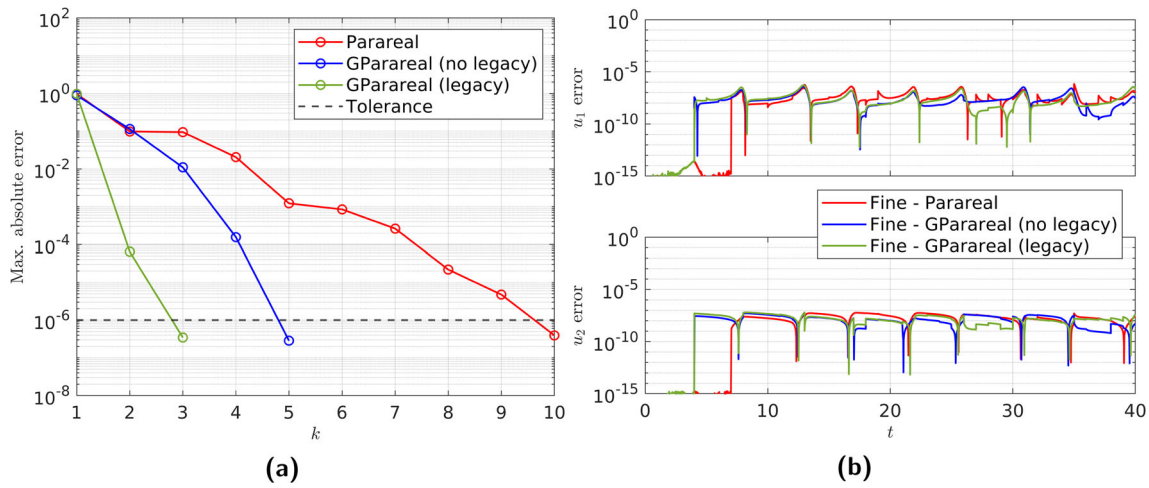


**Fig. 3** Numerical results obtained solving the FHN model (4.1) for  $u_0 = (-1, 1)^T$ . **a** Time-dependent solutions using the fine solver, GParareal, and parareal—both GParareal and parareal plotted only at times  $t$  for clarity. **b** The corresponding absolute errors between solutions from GParareal and parareal vs. the fine solution. **c** Maximum

absolute errors (2.4) 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

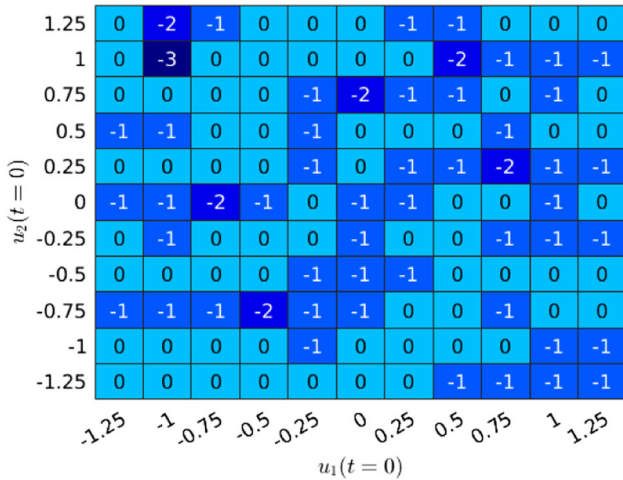


**Fig. 4** Heat maps displaying the number of iterations taken until convergence  $k$  of **a** parareal and **b** GParareal when solving the FHN model (4.1) for different initial values  $u_0 \in [-1.25, 1.25]^2$ . Black boxes indicate where parareal returned a NaN value during simulation



**Fig. 5** Numerical simulations solving (4.1) for  $\mathbf{u}(0) = (0.75, 0.25)^\top$  using GParareal with and without access to legacy data, i.e.  $\mathcal{F}_{\Delta T} - \mathcal{G}_{\Delta T}$  data obtained solving (4.1) for  $\mathbf{u}(0) = (-1, 1)^\top$ . The parareal simulation of the same problem is also shown for comparison. **a** Maximum

absolute errors (2.4) 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



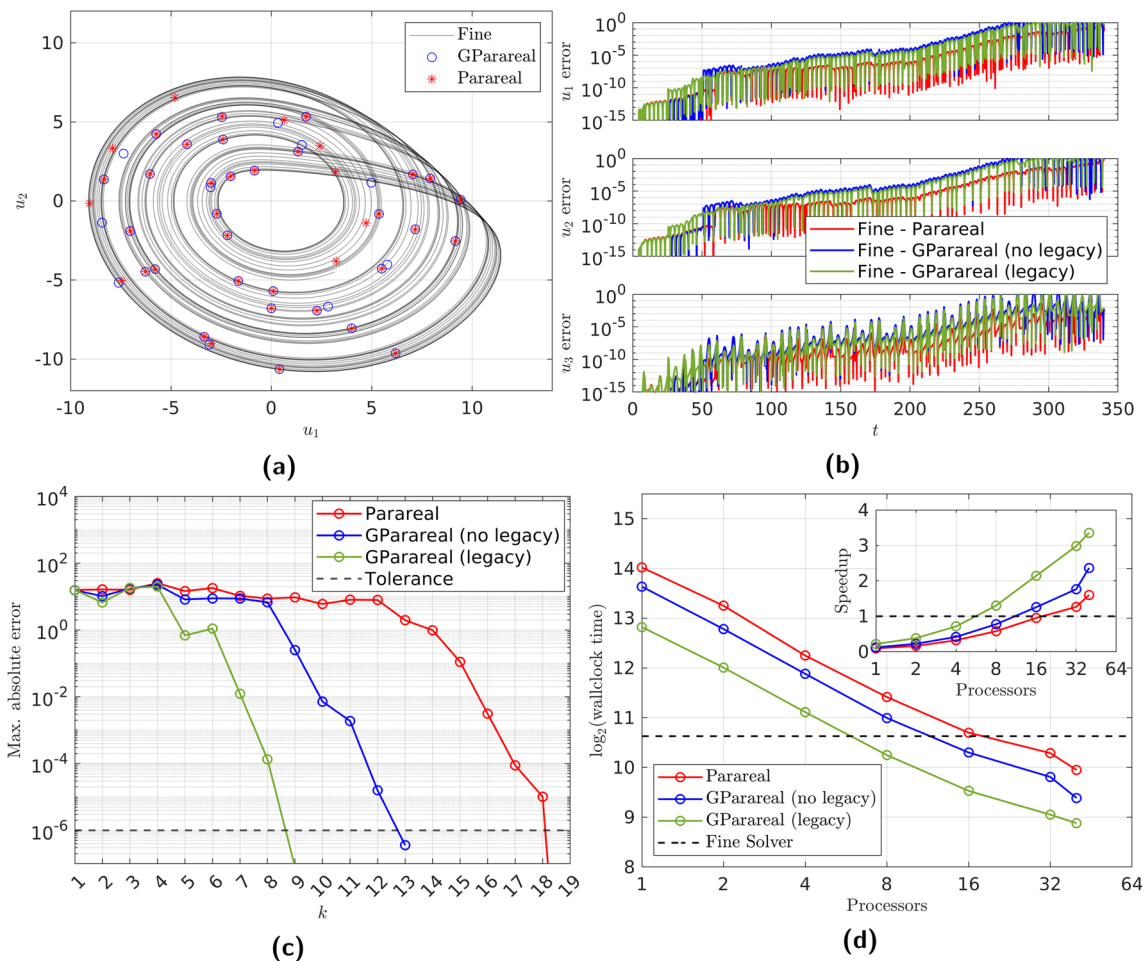
**Fig. 6** Heat map displaying the decrease in the number of iterations taken until convergence of GParareal when solving (4.1) for different initial values  $\mathbf{u}_0 \in [-1.25, 1.25]^2$  with legacy data compared to without, i.e. compared to Fig. 4b. Legacy data was obtained by solving (4.1) for  $\mathbf{u}_0 = (-1, 1)^\top$

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 (4.2) over  $t \in [0, 340]$  with initial values  $\mathbf{u}_0 = (0, -6.78, 0.02)^\top$  and solvers  $\mathcal{G}_{\Delta T} = \text{RK1}$  and  $\mathcal{F}_{\Delta T} = \text{RK4}$ . The interval is divided into  $J = 40$  time slices,  $N_G = 9 \times 10^4$  coarse steps, and  $N_F = 4.5 \times 10^8$  fine steps. The tolerance is set to  $\varepsilon = 10^{-6}$ .

In this experiment, rather than obtaining legacy data by solving (4.2) using alternative initial values (as we did in Sect. 4.1), 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, 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 (Legoll et al. 2021; Grafke et al. 2017). 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  $J$ ,  $N_G$ , and  $N_F$  by a factor of two, i.e. use  $J^{(2)} = J/2$ ,  $N_G^{(2)} = N_G/2$ , and  $N_F^{(2)} = N_F/2$  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  $\mathcal{F}_{\Delta T} - \mathcal{G}_{\Delta T}$  (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 Fig. 7c. In Fig. 7d we can see that using the legacy data achieves a higher numerical speedup ( $3.4\times$ ) compared to parareal ( $1.6\times$ ). In Fig. 7a we see the trajectories from each simulation converging toward the Rössler attractor and Fig. 7b illustrates GParareal retaining a similar numerical accuracy to parareal with and without the legacy data. Note the



**Fig. 7** Numerical results obtained solving the Rössler system (4.2) 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  $t$  for clarity. **b** The corresponding absolute errors between solutions from GParareal and parareal vs. the fine solu-

tion. **c** Maximum absolute errors (2.4) 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

steadily increasing errors for both algorithms is due to the chaotic nature of the Rössler system.

### 4.3 Nonautonomous system

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

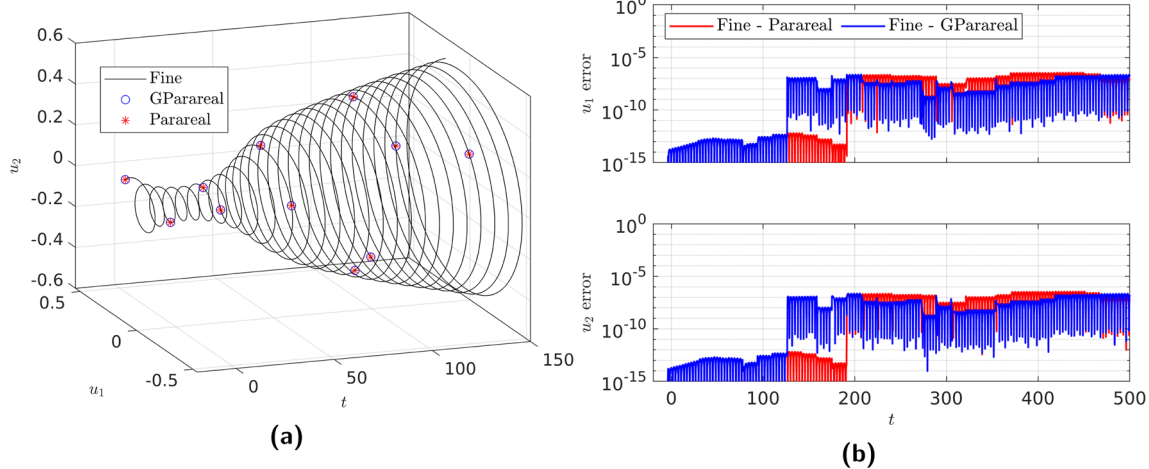
$$\begin{aligned} \frac{du_1}{dt} &= -u_2 + u_1 \left( \frac{t}{500} - u_1^2 - u_2^2 \right), \\ \frac{du_2}{dt} &= u_1 + u_2 \left( \frac{t}{500} - u_1^2 - u_2^2 \right), \end{aligned} \quad (4.3)$$

over  $t \in [-20, 500]$ —adapted from Trefethen et al. (2017). As described in Sect. 3.5, we transform this two-dimensional nonautonomous system into a three-dimensional autonomous system by introducing an additional variable  $u_3(t) = t$ ,

where  $du_3/dt = 1$ . 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 GParareal.

We select solvers  $\mathcal{G}_{\Delta T} = \text{RK1}$  and  $\mathcal{F}_{\Delta T} = \text{RK8}$  with  $N_G = 2048$  and  $N_F = 5.12 \times 10^5$  steps, respectively. We use  $J = 32$  time slices, initial condition  $\mathbf{u}_0 = (0.1, 0.1, -20)^\top$ , and a stopping tolerance of  $\varepsilon = 10^{-6}$ . In Fig. 8, we plot the 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 Fig. 8a, enabling the emulator to reproduce the dynamics of  $\mathcal{F}_{\Delta T} - \mathcal{G}_{\Delta T}$  quite well.

Next, we run a series of simulations to measure the effect of increasing the number of time slices  $J$  (and hence proces-



**Fig. 8** Numerical results obtained solving the nonautonomous system (4.3). **a** Time-dependent solutions using the fine solver, GParareal, and parareal—both GParareal and parareal plotted only at times  $t$  on

$[-20, 150]$  for clarity. **b** The corresponding absolute errors between solutions from GParareal and parareal vs. the fine solution, having converged after 10 and 20 iterations, respectively

**Table 1** Wallclock time and speedup results obtained solving the nonautonomous system (4.3) for  $J \in \{32, 64, 128, 256, 512\}$ , where  $k_{\text{para}}$  and  $k_{\text{GPara}}$  are the number of iterations taken for parareal and GParareal to converge, respectively. (Top) Numerical results obtained upon simulation. (Bottom) Theoretical results calculated using (2.5), (3.18), (2.6), and (3.19). All timings are measured in seconds

$J$	$k_{\text{para}}$	$k_{\text{GPara}}$	$T_{\mathcal{G}}$	$T_{\mathcal{F}}$	$T_{\text{GP}}$	$T_{\text{serial}}$	$T_{\text{para}}$	$T_{\text{GPara}}$	$S_{\text{para}}$	$S_{\text{GPara}}$
32	20	10	1.60E−4	4.23E3	5.37	1.35E5	8.92E4	4.33E4	1.52	3.13
64	31	14	9.80E−5	2.10E3	18.36	1.35E5	6.75E4	3.20E4	2.00	4.21
128	55	16	9.10E−5	1.06E3	2.26E2	1.35E5	6.47E4	1.90E4	2.09	7.13
256	99	18	6.90E−5	5.23E2	1.02E2	1.34E5	5.64E4	1.17E4	2.37	11.42
512	151	15	6.30E−5	2.62E2	1.09E4	1.34E5	4.42E4	2.10E4	3.03	6.39

$J$	$T_{\text{para}}$	$T_{\text{GPara}}$	$S_{\text{para}}$	$S_{\text{GPara}}$
32	8.47E4	4.23E4	1.60	3.20
64	6.52E4	2.95E4	2.06	4.57
128	5.81E4	1.71E4	2.33	7.89
256	5.17E4	9.51E3	2.59	14.07
512	3.95E4	1.49E4	3.39	9.02

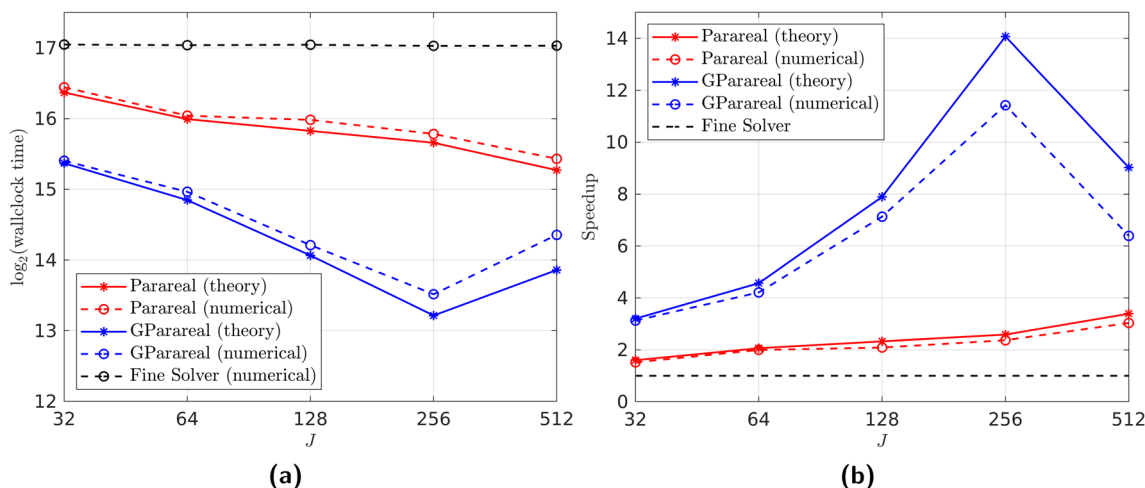
sors) on convergence, wallclock times, and speedup—see Table 1. To do this, we increase the number of fine time steps to  $N_{\mathcal{F}} = 5.12 \times 10^{10}$ , so that  $\mathcal{F}_{\Delta T}$  is sufficiently expensive to observe speedup. We observe a good match between the numerical and theoretical results, presented in the top and bottom tables of Table 1, respectively, and visualised graphically in Fig. 9. Firstly, notice that  $k_{\text{para}}$  increases with  $J$  whilst  $k_{\text{GPara}}$  remains largely unaffected, leading to speedups for GParareal being roughly  $2\times$  to  $4\times$  that of parareal, up to  $J = 256$ . For both algorithms, the cost of  $T_{\mathcal{G}}$  and  $T_{\mathcal{F}}$  decreases as  $J$  increases (due to fewer time steps per time slice), whilst  $T_{\text{GP}}$  increases (due to increasing numbers of data points each simulation). Note the exception of  $T_{\text{GP}} = 1.02\text{E}2$  when  $J = 256$  because hyperparameter optimisation converged within a few iterations and was therefore not carried out after this. Up to  $J = 256$ ,  $T_{\text{GP}} < T_{\mathcal{F}}$  and so we observe increasing parallel speedup for GParareal. When

$J = 512$ , the cost of the GP overtakes that of  $\mathcal{F}_{\Delta T}$  and so parallel speedup decreases, albeit still being double that of parareal. Recall that if  $T_{\text{GP}} > T_{\mathcal{F}}$ , we may not opt to use GParareal in the first place, for the reasons outlined in Sect. 3.4.

### 4.4 Double pendulum system

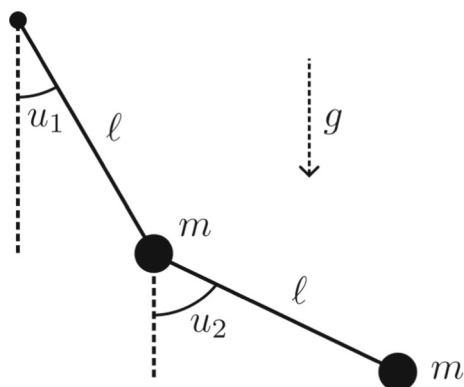
Consider now the double pendulum system: a simple pendulum of mass  $m$ , rod length  $\ell$ , connected to another simple pendulum of equal mass  $m$ , rod length  $\ell$ , acting under gravity  $g$  (see Fig. 10). Four ODEs govern the dynamics of this system:

$$\begin{aligned} \frac{du_1}{dt} &= u_3, \\ \frac{du_2}{dt} &= u_4, \end{aligned}$$



**Fig. 9** Numerical results obtained solving the nonautonomous system (4.3) for  $J \in \{32, 64, 128, 256, 512\}$ . **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, cal-

culated using (2.5) and (3.18) for parareal and GParareal, respectively. **b** The corresponding speedup results using the same lines and colours. The theoretical results were calculated using (2.6) and (3.19) for parareal and GParareal, respectively



**Fig. 10** A schematic of the double pendulum system

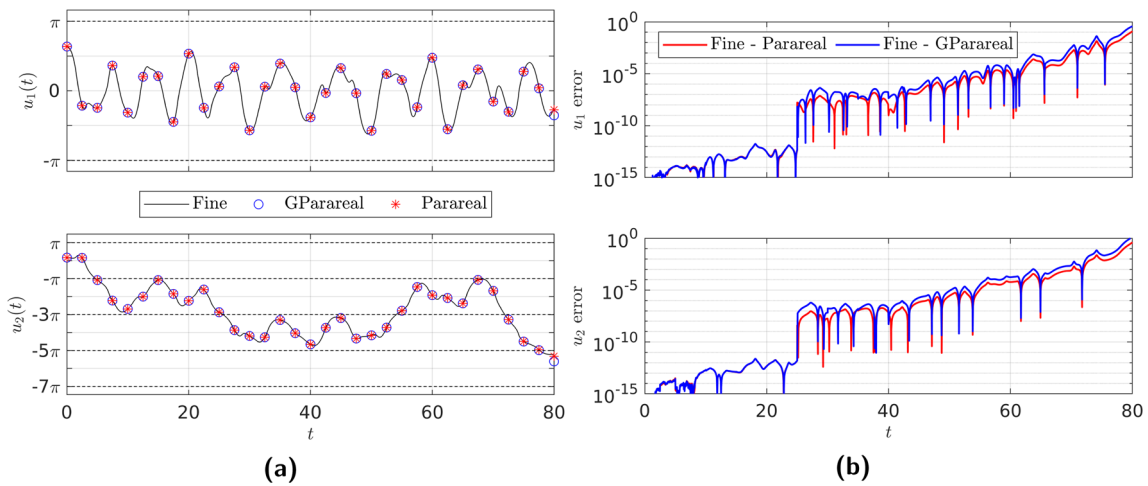
$$\begin{aligned} \frac{du_3}{dt} &= \frac{-u_3^2 f_1(u_1, u_2) - u_4^2 \sin(u_1 - u_2) - 2 \sin(u_1) + \cos(u_1 - u_2) \sin(u_2)}{f_2(u_1, u_2)}, \\ \frac{du_4}{dt} &= \frac{2u_3^2 \sin(u_1 - u_2) + u_4^2 f_1(u_1, u_2) + 2 \cos(u_1 - u_2) \sin(u_1) - 2 \sin(u_2)}{f_2(u_1, u_2)}, \end{aligned} \quad (4.4)$$

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 (4.4) by letting  $l = g$ . 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  $N_G = 3072$  and  $N_F = 2.1504 \times 10^5$  steps, respectively. We integrate over  $t \in [0, 80]$ , using  $J = 32$  time slices with a stopping tolerance  $\varepsilon = 10^{-6}$ .

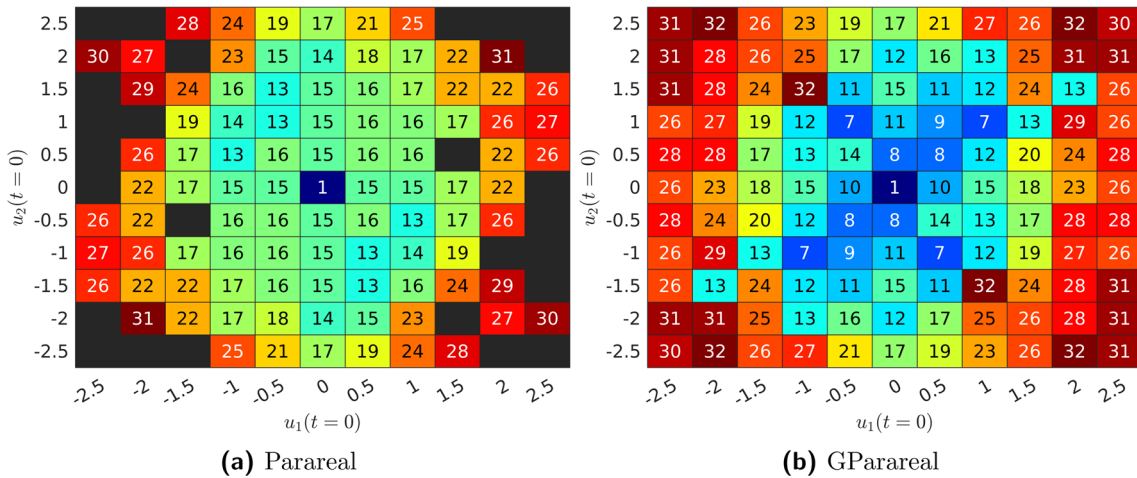
In Fig. 11, we plot solutions for  $u_1$  and  $u_2$  over time using initial conditions  $\mathbf{u}_0 = (2, 0.5, 0, 0)^T$ , i.e. the pendulums are positioned at some (positive) initial angles and released from rest. Observe how both pendulums move chaotically, with the inner pendulum oscillating within  $[-\pi, \pi]$  and the outer pendulum oscillating between odd multiples of  $\pi$ , “turning over” a number of times. We attain good solution accuracy from GParareal with respect to the fine solution with errors slowly increasing over time due to the chaotic nature of the system, much like what was seen in the Rössler experiments in Sect. 4.2. We plot  $k$  for various initial angles in Fig. 12 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).

In Table 2 and Fig. 13, we again test the effect of increasing  $J$  on wallclock times, speedup, and convergence. To do this, we increase the number of fine time steps to  $N_F = 2.1504 \times 10^{10}$ . We purposefully choose an initial condition ( $\mathbf{u}_0$  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 GParareal for large  $J$ . Under these circumstances, we can think of the wallclock time for GParareal as (approximately) the wallclock time of parareal plus the wallclock time of the GP conditioning/optimisation. For  $J \leq 128$ , we observe that  $T_{GP} < T_F$  and so the speedup of GParareal and parareal are approximately the same. In these cases, using GParareal 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  $J \geq 256$ , we begin to observe  $T_{GP} \approx T_F$



**Fig. 11** Numerical results obtained solving the double pendulum system (4.4). **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  $t$  for clarity. Dashed lines indicate “turning over” angles,

at which either pendulum passes through an odd multiple of  $\pi$ . **b** The corresponding absolute errors between solutions from GParareal and parareal vs. the fine solution, having converged after 23 and 22 iterations, respectively



**Fig. 12** Heat maps displaying the number of iterations taken until convergence  $k$  of **a** parareal and **b** GParareal when solving the double pendulum system (4.4) for different initial angles  $(u_1(0), u_2(0)) \in$

$[-2.5, 2.5]^2$  and initial angular velocities  $(u_3(0), u_4(0)) = (0, 0)$ , i.e. the pendulums are released from rest. Black boxes indicate where parareal returned a NaN value during simulation

(or larger), so the numerical speedup of GParareal begins to plateau. We should reiterate, however, that using so many processors for such a small test problem is quite excessive.

### 5 Discussion

In this paper, we present a 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 ( $\mathcal{G}_{\Delta T}$ ) and fine ( $\mathcal{F}_{\Delta T}$ ) integrators. However, unlike the classical parareal algorithm, it uses a Gaussian process (GP) emulator to infer the

correction term  $\mathcal{F}_{\Delta T} - \mathcal{G}_{\Delta T}$ . The numerical experiments reported in Sect. 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  $\mathcal{F}_{\Delta T}$  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—something that existing time-parallel methods cannot do.

In Sect. 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-

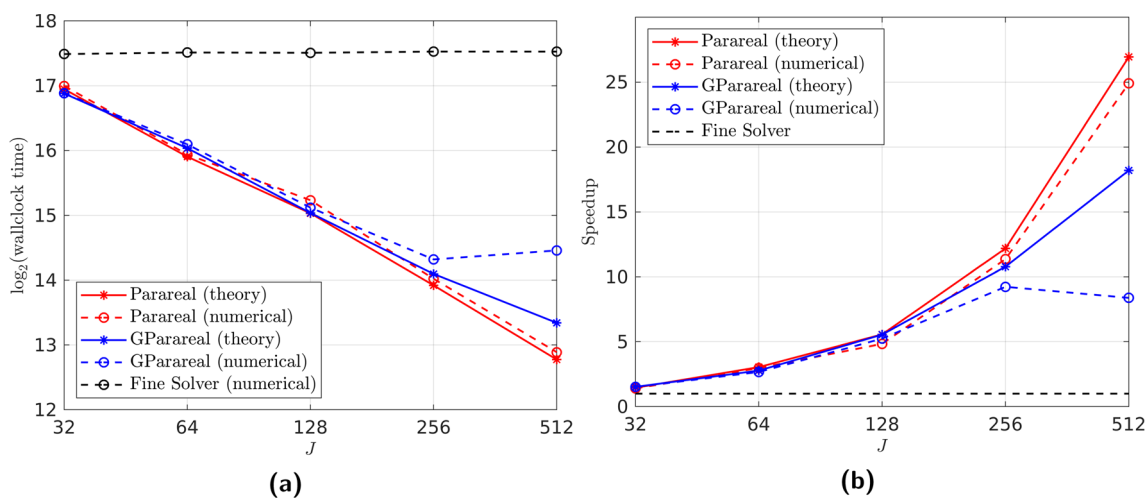


**Table 2** Wallclock time and speedup results obtained solving the double pendulum system (4.4) for  $J \in \{32, 64, 128, 256, 512\}$ , where  $k_{\text{para}}$  and  $k_{\text{GPara}}$  are the number of iterations taken for parareal and GParareal to converge, respectively. (Top) Numerical results obtained upon simulation. (Bottom) Theoretical results calculated using (2.5), (3.18), (2.6), and (3.19). All timings are measured in seconds

$J$	$k_{\text{para}}$	$k_{\text{GPara}}$	$T_{\mathcal{G}}$	$T_{\mathcal{F}}$	$T_{\text{GP}}$	$T_{\text{serial}}$	$T_{\text{para}}$	$T_{\text{GPara}}$	$S_{\text{para}}$	$S_{\text{GPara}}$
32	22	21	2.53E-4	5.75E3	19.75	1.84E5	1.31E5	1.21E5	1.41	1.52
64	21	23	1.46E-4	2.93E3	17.66	1.87E5	6.29E4	7.00E4	2.97	2.67
128	23	23	1.27E-4	1.46E3	1.20E2	1.86E5	3.85E4	3.56E4	4.84	5.24
256	21	23	9.10E-5	7.35E2	5.45E2	1.89E5	1.66E4	2.04E4	11.36	9.24
512	19	22	7.00E-5	3.69E2	2.26E3	1.89E5	7.58E3	2.25E4	24.90	8.38

$J$	$T_{\text{para}}$	$T_{\text{GPara}}$	$S_{\text{para}}$	$S_{\text{GPara}}$
32	1.26E5	1.21E5	1.45	1.52
64	6.14E4	6.72E4	3.05	2.78
128	3.35E4	3.36E4	5.57	5.54
256	1.55E4	1.75E4	12.19	10.78
512	7.01E3	1.04E4	26.94	18.20



**Fig. 13** Numerical results obtained solving the double pendulum system (4.4) for  $J \in \{32, 64, 128, 256, 512\}$ . **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 (2.5) and (3.18) for parareal and GParareal, respectively. **b** The corresponding speedup results using the same lines and colours. The theoretical results were calculated using (2.6) and (3.19) for parareal and GParareal, respectively

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 could converge in even fewer iterations. Similar results were illustrated for the Rössler system in Sect. 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 Sects. 4.3, and 4.4, GParareal was tested on a larger number of processors (up to 512), verifying the theoretical computational complexity results given in Sect. 3.4 and that the cost of the GP needs to be much smaller than the cost of the fine solver in order for speedup to be maximised. 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 in two ways. First, an increasing number of data points,  $\mathcal{O}(kJ)$ , is problematic for the standard cubic complexity GP implemented here. In this case, a more sophisticated (non-cubic complexity) emulator or perhaps using neural networks could be beneficial. Second, trying to emulate a  $d$ -dimensional function  $\mathcal{F}_{\Delta T} - \mathcal{G}_{\Delta T}$  is difficult if the number of evaluation points is not sufficient. One option to tackle this may be to obtain more acquisition data by launching more  $\mathcal{F}_{\Delta T}$  and  $\mathcal{G}_{\Delta T}$  runs using the idle processors to further train the emulator at little additional computational cost. However, as shown in Sect. 3.3, the accuracy of the GP emulator is strongly controlled by the fill distance of the set of evaluation points, which is generally difficult to restrict when  $d$  is large. One could think about using legacy data generated by evaluating  $\mathcal{F}_{\Delta T} - \mathcal{G}_{\Delta T}$  at

specific input locations (for example, a uniform grid) that satisfy certain fill distance requirements in the state space.

### A. Pseudocode 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  $j$ th mesh
point and  $k$ th iteration respectively (note
 $V_0^k = \hat{V}_0^k = \tilde{V}_0^k = u^0 \forall k$ ). If known, initialise any
legacy  $\mathcal{F}_{\Delta T} - \mathcal{G}_{\Delta T}$  input data  $\mathbf{x}$ , output data  $\mathbf{y}$ , and
hyperparameters  $\theta$ .
%Calculate approximate initial values at
each  $t_j$  by running  $\mathcal{G}_{\Delta T}$  serially.
1 for  $j = 1$  to  $J$  do
2    $\tilde{V}_j^0 = \mathcal{G}_{\Delta T}(\tilde{V}_{j-1}^0)$ ;
3    $V_j^0 = \hat{V}_j^0$ ;
4 end
5 for  $k = 1$  to  $J$  do
   %Propagate refined solutions (from
   iteration  $k - 1$ ) on unconverged
   sub-intervals by running  $\mathcal{F}_{\Delta T}$  in
   parallel.
6   for  $j = I + 1$  to  $J$  do
7      $\tilde{V}_j^{k-1} = \mathcal{F}_{\Delta T}(V_{j-1}^{k-1})$ ;
8   end
9    $I = I + 1$ ;
10   $V_I^k = \tilde{V}_I^{k-1}$  for all  $k$ ; %copy converged solution
   at  $t_I$  to future  $k$ .
11   $\mathbf{x} = \text{append}(\mathbf{x}, (V_I^{k-1}, \dots, V_{J-1}^{k-1})^T)$ ; %collect new
   input data.
12   $\mathbf{y} = \text{append}(\mathbf{y}, (\tilde{V}_{I+1}^{k-1} - \hat{V}_{I+1}^{k-1}, \dots, \tilde{V}_J^{k-1} - \hat{V}_J^{k-1})^T)$ ;
   %collect new output data.
13   $\theta = \text{GPoptimise}(\mathbf{x}, \mathbf{y}, \theta)$ ; %optimise
   hyperparameters.
   %Propagate refined solution (at iteration
    $k$ ) with  $\mathcal{G}_{\Delta T}$ , then correct using the
   expected value of the GP prediction
   (3.7) (this step cannot be carried out
   in parallel).
14  for  $j = I + 1$  to  $J$  do
15      $x^* = V_{j-1}^k$ ;
16      $\hat{V}_j^k = \mathcal{G}_{\Delta T}(x^*)$ ;
17      $y^* = \text{GPpredict}(\mathbf{x}, \mathbf{y}, \theta, x^*)$ ; %returns Gaussian
   random variable
18      $V_j^k = \mathbb{E}[y^*] + \hat{V}_j^k$ ;
19  end
   %Evaluate the stopping criterion, saving
   all solutions up to  $t_I$ .
20   $I = \max_{n \in \{1, \dots, N\}} |V_n^k - V_n^{k-1}| < \epsilon \forall i < n$ ;
21  if  $I = N$  then
22     return  $k, V^k, \mathbf{x}, \mathbf{y}, \theta$ ; %if tolerance met for
   all time steps, stop.
23  end
24 end

```

---

It should also be noted that GParareal may not always provide faster convergence using legacy data if such legacy evaluations of  $\mathcal{F}_{\Delta T} - \mathcal{G}_{\Delta T}$  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 equation (3.11), 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). During early iterations of GParareal, when little acquisition data are available, the uncertainty in the GP posterior (i.e the variance) may be large at points of interest. By retaining the GP posterior uncertainty, one could (ideally) propagate the full uncertainty using the coarse solver to the next time step and then continue. While this would produce a probabilistic version of GParareal, this would be a computationally expensive process that we wish to avoid at this stage. One alternative to approximating (3.11) by its expected value could be to draw a random sample instead. 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. (2022). 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 (3.11) arises if the input initial value is at least one or two length-scale distances away from any other known input value in our acquisition dataset. In this case, we then might expect the GP emulation of the mean in (3.11) to have high variance and so a fallback to the deterministic parareal correction for  $\mathcal{F}_{\Delta T} - \mathcal{G}_{\Delta T}$  (see (2.3)) could be built in as a next best correction to the coarse prediction in (3.11). Among others, these are two alternative formulations of GParareal that are worth investigating to account for the whole Gaussian distribution provided by the emulator and not just its mean value.

Follow-up work will focus on extending GParareal, using some of the methods suggested above, to solve higher-dimensional systems of ODEs in parallel (possibly PDEs). In the longer term, we aim to develop a truly probabilistic time-parallel 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, in particular Arkadiy Davydov. The views and opinions expressed herein do not necessarily reflect those of any of the above-named institutions or funding agencies. For the purpose of open access, the author has applied a CC BY public copyright licence to any Author Accepted Manuscript version arising.

**Open Access** This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

## References

- Ait-Ameur, K., Maday, Y., Tajchman, M.: Multi-step variant of the parareal algorithm. In: Domain Decomposition Methods in Science and Engineering XXV, pp. 393–400. Springer, Cham (2020)
- Ait-Ameur, K., Maday, Y., Tajchman, M.: 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, pp. 169–178. Springer, Cham (2021)
- Álvarez, M.A., Rosasco, L., Lawrence, N.D.: Kernels for vector-valued functions: a review. *Found. Trends Mach. Learn.* **4**, 195–266 (2011). <https://doi.org/10.1561/22000000036>
- Baffico, L., Bernard, S., Maday, Y., Turinici, G., Zérah, G.: Parallel-in-time molecular-dynamics simulations. *Phys. Rev. E Stat. Phys. Plasmas Fluid Relat. Interdiscip. Top.* **66**, 4–4 (2002). <https://doi.org/10.1103/PhysRevE.66.057701>
- Bal, G.: On the convergence and the stability of the parareal algorithm to solve partial differential equations. *Lect. Note Comput. Sci. Eng.* **40**, 425–432 (2005). [https://doi.org/10.1007/3-540-26825-1\\_43](https://doi.org/10.1007/3-540-26825-1_43)
- Bosch, N., Hennig, P., Tronarp, F.: Calibrated adaptive probabilistic ODE solvers. In: *Proceedings of the 24th International Conference on Artificial Intelligence and Statistics*, pp. 3466–3474, (2021). URL <http://proceedings.mlr.press/v130/bosch21a/bosch21a.pdf>
- Clarke, A., Davies, C., Ruprecht, D., Tobias, S.: Parallel-in-time integration of kinematic dynamos. *J. Comput. Phys. X* **7**, 100057 (2020). <https://doi.org/10.1016/j.jcpx.2020.100057>
- Cressie, N.: Spatial prediction and kriging. In: *Statistics for Spatial Data*, pp. 105–209. Wiley, New Jersey (1993)
- Dai, X., Le Bris, C., Legoll, F., Maday, Y.: Symmetric parareal algorithms for hamiltonian systems. *ESAIM Math. Model. Numer. Anal.* **47**, 717–742 (2013). <https://doi.org/10.1051/m2an/2012046>
- Danby, J.: Computer Modeling: From Sports to Spaceflight – From Order to Chaos. Willmann-Bell, Richmond, VA (1997)
- Dolean, V., Jolivet, P., Nataf, F.: An Introduction to Domain Decomposition Methods. Society for Industrial and Applied Mathematics, Philadelphia, PA (2015)
- Elwasif, W. R., Foley, S. S., Bernholdt, D. E., Berry, L. A., Samaddar, D., Newman, D. E., Sanchez, R.: 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*, pp. 15–24, ACM Press, New York, NY, (2011). <https://doi.org/10.1145/2132876.2132883>
- Engblom, S.: Parallel in time simulation of multiscale stochastic chemical kinetics. *Multiscale Model. Simul.* **8**, 46–68 (2009). <https://doi.org/10.1137/080733723>
- Farhat, C., Chandesris, M.: Time-decomposed parallel time-integrators: theory and feasibility studies for fluid, structure, and fluid-structure applications. *Int. J. Numer. Method Eng.* **58**, 1397–1434 (2003). <https://doi.org/10.1002/nme.860>
- FitzHugh, R.: Impulses and physiological states in theoretical models of nerve membrane. *Biophys. J.* **1**, 445–466 (1961). [https://doi.org/10.1016/S0006-3495\(61\)86902-6](https://doi.org/10.1016/S0006-3495(61)86902-6)
- Gander, M.J.: 50 Years of time parallel time integration. In: *Multiple Shooting and Time Domain Decomposition Methods*, pp. 69–113. Springer, New York (2015)
- Gander, M.J., Hairer, E.: Nonlinear convergence analysis for the parareal algorithm. In: *Lecture Notes in Computational Science and Engineering*, pp. 45–56. Springer, New York (2008)
- Gander, M.J., Vandewalle, S.: Analysis of the parareal time-parallel time-integration method. *SIAM J. Sci. Comput.* **29**, 556–578 (2007). <https://doi.org/10.1137/05064607X>
- Garrido, I., Lee, B., Fladmark, G.E., Espedal, M.S.: Convergent iterative schemes for time-parallelization. *Math. Comput.* **75**(255), 1403–1428 (2006). <https://doi.org/10.1090/S0025-5718-06-01832-1>
- Grafke, T., Schäfer, T., Vanden-Eijnden, E.: 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*, pp. 17–55. Springer, New York (2017)
- Hairer, E., Nørsett, S.P., Wanner, G.: Solving Ordinary Differential Equations I: Nonstiff Problems. Springer, Cham (1993)
- Hamon, F.P., Schreiber, M., Minion, M.: Parallel-in-time multi-level integration of the shallow-water equations on the rotating sphere. *J. Comput. Phys.* **407**, 109210 (2020). <https://doi.org/10.1016/j.jcp.2019.109210>
- Hennig, P., Osborne, M.A., Kersting, H.P.: Probabilistic Numerics: Computation as Machine Learning. Cambridge University Press, Cambridge (2022)
- Kanagawa, M., Hennig, P., Sejdinovic, D., Sriperumbudur, B. K.: Gaussian processes and kernel methods: a review on connections and equivalences, (2018). [arXiv:1807.02582](https://arxiv.org/abs/1807.02582)
- Karvonen, T.: Asymptotic bounds for smoothness parameter estimates in Gaussian process interpolation, (2022). [arXiv:2203.05400](https://arxiv.org/abs/2203.05400)
- Karvonen, T., Oates, C. J.: Maximum likelihood estimation in Gaussian process regression is ill-posed, (2022). [arXiv:2203.09179](https://arxiv.org/abs/2203.09179)
- Kersting, H., Hennig, P.: Active uncertainty calibration in Bayesian ODE solvers. In: *Proceedings of the Thirty-Second Conference on Uncertainty in Artificial Intelligence*, pp. 309–318, (2016). <https://doi.org/10.5555/3020948.3020981>
- Kersting, H., Sullivan, T.J., Hennig, P.: Convergence rates of Gaussian ODE filters. *Stat. Comput.* **30**(6), 1791–1816 (2020). <https://doi.org/10.1007/s11222-020-09972-4>
- Krämer, N., Bosch, N., Schmidt, J., Hennig, P.: Probabilistic ODE solutions in millions of dimensions. In: K. Chaudhuri, S. Jegelka, L. Song, C. Szepesvari, G. Niu, and S. Sabato, (eds.) *Proceedings of the 39th International Conference on Machine Learning*, vol. 162, pp. 11634–11649, (2022). <https://proceedings.mlr.press/v162/kramer22b/kramer22b.pdf>
- Legoll, F., Lelièvre, T., Myerscough, K., Samaey, G.: Parareal computation of stochastic differential equations with time-scale separation:

- a numerical convergence study. *Comput. Vis. Sci.* **23**, 1–18 (2020). <https://doi.org/10.1007/s00791-020-00329-y>
- Legoll, F., Lelièvre, T., Sharma, U.: (2021) An adaptive parareal algorithm: application to the simulation of molecular dynamics trajectories. URL <https://hal.archives-ouvertes.fr/hal-03189428>
- Lions, J.L., Maday, Y., Turinici, G.: Resolution d'EDP par un schema en temps parareel. *Compte Rendus Acad. Sci. Ser. I Math.* (2001). [https://doi.org/10.1016/S0764-4442\(00\)01793-6](https://doi.org/10.1016/S0764-4442(00)01793-6)
- Maday, Y., Mula, O.: An adaptive parareal algorithm. *J. Comput. Appl. Math.* **377**, 112915–112915 (2020). <https://doi.org/10.1016/j.cam.2020.112915>
- Maday, Y., Turinici, G.: The parareal in time iterative solver: a further direction to parallel implementation. *Lect. Note Comput. Sci. Eng.* **40**, 441–448 (2005). [https://doi.org/10.1007/3-540-26825-1\\_45](https://doi.org/10.1007/3-540-26825-1_45)
- Mann, A.: Core concept: nascent exascale supercomputers offer promise, present challenges. *Proc. Nat. Acad. Sci.* **117**, 22623–22625 (2020). <https://doi.org/10.1073/pnas.2015968117>
- Meng, X., Li, Z., Zhang, D., Karniadakis, G.E.: PPINN: parareal physics-informed neural network for time-dependent PDEs. *Comput. Method Appl. Mech. Eng.* **370**, 113250 (2020). <https://doi.org/10.1016/j.cma.2020.113250>
- Murphy, K.P.: *Probabilistic Machine Learning: Advanced Topics*. MIT Press, Cambridge (2023)
- Nagumo, J., Arimoto, S., Yoshizawa, S.: An active pulse transmission line simulating nerve axon. *Proc. IRE* **50**, 2061–2070 (1962). <https://doi.org/10.1109/JRPROC.1962.288235>
- Oates, C.J., Sullivan, T.J.: A modern retrospective on probabilistic numerics. *Stat. Comput.* **29**, 1335–1351 (2019). <https://doi.org/10.1007/s11222-019-09902-z>
- O'Hagan, A.: Curve fitting and optimal design for prediction. *J. R. Stat. Soc. Ser. B Methodol.* **40**, 1–24 (1978). <https://doi.org/10.1111/j.2517-6161.1978.tb01643.x>
- Ong, B.W., Schroder, J.B.: Applications of time parallelization. *Comput. Vis. Sci.* (2020). <https://doi.org/10.1007/s00791-020-00331-4>
- Pentland, K., Tamborrino, M., Samaddar, D., Appel, L.C.: Stochastic parareal: an application of probabilistic methods to time-parallelization. *SIAM J. Sci. Comput.* (2022). <https://doi.org/10.1137/21M1414231>
- Quiñonero Candela, J., Rasmussen, C.E.: A unifying view of sparse approximate Gaussian process regression. *J. Mach. Learn. Res.* **6**(65), 1939–1959 (2005)
- Rasmussen, C.E.: Gaussian processes in machine learning. In: *Advanced Lectures on Machine Learning: ML Summer Schools*, pp. 63–71. Springer, Cham (2003)
- Rasmussen, C.E., Williams, C.K.I.: *Gaussian Processes for Machine Learning*, Adaptive Computation and Machine Learning. MIT Press, Cambridge (2006)
- Rössler, O.E.: An equation for continuous chaos. *Phys. Lett. A* **57**, 397–398 (1976). [https://doi.org/10.1016/0375-9601\(76\)90101-8](https://doi.org/10.1016/0375-9601(76)90101-8)
- Ruprecht, D.: Convergence of Parareal with spatial coarsening. *Proc. Appl. Math. Mech.* **14**, 1031–1034 (2014). <https://doi.org/10.1002/pamm.201410490>
- Samaddar, D., Newman, D.E., Sánchez, R.: Parallelization in time of numerical simulations of fully-developed plasma turbulence using the parareal algorithm. *J. Comput. Phys.* **229**, 6558–6573 (2010). <https://doi.org/10.1016/j.jcp.2010.05.012>
- Samaddar, D., Coster, D.P., Bonnin, X., Berry, L.A., Elwasif, W.R., Batchelor, D.B.: Application of the parareal algorithm to simulations of ELMs in ITER plasma. *Comput. Phys. Commun.* **235**, 246–257 (2019). <https://doi.org/10.1016/j.cpc.2018.08.007>
- Schäfer, F., Sullivan, T.J., Owhadi, H.: Compression, inversion, and approximate PCA of dense kernel matrices at near-linear computational complexity. *Multiscale Model. Simul.* **19**(2), 688–730 (2021). <https://doi.org/10.1137/19M129526X>
- Schober, M., Särkkä, S., Hennig, P.: A probabilistic model for the numerical solution of initial value problems. *Stat. Comput.* **29**, 99–122 (2019). <https://doi.org/10.1007/s11222-017-9798-7>
- Snelson, E., Ghahramani, Z.: Sparse Gaussian processes using pseudo-inputs. *Adv. Neural Inf. Process. Syst.* **18**, 1259–1266 (2006)
- Snelson, E., Ghahramani, Z.: (2007) Local and global sparse Gaussian process approximations. In: *Proceedings of the Eleventh International Conference on Artificial Intelligence and Statistics*, p. 524–531
- Stuart, A.M., Teckentrup, A.L.: Posterior consistency for Gaussian process approximations of bayesian posterior distributions. *Math. Comput.* **87**(310), 721–753 (2018). <https://doi.org/10.1090/mcom/3244>
- Trefethen, L.N., Birkisson, A., Driscoll, T.: *Exploring ODEs*. Society for Industrial and Applied Mathematics, Philadelphia (2017)
- Trindade, J.M.F., Pereira, J.C.F.: Parallel-in-time simulation of two-dimensional, unsteady, incompressible laminar flows. *Numer. Heat Transf. Part B Fundam.* **50**, 25–40 (2006). <https://doi.org/10.1080/10407790500459379>
- Tronarp, F., Kersting, H., Särkkä, S., Hennig, P.: Probabilistic solutions to ordinary differential equations as nonlinear Bayesian filtering: a new perspective. *Stat. Comput.* **29**, 1297–1315 (2019). <https://doi.org/10.1007/s11222-019-09900-1>
- Wendland, H.: *Scattered Data Approximation*, Cambridge Monographs on Applied and Computational Mathematics. Cambridge University Press, Cambridge (2004)
- Wenger, J., Krämer, N., Pfortner, M., Schmidt, J., Bosch, N., Effenberger, N., Zenn, J., Gessner, A., Karvonen, T., Briol, F.-X., Mahsereci, M., Hennig, P.: ProbNum: probabilistic numerics in python, (2021). [arXiv:2112.02100](https://arxiv.org/abs/2112.02100)

**Publisher's Note** Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.