GParareal: (Towards) A Probabilistic Time-Parallel ODE Solver

K. Pentland[1] M. Tamborrino[1] T. J. Sullivan[1,2] J. Buchanan[3] L. C. Appel[3]

Exascale Computing Challenges: Parallel-in-Time Algorithms
University of Exeter, UK
11 January 2023

[1]**University of Warwick, UK**
[2]Alan Turing Institute, UK
[3]Culham Centre for Fusion Energy, UK

Overview

- We are interested in solving IVPs of the form

$$\frac{\mathrm{d}\boldsymbol{u}}{\mathrm{d}t} = \boldsymbol{f}\big(t, \boldsymbol{u}(t)\big) \quad \text{over} \quad t \in [t_0, T], \quad \text{with} \quad \boldsymbol{u}(t_0) = \boldsymbol{u}^0 \in \mathcal{U} \subseteq \mathbb{R}^d, \tag{1}$$

  using two time-stepping schemes, an expensive high accuracy fine solver ($\mathcal{F}$) and a cheap less accurate coarse solver ($\mathcal{G}$).

Overview

- We are interested in solving IVPs of the form

$$\frac{\mathrm{d}\boldsymbol{u}}{\mathrm{d}t} = \boldsymbol{f}\big(t, \boldsymbol{u}(t)\big) \quad \text{over} \quad t \in [t_0, T], \quad \text{with} \quad \boldsymbol{u}(t_0) = \boldsymbol{u}^0 \in \mathcal{U} \subseteq \mathbb{R}^d, \tag{1}$$

  using two time-stepping schemes, an expensive high accuracy fine solver ($\mathcal{F}$) and a cheap less accurate coarse solver ($\mathcal{G}$).
- We seek numerical solutions $\boldsymbol{U}_j \approx \boldsymbol{u}(t_j)$ to (1) on a pre-defined mesh $\boldsymbol{t} = (t_0, \ldots, t_J)$, where $t_{j+1} = t_j + \Delta T$ for fixed $\Delta T = (T - t_0)/J$.

Overview

- We are interested in solving IVPs of the form

$$\frac{\mathrm{d}\boldsymbol{u}}{\mathrm{d}t} = \boldsymbol{f}\big(t, \boldsymbol{u}(t)\big) \quad \text{over} \quad t \in [t_0, T], \quad \text{with} \quad \boldsymbol{u}(t_0) = \boldsymbol{u}^0 \in \mathcal{U} \subseteq \mathbb{R}^d, \tag{1}$$

  using two time-stepping schemes, an expensive high accuracy fine solver ($\mathcal{F}$) and a cheap less accurate coarse solver ($\mathcal{G}$).
- We seek numerical solutions $\boldsymbol{U}_j \approx \boldsymbol{u}(t_j)$ to (1) on a pre-defined mesh $\boldsymbol{t} = (t_0, \ldots, t_J)$, where $t_{j+1} = t_j + \Delta T$ for fixed $\Delta T = (T - t_0)/J$.
- Computational budget does not allow $\mathcal{F}$ to be run over whole $[t_0, T]$ but does allow on time slices $[t_j, t_{j+1}]$ **in parallel**.

Overview

- We are interested in solving IVPs of the form

$$\frac{\mathrm{d}\boldsymbol{u}}{\mathrm{d}t} = \boldsymbol{f}\big(t, \boldsymbol{u}(t)\big) \quad \text{over} \quad t \in [t_0, T], \quad \text{with} \quad \boldsymbol{u}(t_0) = \boldsymbol{u}^0 \in \mathcal{U} \subseteq \mathbb{R}^d, \tag{1}$$

  using two time-stepping schemes, an expensive high accuracy fine solver ($\mathcal{F}$) and a cheap less accurate coarse solver ($\mathcal{G}$).
- We seek numerical solutions $\boldsymbol{U}_j \approx \boldsymbol{u}(t_j)$ to (1) on a pre-defined mesh $\boldsymbol{t} = (t_0, \ldots, t_J)$, where $t_{j+1} = t_j + \Delta T$ for fixed $\Delta T = (T - t_0)/J$.
- Computational budget does not allow $\mathcal{F}$ to be run over whole $[t_0, T]$ but does allow on time slices $[t_j, t_{j+1}]$ **in parallel**.
- **Takeaway message:** We propose the GPparareal algorithm, a "parareal"-type algorithm (Lions et al., 2001) that uses a Gaussian process (GP) emulator (trained on solution data from $\mathcal{F}$ and $\mathcal{G}$) to solve (1) in parallel.
- **Motivation:** Borrow ideas from probabilistic numerics (PN) to make more efficient use of the simulation data generated within parareal and perhaps quantify uncertainty on the fly.

Parareal

Parareal: The algorithm

- **Iteration $k = 0$:** calculate approximate solutions to (1) sequentially using $\mathcal{G}$, on a single processor, such that

$$U_{j+1}^0 = \mathcal{G}(U_j^0) \quad j = 0, \ldots, J-1. \tag{2}$$

Parareal: The algorithm

- **Iteration $k = 0$:** calculate approximate solutions to (1) sequentially using $\mathcal{G}$, on a single processor, such that

$$U_{j+1}^0 = \mathcal{G}(U_j^0) \quad j = 0, \ldots, J-1. \tag{2}$$

- **Iteration $k \geqslant 1$:** propagate each approximation in (2) using $\mathcal{F}$ *in parallel*, on J processors, to obtain $\mathcal{F}(U_j^0)$ for $j = 0, \ldots, J-1$. These values are then used in the predictor-corrector (PC):

$$U_{j+1}^k = \underbrace{\mathcal{G}(U_j^k)}_{\text{predict}} + \underbrace{\mathcal{F}(U_j^{k-1}) - \mathcal{G}(U_j^{k-1})}_{\text{correct}} \quad \text{for} \quad j = 0, \ldots, J-1. \tag{3}$$

For pre-defined tolerance $\varepsilon > 0$, the solution U_j^k has converged up to time t_l if

$$|U_j^k - U_j^{k-1}| < \varepsilon \quad \forall j \leqslant l. \tag{4}$$

Parareal: The algorithm

- **Iteration $k = 0$:** calculate approximate solutions to (1) sequentially using $\mathcal{G}$, on a single processor, such that

$$U_{j+1}^0 = \mathcal{G}(U_j^0) \quad j = 0, \ldots, J-1. \tag{2}$$

- **Iteration $k \geqslant 1$:** propagate each approximation in (2) using $\mathcal{F}$ *in parallel*, on J processors, to obtain $\mathcal{F}(U_j^0)$ for $j = 0, \ldots, J-1$. These values are then used in the predictor-corrector (PC):

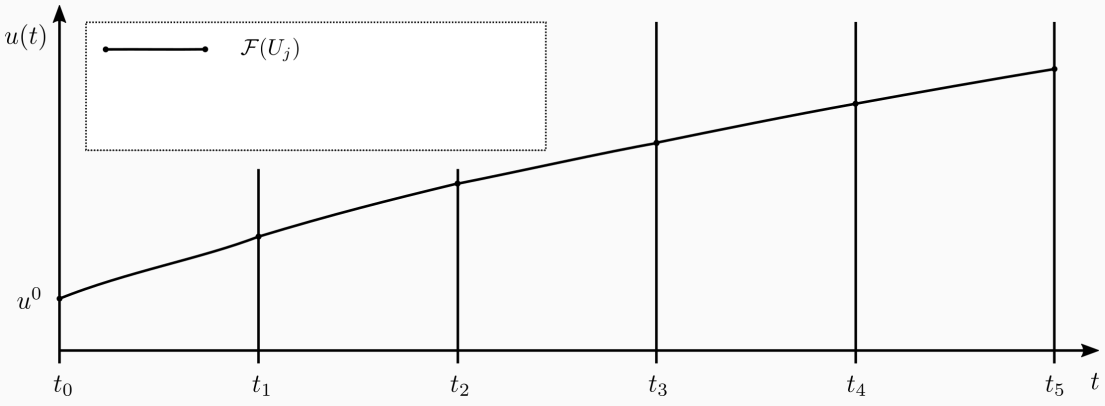
$$U_{j+1}^k = \underbrace{\mathcal{G}(U_j^k)}_{\text{predict}} + \underbrace{\mathcal{F}(U_j^{k-1}) - \mathcal{G}(U_j^{k-1})}_{\text{correct}} \quad \text{for} \quad j = 0, \ldots, J-1. \tag{3}$$

  For pre-defined tolerance $\varepsilon > 0$, the solution U_j^k has converged up to time t_l if

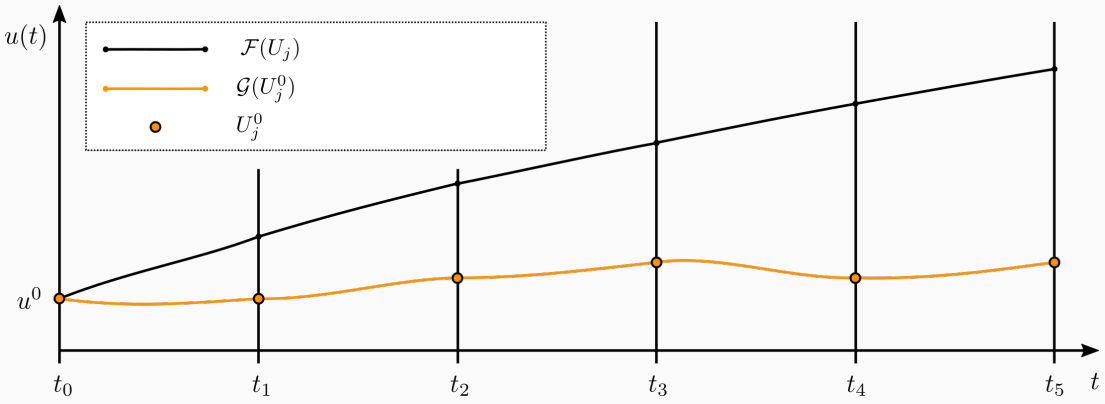
$$|U_j^k - U_j^{k-1}| < \varepsilon \quad \forall j \leqslant l. \tag{4}$$

- **Key point:** Algorithm stops once $l = J$, "converging" in k (out of J) iterations.

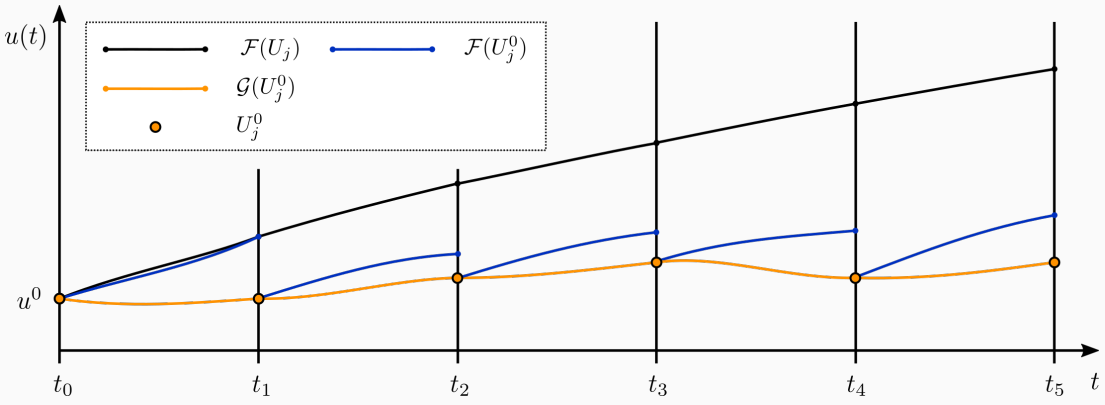
Parareal: How it works



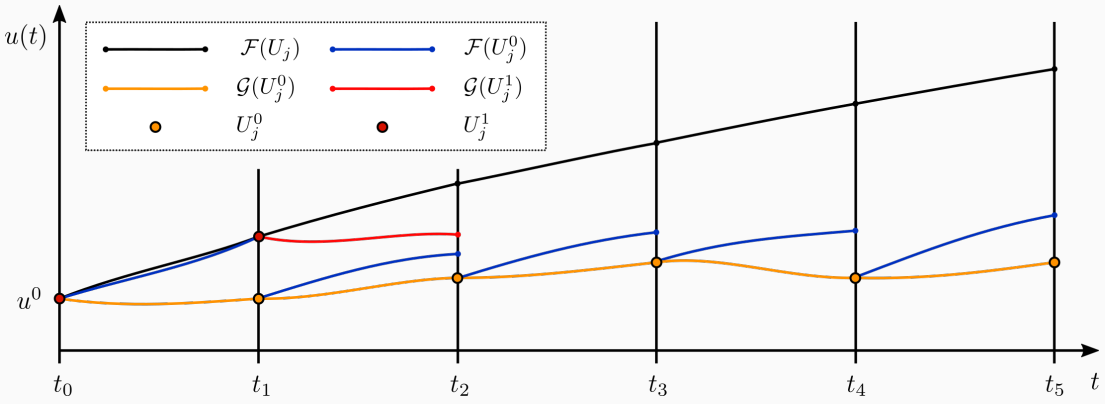
Parareal: How it works



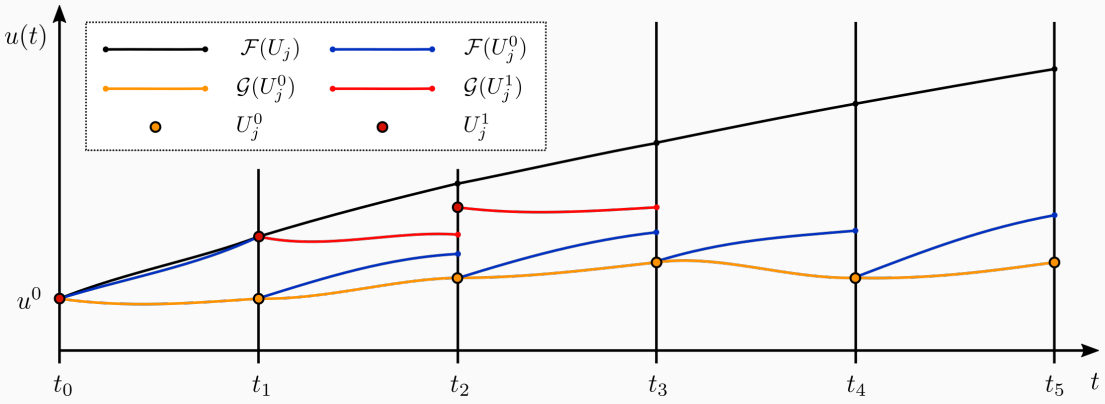
Parareal: How it works



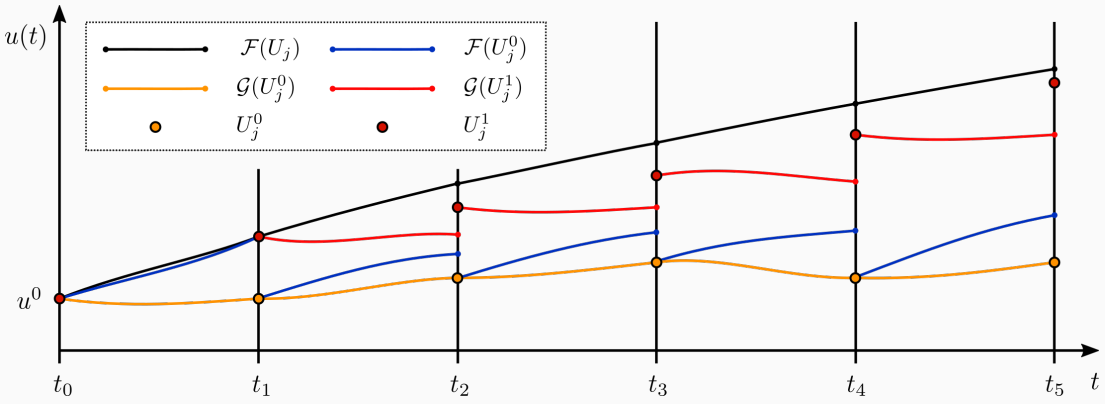
Parareal: How it works



Parareal: How it works



Parareal: How it works



Parareal: Convergence and Complexity

- After k iterations, the first k time slices (at minimum) are converged, as the exact initial condition (u_0) has been propagated by $\mathcal{F}$ at least k times.
- If parareal converges in $k = J$ iterations, the solution will be equal to the one found by calculating (1) serially, at an even higher computational cost! Convergence in $k \ll J$ iterations is necessary if significant parallel speed-up is to be realised.
- Assume, assume running $\mathcal{F}$ over any $[t_j, t_{j+1}]$, $j \in \{0, \ldots, J-1\}$, takes wallclock time $T_{\mathcal{F}}$ (denote time $T_{\mathcal{G}}$ similarly for $\mathcal{G}$). Therefore, calculating (1) using $\mathcal{F}$ serially, takes approximately $T_{\text{serial}} = JT_{\mathcal{F}}$ seconds. Using parareal, the total wallclock time (in the worst case, excluding any serial overheads) can be approximated by

$$T_{\text{para}} \approx \underbrace{JT_{\mathcal{G}}}_{\text{Iteration } 0} + \sum_{i=1}^{k} \underbrace{\left(T_{\mathcal{F}} + (J-i)T_{\mathcal{G}}\right)}_{\text{Iterations } 1 \text{ to } k} = kT_{\mathcal{F}} + (k+1)\left(J - \frac{k}{2}\right)T_{\mathcal{G}}. \tag{5}$$

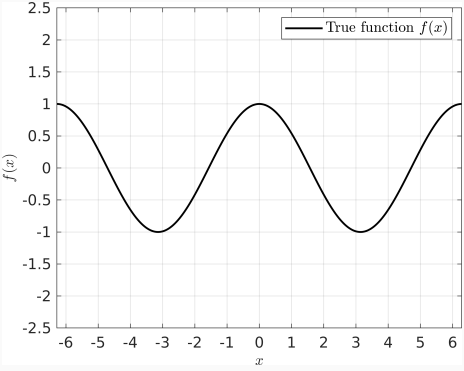
- The approximate parallel speed-up is therefore

$$S_{\text{para}} \approx \frac{T_{\text{serial}}}{T_{\text{para}}(k)} = \left[\frac{k}{J} + (k+1)\left(1 - \frac{k}{2J}\right)\frac{T_{\mathcal{G}}}{T_{\mathcal{F}}}\right]^{-1}. \tag{6}$$

GParareal

GPararealㅤ: What is a GP emulator?

GP emulation: a way to **statistically model** an **unknown (expensive-to-evaluate) function** using multivariate **Gaussian** distributions (Rasmussen and Williams, 2006).



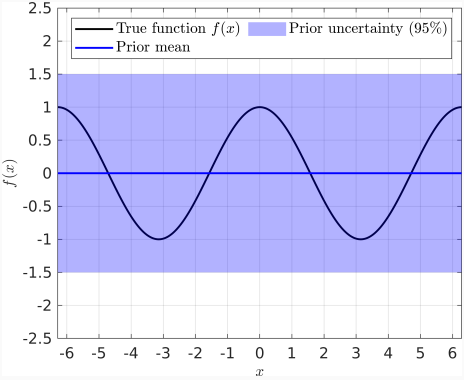
oth $\hat{\mu}(x^*)$ and $\hat{K}(x^*, x^*)$ have analytical expressions (not shown for clarity).

GPararealㅤ: What is a GP emulator?

GP emulation: a way to **statistically model** an **unknown (expensive-to-evaluate) function** using multivariate **Gaussian** distributions (Rasmussen and Williams, 2006).

- Step 1: Gaussian prior placed over the unknown function $f(x)$ (with known mean/covariance functions)

$$f(\boldsymbol{x}) \sim \mathcal{N}\big(\mu(\boldsymbol{x}), K(\boldsymbol{x}, \boldsymbol{x})\big).$$



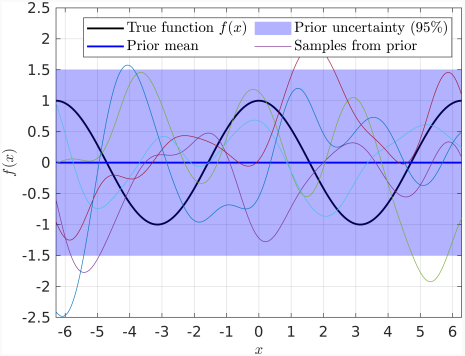
oth $\hat{\mu}(x^*)$ and $\hat{K}(x^*, x^*)$ have analytical expressions (not shown for clarity).

GPararealㅤ: What is a GP emulator?

GP emulation: a way to **statistically model** an **unknown (expensive-to-evaluate) function** using multivariate **Gaussian** distributions (Rasmussen and Williams, 2006).

- Step 1: Gaussian prior placed over the unknown function $f(x)$ (with known mean/covariance functions)

$$f(\boldsymbol{x}) \sim \mathcal{N}\big(\mu(\boldsymbol{x}), K(\boldsymbol{x}, \boldsymbol{x})\big).$$



oth $\hat{\mu}(x^*)$ and $\hat{K}(x^*, x^*)$ have analytical expressions (not shown for clarity).

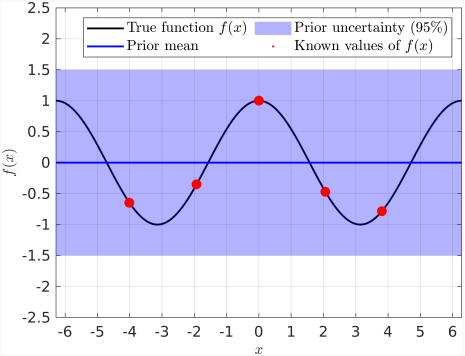
GPararealㅤ: What is a GP emulator?

GP emulation: a way to **statistically model** an **unknown (expensive-to-evaluate) function** using multivariate **Gaussian** distributions (Rasmussen and Williams, 2006).

- Step 1: Gaussian prior placed over the unknown function $f(x)$ (with known mean/covariance functions)

$$f(\boldsymbol{x}) \sim \mathcal{N}\big(\mu(\boldsymbol{x}), K(\boldsymbol{x}, \boldsymbol{x})\big).$$

- Step 2: Condition prior on known evaluations (red dots): $(\boldsymbol{x}, \boldsymbol{y}) = \big(x_i, f(x_i)\big)_{i=1,\ldots,N}$.



oth $\hat{\mu}(x^*)$ and $\hat{K}(x^*, x^*)$ have analytical expressions (not shown for clarity).

GPararealㅤ: What is a GP emulator?

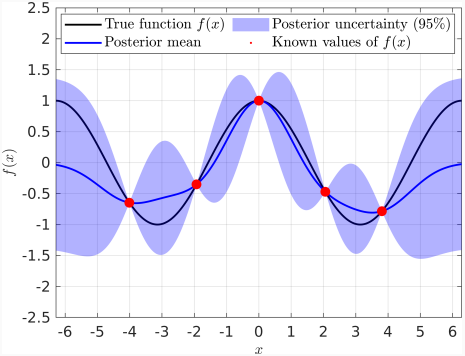
GP emulation: a way to **statistically model** an **unknown (expensive-to-evaluate) function** using multivariate **Gaussian** distributions (Rasmussen and Williams, 2006).

- Step 1: Gaussian prior placed over the unknown function $f(x)$ (with known mean/covariance functions)

$$f(\boldsymbol{x}) \sim \mathcal{N}\big(\mu(\boldsymbol{x}), K(\boldsymbol{x}, \boldsymbol{x})\big).$$

- Step 2: Condition prior on known evaluations (red dots): $(\boldsymbol{x}, \boldsymbol{y}) = \big(x_i, f(x_i)\big)_{i=1,\ldots,N}$.
- Step 3: Obtain Gaussian posterior, which can be queried at any unknown x^*:

$$f(x^*) \mid (\boldsymbol{x}, \boldsymbol{y}) \sim \mathcal{N}\big(\hat{\mu}(x^*), \hat{K}(x^*, x^*)\big).$$



Both $\hat{\mu}(x^*)$ and $\hat{K}(x^*, x^*)$ have analytical expressions (not shown for clarity).

GPararealː The idea

- Corrections in parareal PC based on information from **single previous iteration** $\rightarrow$ all other solution information ignored in Markovian-like manner.
- **Our idea:** improve corrections using GP emulator to reduce iterations k.
- **How?** We re-formulate the PC

$$U_{j+1}^k = \mathcal{F}(U_j^k) = (\mathcal{F} - \mathcal{G} + \mathcal{G})(U_j^k) = \underbrace{\mathcal{G}(U_j^k)}_{\text{prediction}} + \underbrace{(\mathcal{F} - \mathcal{G})(U_j^k)}_{\text{correction}}. \tag{7}$$

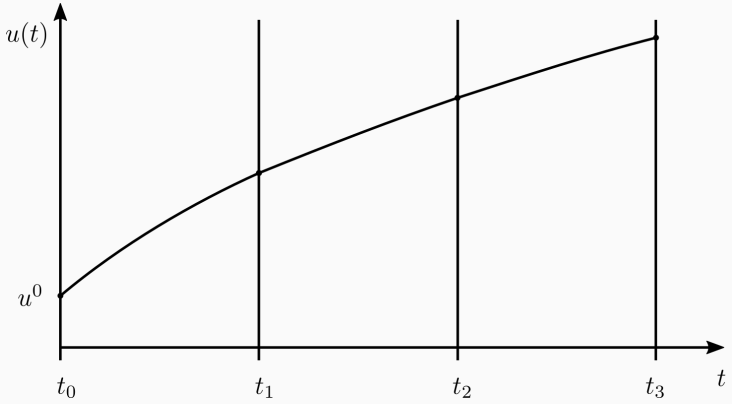
- We use a GP emulator to model the correction term, trained on *all* previously obtained evaluations of $\mathcal{F}$ and $\mathcal{G}$ ($(\boldsymbol{x}, \boldsymbol{y})$ is the dataset):

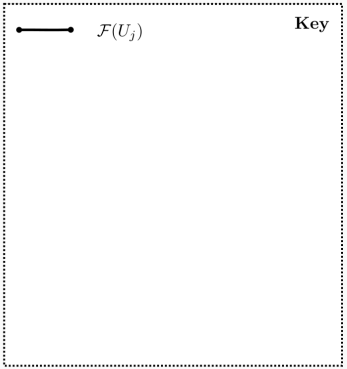
$$(\mathcal{F} - \mathcal{G})(U_j^k) \mid (\boldsymbol{x}, \boldsymbol{y}) \sim \mathcal{N}\big(\hat{\mu}(U_j^k), \hat{K}(U_j^k, U_j^k)\big). \tag{8}$$

- Whole Gaussian cannot be propagated in (7), so we approximate using the **mean value** and carry out the refinement:

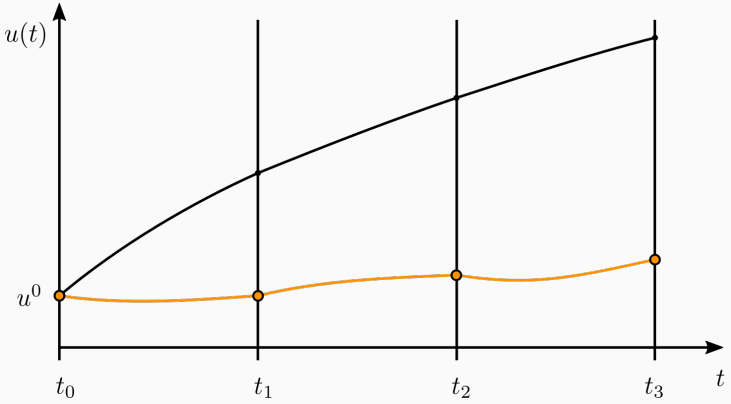
$$U_{j+1}^k = \mathcal{G}(U_j^k) + \hat{\mu}(U_j^k). \tag{9}$$

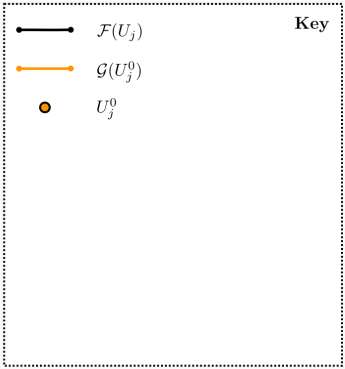
GParareal: How it works



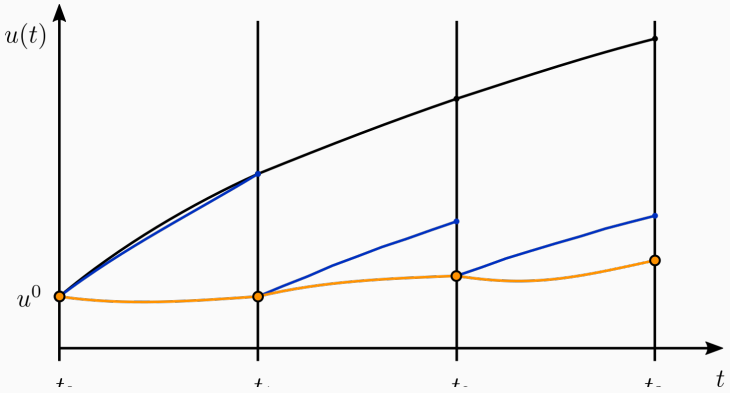


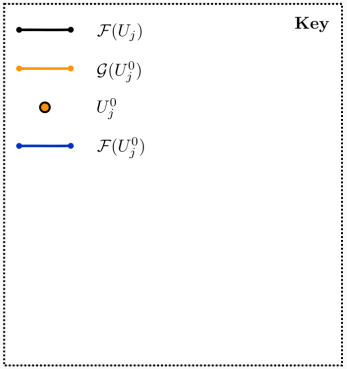
GPararealㅤ: How it works



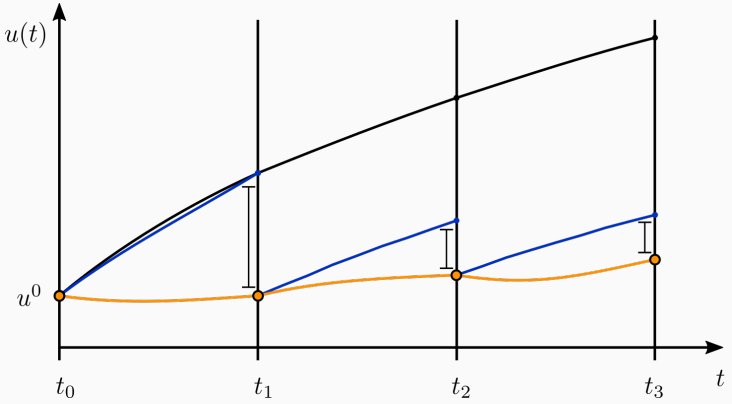


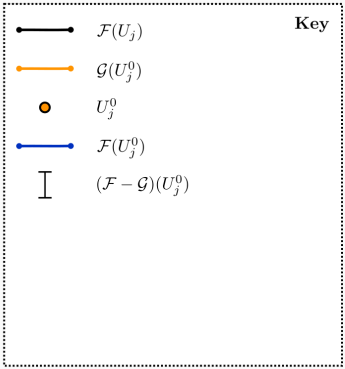
GPararealㅤ: How it works



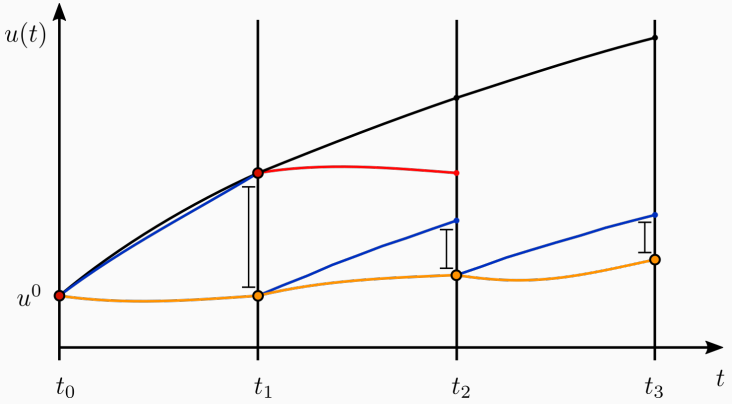


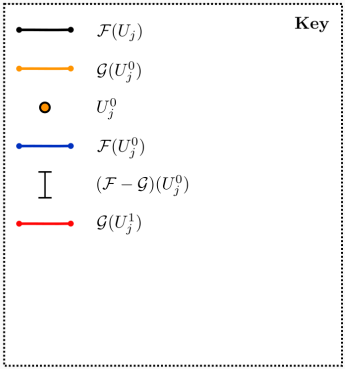
GPararealㅤ: How it works



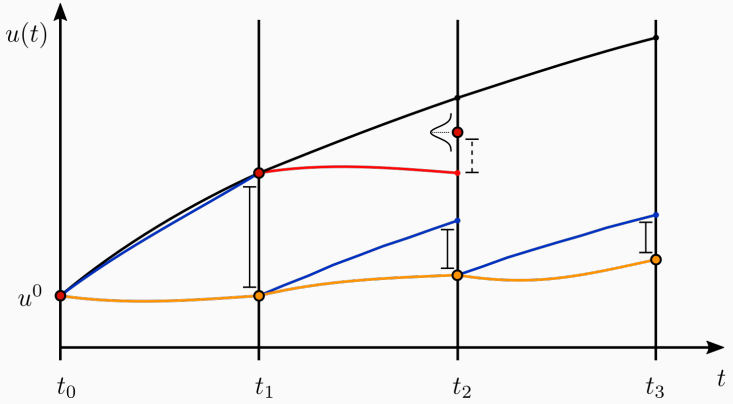


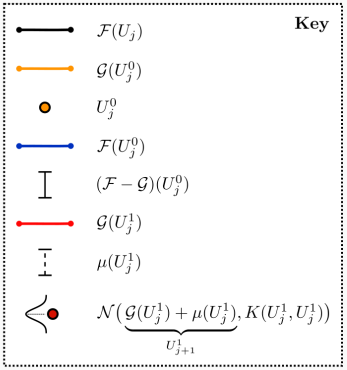
GPararealㅤ: How it works



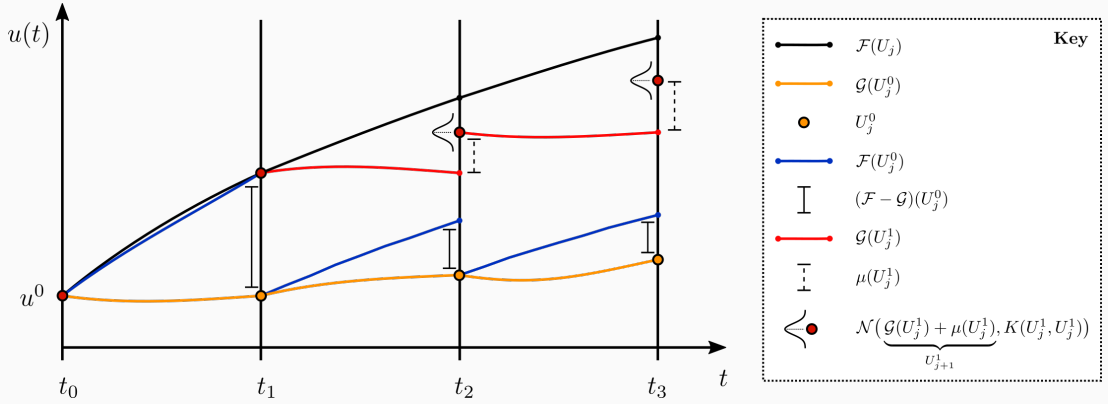


GPararealㅤ: How it works





GPararealㅤ: How it works



Key benefit: GPararealㅤcan re-use the $\mathcal{F}-\mathcal{G}$ data in future GPararealㅤsimulations as "legacy data" to pre-train the GP emulator and provide additional speedup — see the numerical experiments later.

Numerical Experiments

FitzHugh–Nagumo model: Solutions

Consider the FitzHugh–Nagumo (FHN) model (FitzHugh, 1961; Nagumo et al., 1962) given by

$$\frac{\mathrm{d}u_1}{\mathrm{d}t} = c\big(u_1 - \frac{u_1^3}{3} + u_2\big), \quad \frac{\mathrm{d}u_2}{\mathrm{d}t} = -\frac{1}{c}(u_1 - a + bu_2), \quad t \in [0, 40].$$

We integrate divide the interval into $J = 40$ slices and set the tolerance for both GParareal and parareal to $\varepsilon = 10^{-6}$. We use solvers $\mathcal{G} = \mathrm{RK2}$ and $\mathcal{F} = \mathrm{RK4}$ with $N_{\mathcal{G}} = 160$ and $N_{\mathcal{F}} = 1.6 \times 10^8$ steps respectively. Note that the large value of $N_{\mathcal{F}}$ is required to ensure that $\mathcal{F}$ is expensive to run and that parallel speedup can be realised (as both algorithms require $T_{\mathcal{G}}/T_{\mathcal{F}} \ll 1$).

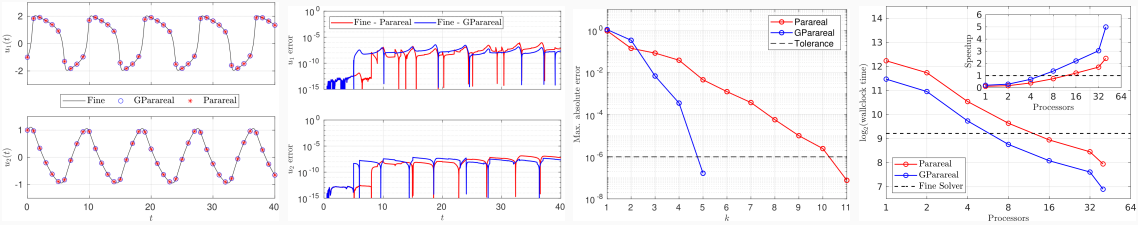
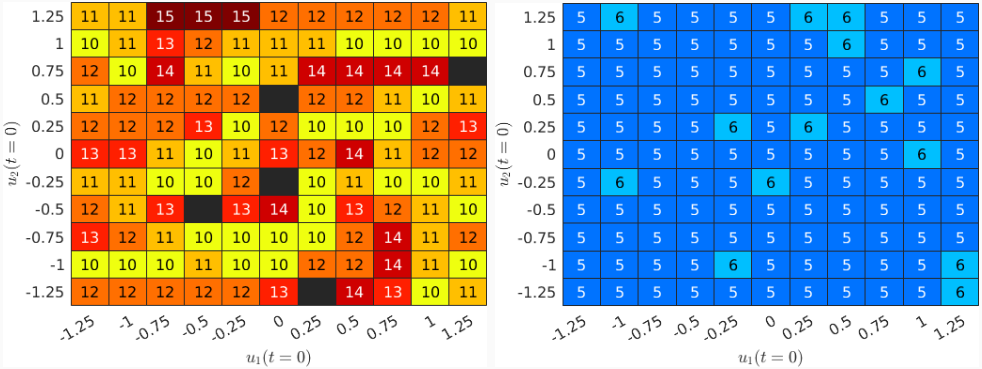


Figure 1: Numerical results obtained solving the FHN model for $\boldsymbol{u}^0 = (-1, 1)^\intercal$.

FitzHugh–Nagumo model: Convergence



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

Figure 2: Iterations count k (max. $J = 40$) for various initial values $\boldsymbol{u}^0 \in [-1.25, 1.25]^2$.

FitzHugh–Nagumo: Legacy Data

GParareal can use legacy data to pre-train the emulator and solve faster!

- Step 1: Solve FHN model using initial condition $\boldsymbol{u}^0 = (-1, 1)^\intercal$.
- Step 2: Store $\mathcal{F}$ and $\mathcal{G}$ solution data (= legacy data).
- Step 3: Re-initialise GParareal using legacy data to solve for new initial condition $\boldsymbol{u}^0 = (0.75, 0.25)^\intercal$.

Accuracy of solutions with or without legacy data is similar to that of parareal.

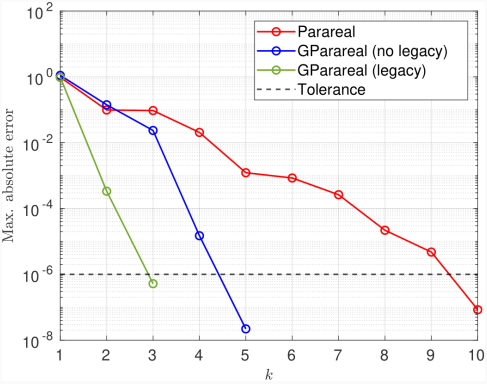


Figure 3: Iterations until convergence k (with/without legacy data).

Additional experiments on nonautonomous and chaotic systems in paper!

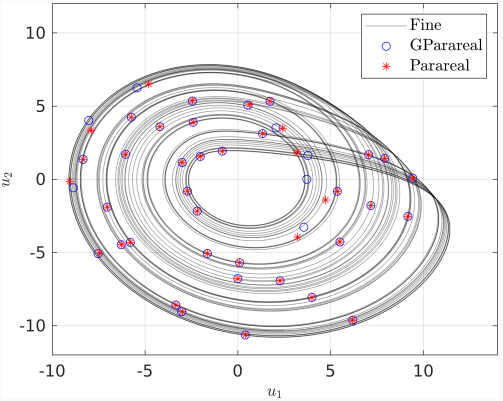
Rössler system: Solution

- Next we solve the Rössler system,

$$\frac{\mathrm{d}u_1}{\mathrm{d}t} = -u_2 - u_3, \quad \frac{\mathrm{d}u_2}{\mathrm{d}t} = u_1 + \hat{a}u_2, \quad \frac{\mathrm{d}u_3}{\mathrm{d}t} = \hat{b} + u_3(u_1 - \hat{c}), \tag{10}$$

  with parameters $(\hat{a}, \hat{b}, \hat{c}) = (0.2, 0.2, 5.7)$ that cause the system to exhibit chaotic behaviour (Rössler, 1976).

- Suppose we wish to integrate (10) over $t \in [0, 340]$ with initial values $\boldsymbol{u}_0 = (0, -6.78, 0.02)^\mathsf{T}$ and solvers $\mathcal{G} = \mathrm{RK1}$ and $\mathcal{F} = \mathrm{RK4}$. The interval is divided into $J = 40$ time slices, $N_\mathcal{G} = 9 \times 10^4$ coarse steps, and $N_\mathcal{F} = 4.5 \times 10^8$ fine steps.
- The convergence tolerance is set to $\varepsilon = 10^{-6}$.



Rössler system: Convergence

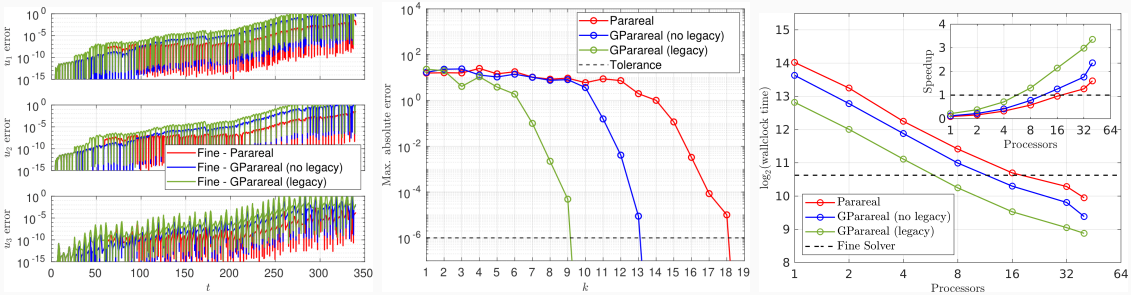


Figure 4: Numerical results obtained solving the Rössler system (10) over $t \in [0, 340]$. (a) The corresponding absolute errors between solutions from GParareal and parareal vs. the fine solution. (b) Maximum absolute errors from (4) of each algorithm at successive iterations k until tolerance $\varepsilon = 10^{-6}$ is met. (c) Median wallclock times (taken over 5 runs) of each simulation against the number of processors (up to 40). Inset: The corresponding parallel speedup vs. the serial wallclock time.

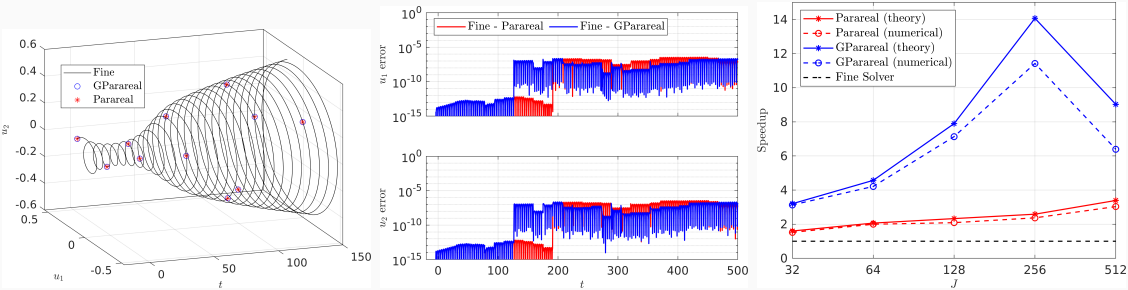
Rössler system: Discussion

- In this experiment, rather than obtaining legacy data by solving (10) using alternative initial values, we instead generate such data by integrating over a shorter time interval. This is particularly useful if we are unsure how long to integrate our system for, i.e. to reach some long-time equilibrium state or reveal certain dynamics of the system, as is the case in many real-world dynamical systems.
- The legacy simulation, integrating over $[0, 170]$, takes nine iterations to converge using GParareal (ten for parareal), giving us approximately $kJ^{(2)} = 9 \times 20 = 180$ legacy evaluations of $\mathcal{F} - \mathcal{G}$ (results not shown).
  Integrating (10) over the full interval $[0, 340]$, GParareal converges in four iterations sooner with the legacy data than without — refer to Figure 4(b). In Figure 4(c) we can see that using the legacy data achieves a higher numerical speedup ($3.4\times$) compared to parareal ($1.6\times$).
- Figure 4(a) illustrates GParareal retaining a similar numerical accuracy to parareal with and without the legacy data. Note the steadily increasing errors for both algorithms is due to the chaotic nature of the Rössler system.

Nonautonomous system: Solutions

Consider the nonautonomous system given by

$$\frac{\mathrm{d}u_1}{\mathrm{d}t} = -u_2 + u_1\left(\frac{t}{500} - u_1^2 - u_2^2\right), \quad \frac{\mathrm{d}u_2}{\mathrm{d}t} = u_1 + u_2\left(\frac{t}{500} - u_1^2 - u_2^2\right), \quad t \in [-20, 500].$$



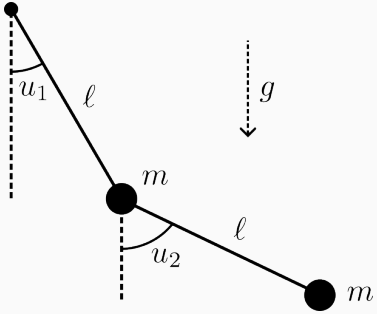
(a) **(b)** **(c)**

Figure 5: Numerical results obtained solving the nonautonomous system over $t \in [-20, 500]$.

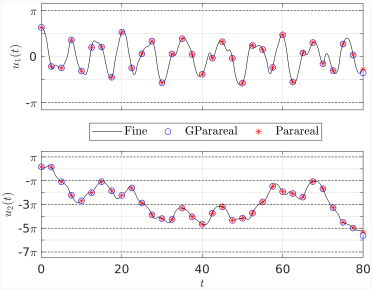
Double Pendulum System: Solutions

Consider the nonautonomous system given by

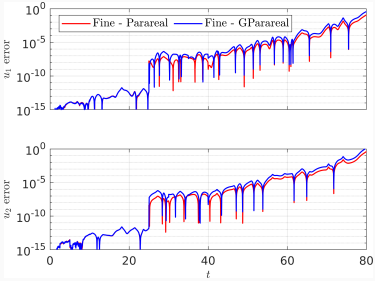
$$\frac{\mathrm{d}u_1}{\mathrm{d}t} = -u_2 + u_1\left(\frac{t}{500} - u_1^2 - u_2^2\right), \quad \frac{\mathrm{d}u_2}{\mathrm{d}t} = u_1 + u_2\left(\frac{t}{500} - u_1^2 - u_2^2\right), \quad t \in [-20, 500].$$



(a)



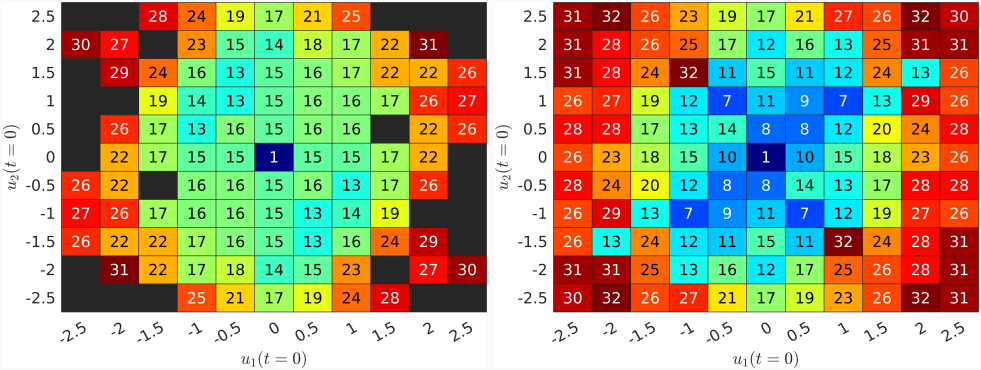
(b)



(c)

Figure 6: Numerical results obtained solving the double pendulum system $t \in [0, 80]$.

Double Pendulum System: Convergence



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

Figure 7: Iterations count k (max. $J = 32$) for various initial angles.

Double Pendulum System: Convergence

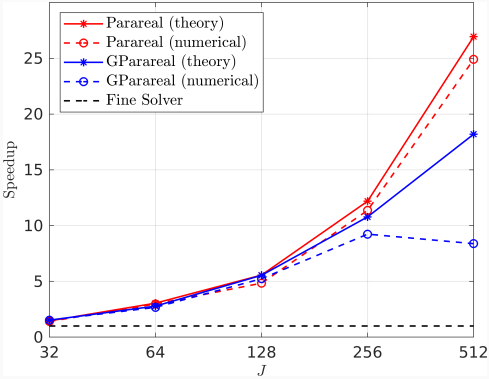


Figure 8: Speedup plot.

Summary

Summary

We presented GPararealal, a PinT algorithm that uses GP emulation to solve (low-dimensional) IVPs in parallel.

- can converge in fewer iterations $\rightarrow$ lower wallclock time.
- solutions accurate wrt parareal.
- can use legacy solution data (from previous solve or uniform grids in $\mathbb{R}^d$).
- can solve problems that parareal fails to converge for.

Other results (see paper for full details!):

- Training GP comes at cost (T_{GP}) which must be small compared to F solve (complexity analysis + experiments in paper show this).

$$T_{\text{GPara}} \approx \underbrace{kT_{\mathcal{F}} + (k+1)(J - k/2)T_{\mathcal{G}}}_{T_{\text{Para}}} + T_{\text{GP}}.$$

- Convergence result shows errors at iteration k bounded by accuracy of emulator:

$$|u(t_j) - U_j^k| \leqslant \Lambda_k \sum_{i=0}^{j-(k+1)} A^i \quad 1 \leqslant k < j \leqslant J.$$

Open problems

- Can better/faster ML/PN methods be used to learn $\mathcal{F} - \mathcal{G} \colon \mathbb{R}^d \to \mathbb{R}^d$? GPs struggle with high-dimensional functions and so we need an alternative method to solve PDEs (work in progress).
- Similar issue wrt the cost of running the GPs $\to$ need to be fast compared to fine solver.
- We currently approximate GP posterior using its mean (ignoring uncertainty). Can we develop a truly probabilistic PinT algorithm? GPararеal is a first positive step in this direction.

Scan the QR code for a link to the paper!

Acknowledgements Funding provided by EPSRC (grant EP/S022244/1), Culham Centre for Fusion Energy, and Euratom (No. 633053). Registration and travel support for this presentation was provided by the Society for Industrial and Applied Mathematics.

Additional results (put complexity, convergence result, numerics here).

References 1

R. FitzHugh. Impulses and physiological states in theoretical models of nerve membrane. *Biophys. J.*, 1:445–466, 1961. doi:10.1016/S0006-3495(61)86902-6.

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

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

C. E. Rasmussen and C. K. I. Williams. *Gaussian processes for machine learning*. Adaptive computation and machine learning. MIT Press, 2006. ISBN 026218253X.

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