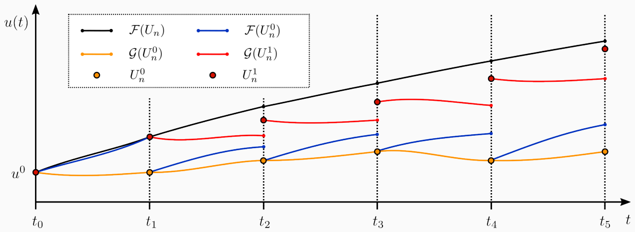
Towards probabilistic time-parallel algorithms for solving initial value problems

Kamran Pentland

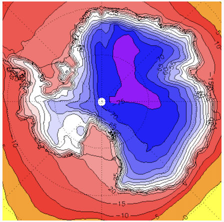


Pre-Viva Talk
MathSys
University of Warwick
22 November 2023

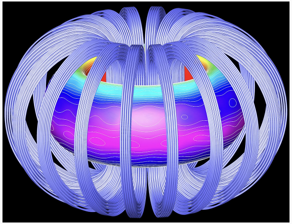
The parallel-in-time (PinT) problem

Motivation and setup

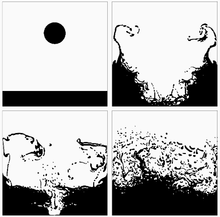
Initial value problems (IVPs) exist all around us:



(a) Weather models



(b) Plasma simulation



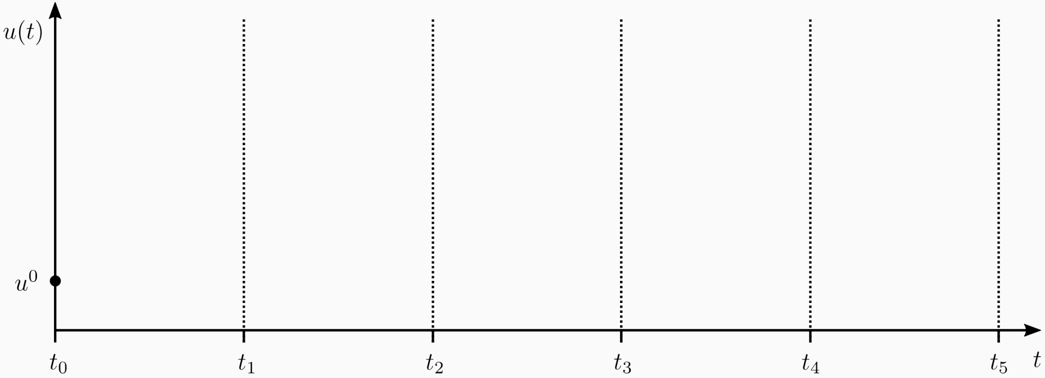
(c) Fluid mechanics

Typically boil down to calculating numerical solutions $\boldsymbol{U}_n \approx \boldsymbol{u}(t_n)$ to

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

on a mesh $\boldsymbol{t} = (t_0, \dots, t_N)$, where $t_{n+1} = t_n + \Delta T$ for fixed $\Delta T = (T - t_0)/N$.

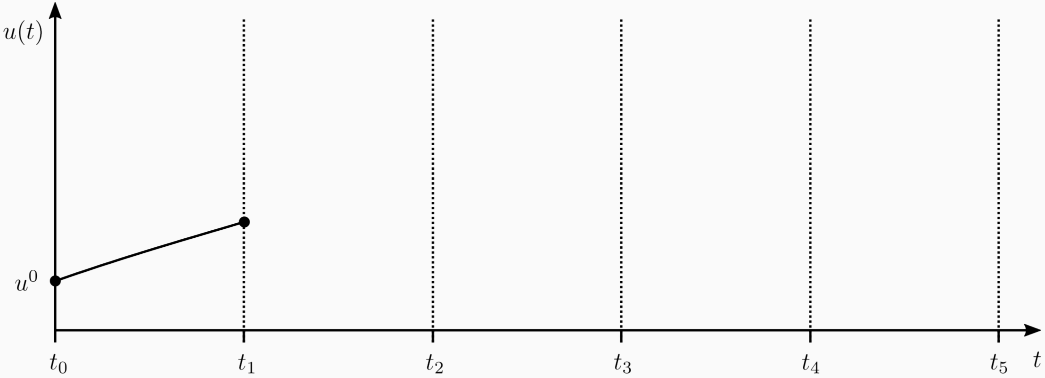
The PinT problem



Suppose we have access to an expensive high accuracy fine solver ($\mathcal{F}$) (e.g. Runge-Kutta).

Aim: Calculate numerical solutions $U_{n+1} = \mathcal{F}(U_n)$ sequentially using one processor.

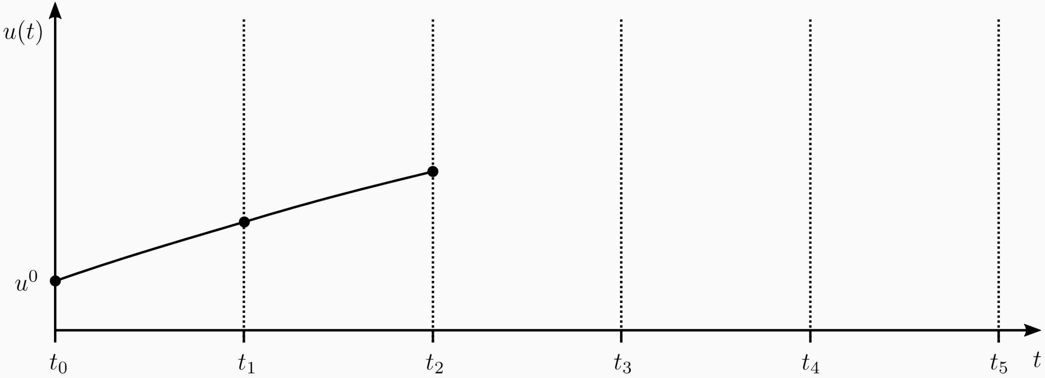
The PinT problem



Suppose we have access to an expensive high accuracy fine solver ($\mathcal{F}$) (e.g. Runge-Kutta).

Aim: Calculate numerical solutions $U_{n+1} = \mathcal{F}(U_n)$ sequentially using one processor.

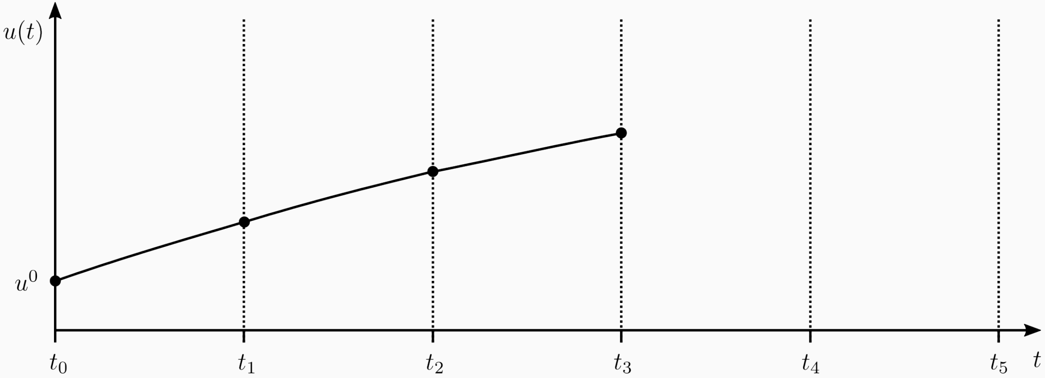
The PinT problem



Suppose we have access to an expensive high accuracy fine solver ($\mathcal{F}$) (e.g. Runge-Kutta).

Aim: Calculate numerical solutions $U_{n+1} = \mathcal{F}(U_n)$ sequentially using one processor.

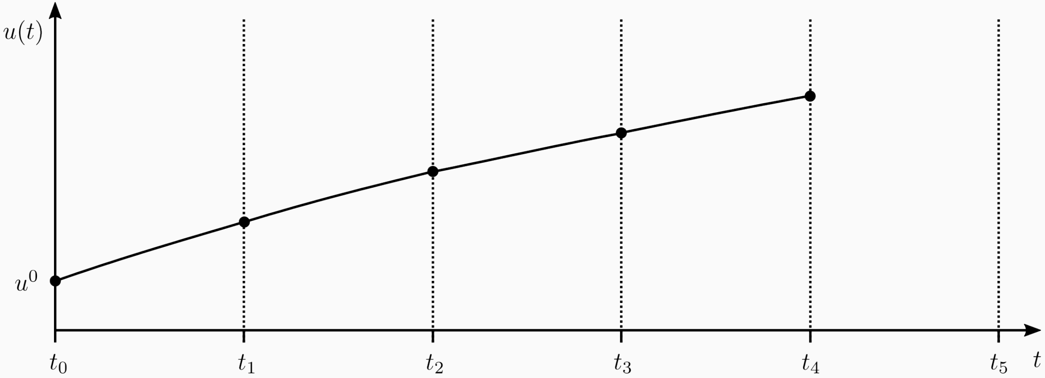
The PinT problem



Suppose we have access to an expensive high accuracy fine solver ($\mathcal{F}$) (e.g. Runge-Kutta).

Aim: Calculate numerical solutions $U_{n+1} = \mathcal{F}(U_n)$ sequentially using one processor.

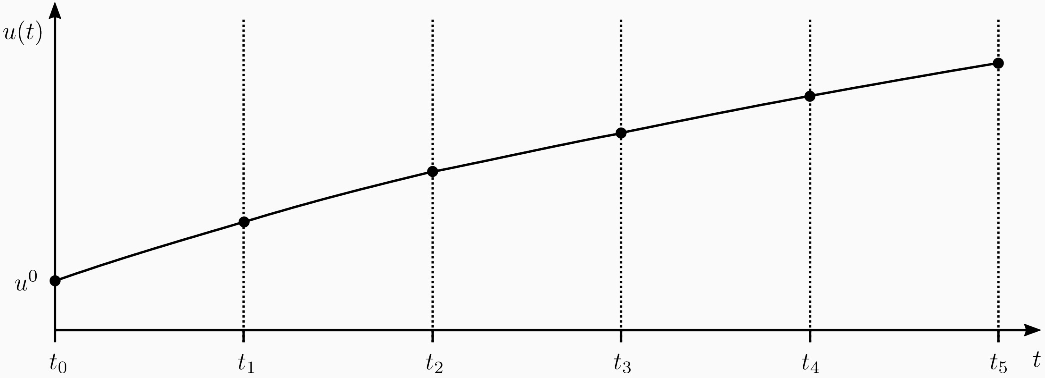
The PinT problem



Suppose we have access to an expensive high accuracy fine solver ($\mathcal{F}$) (e.g. Runge-Kutta).

Aim: Calculate numerical solutions $U_{n+1} = \mathcal{F}(U_n)$ sequentially using one processor.

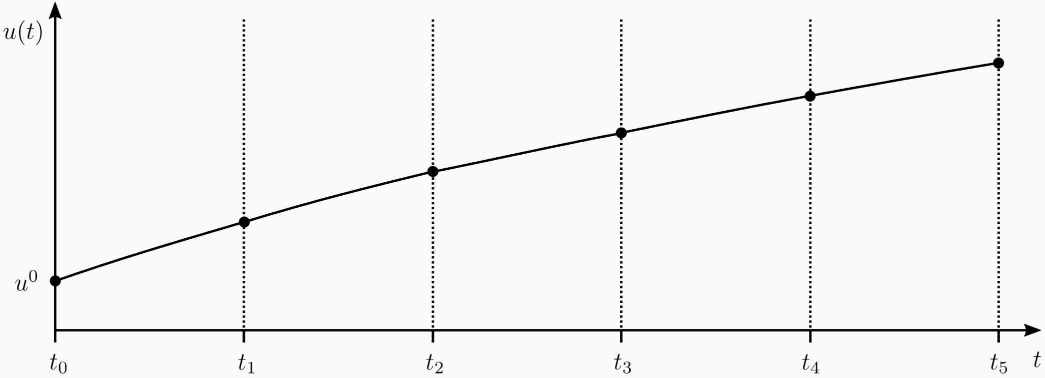
The PinT problem



Suppose we have access to an expensive high accuracy fine solver $(\mathcal{F})$ (e.g. Runge-Kutta).

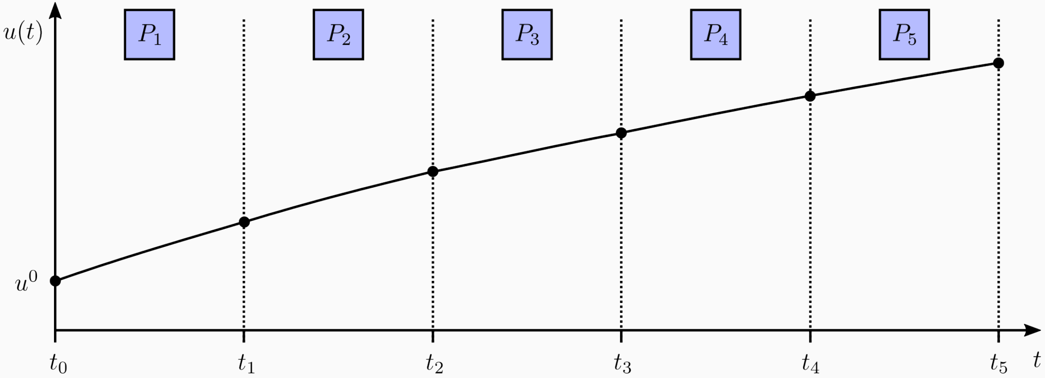
Aim: Calculate numerical solutions $U_{n+1} = \mathcal{F}(U_n)$ sequentially using one processor.

The PinT problem



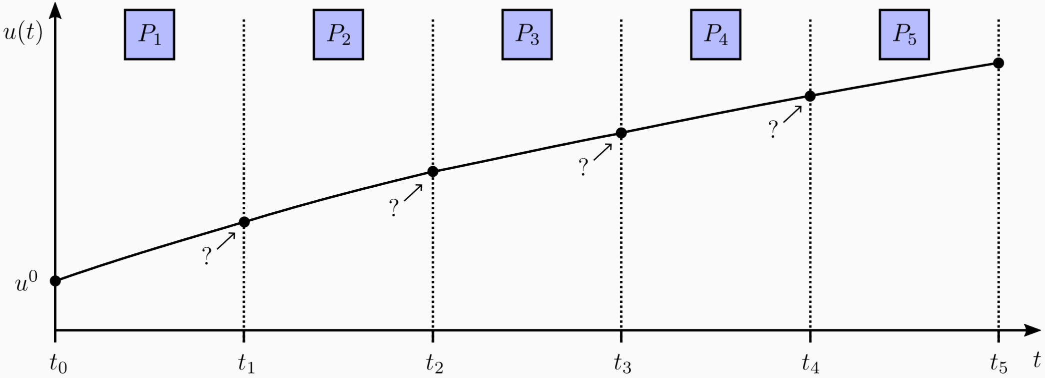
Problem 1: This sequential task could be taking $\mathcal{O}(10^0) - \mathcal{O}(10^6)$ seconds (i.e. up to minutes, hours, or even days!)

The PinT problem



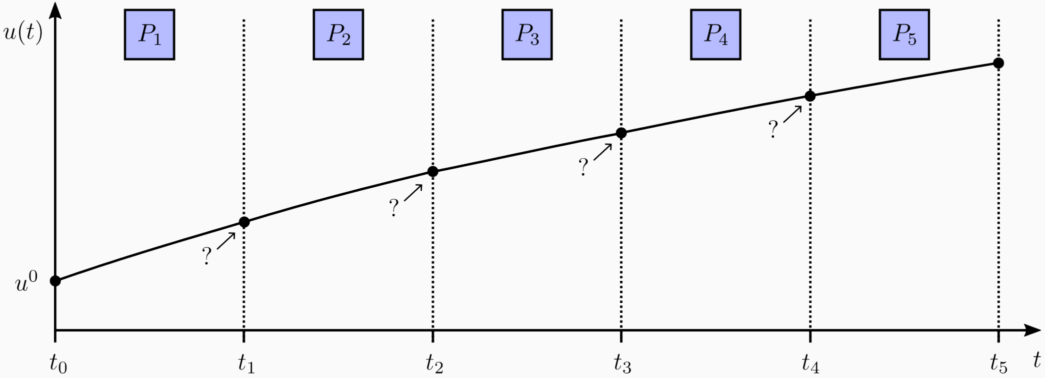
Original idea: Split into smaller IVPs, solve each with its own processor (Nievergelt, 1964).

The PinT problem



Problem 2: How do we solve these smaller IVPs *in parallel* without the unknown initial values?

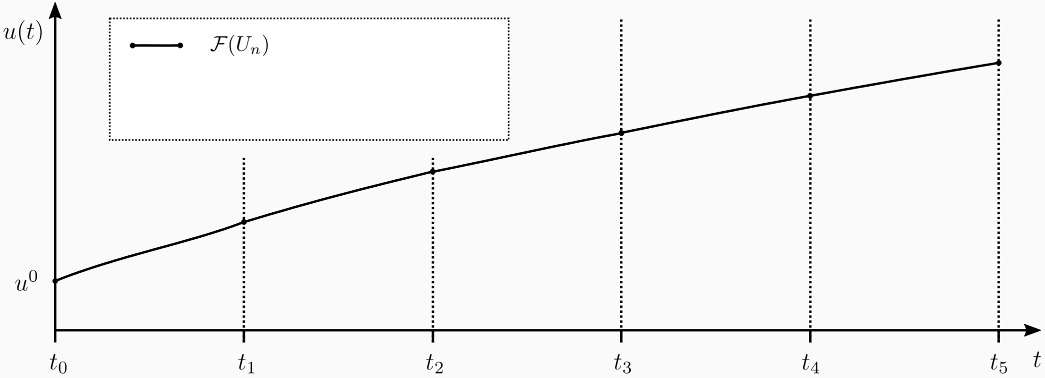
The PinT problem



Solution: Use a PinT scheme! $\rightarrow$ e.g. ParaDiag, PFASST, MGRIT, Parareal.

Parareal: an existing PinT algorithm

Parareal: how it works

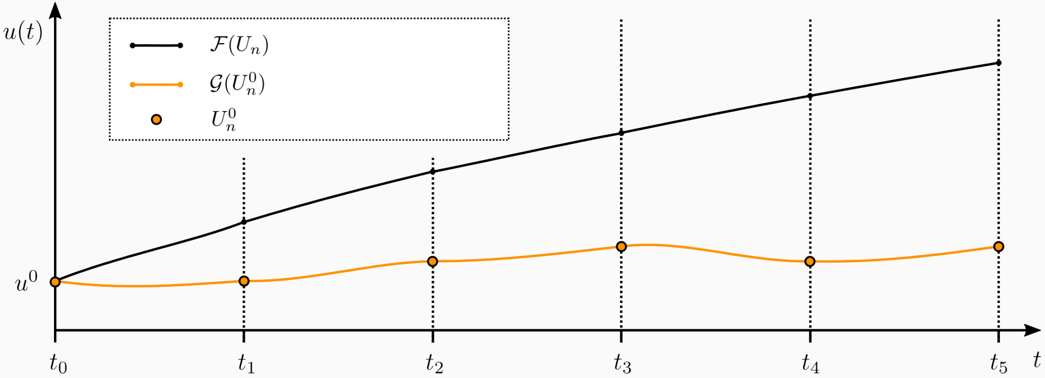


Who: Developed by (Lions et al., 2001) $\rightarrow$ much research into it since then.

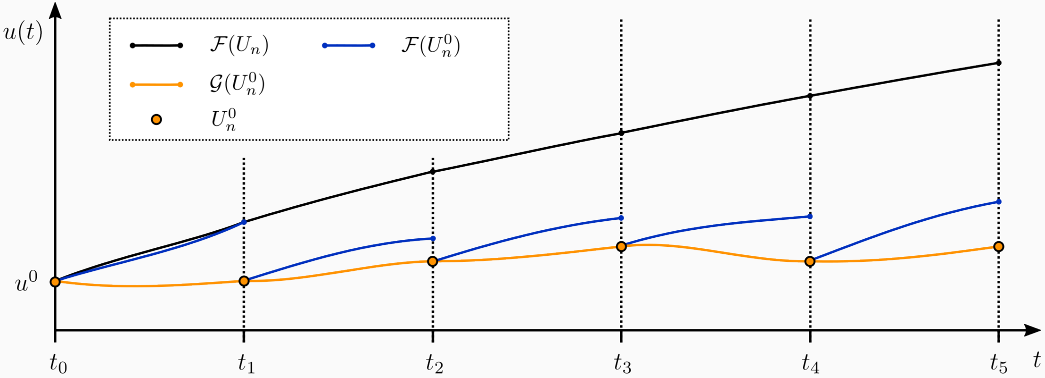
Pros: Flexible, easy-to-use, generates data in favourable way...

How: Utilise N processors and a cheap low accuracy coarse solver $\mathcal{G}$.

Parareal: how it works

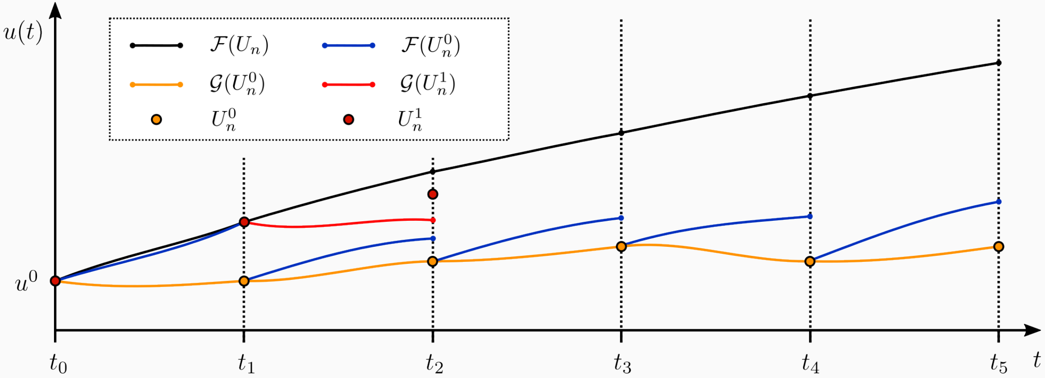


Step 1: Run $\mathcal{G}$ serially (yellow).



Step 2: Using these values, run $\mathcal{F}$ *in parallel* (blue).

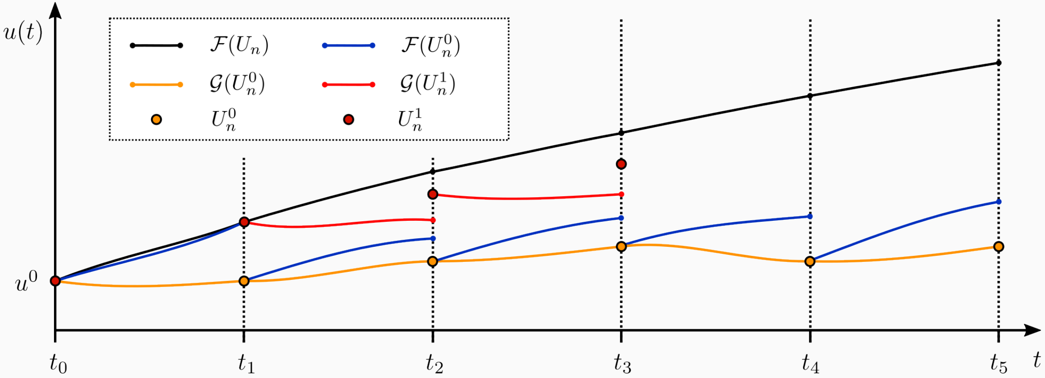
Parareal: how it works



Step 3: Predict with $\mathcal{G}$ (red) and correct using difference of previous $\mathcal{F}$ and $\mathcal{G}$:

$$U_{n+1}^k = \underbrace{\mathcal{G}(U_n^k)}_{\text{predict}} + \underbrace{\mathcal{F}(U_n^{k-1}) - \mathcal{G}(U_n^{k-1})}_{\text{correct}}.$$

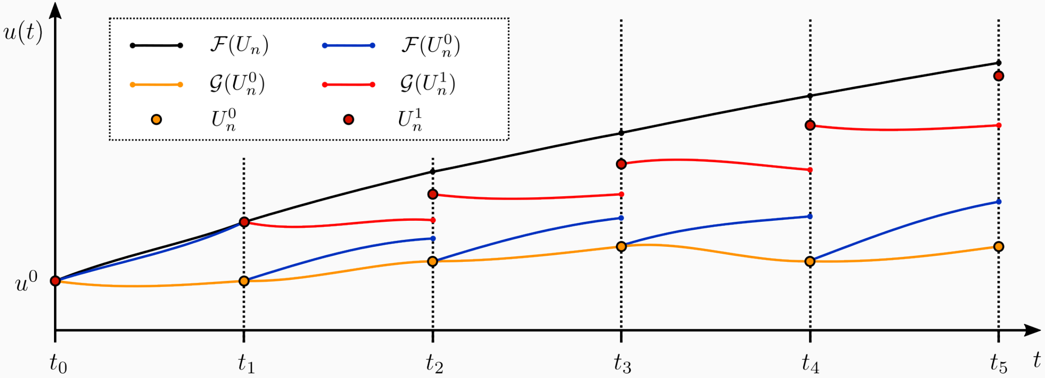
Parareal: how it works



Step 3: Predict with $\mathcal{G}$ (red) and correct using difference of previous $\mathcal{F}$ and $\mathcal{G}$:

$$U_{n+1}^k = \underbrace{\mathcal{G}(U_n^k)}_{\text{predict}} + \underbrace{\mathcal{F}(U_n^{k-1}) - \mathcal{G}(U_n^{k-1})}_{\text{correct}}.$$

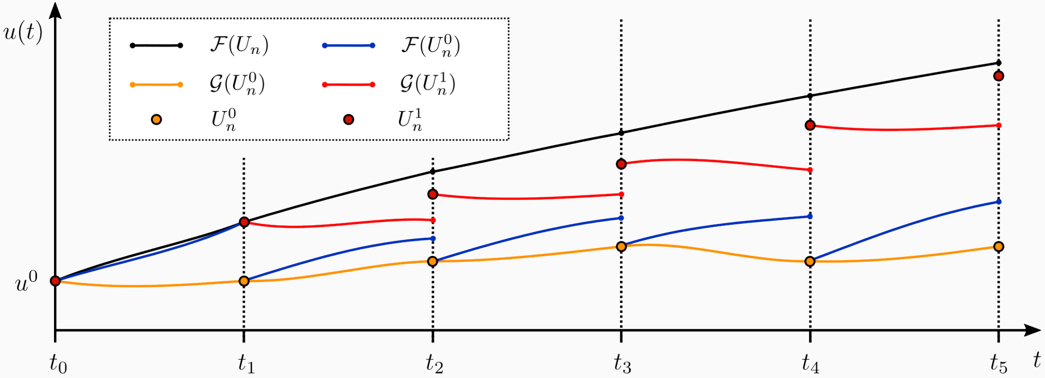
Parareal: how it works



Step 3: Predict with $\mathcal{G}$ (red) and correct using difference of previous $\mathcal{F}$ and $\mathcal{G}$:

$$U_{n+1}^k = \underbrace{\mathcal{G}(U_n^k)}_{\text{predict}} + \underbrace{\mathcal{F}(U_n^{k-1}) - \mathcal{G}(U_n^{k-1})}_{\text{correct}}.$$

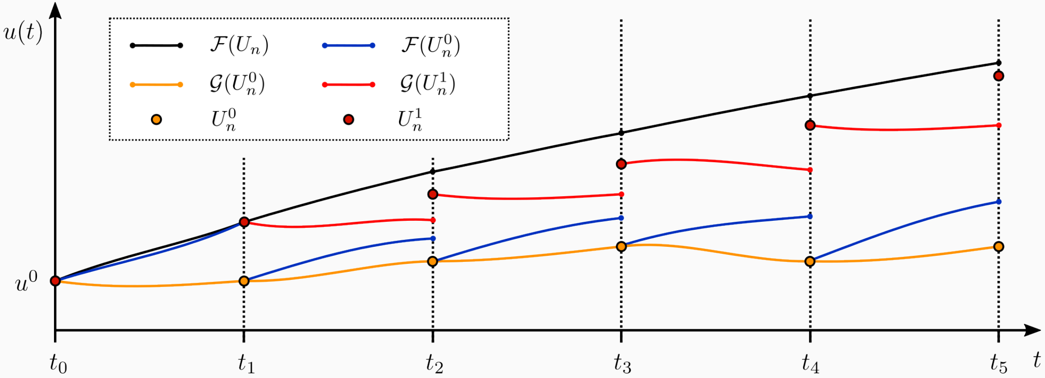
Parareal: how it works



Step 4: Repeat steps 2 and 3 until tolerance met:

$$|U_n^k - U_n^{k-1}| < \varepsilon \quad \forall n \leqslant N.$$

Parareal: how it works



Key point: Parareal stops in $k \leqslant N$ iterations $\rightarrow$ maximal speedup $= N/k$.

(n.b. scales to the multivariate case easily.)

So what did I actually do?

Thesis overview

Main research question: Can we accelerate convergence (i.e. reduce k) by making better use of the discarded ($\mathcal{F}$ and $\mathcal{G}$) solution data in Parareal?

$$U_{n+1}^{k} = \underbrace{\mathcal{G}(U_n^k)}_{\text{predict}} + \underbrace{\mathcal{F}(U_n^{k-1}) - \mathcal{G}(U_n^{k-1})}_{\text{correct}} \;\rightarrow\; \underbrace{\{U_n^k, \mathcal{F}(U_n^k), \mathcal{G}(U_n^k)\}}_{\text{Lots of data discarded!}}{}_{0\leqslant k,n \leqslant N}$$

Recall $\rightarrow$ lower k = higher speedup.

Thesis overview

Main research question: Can we accelerate convergence (i.e. reduce k) by making better use of the discarded ($\mathcal{F}$ and $\mathcal{G}$) solution data in Parareal?

$$U_{n+1}^{k} = \underbrace{\mathcal{G}(U_n^k)}_{\text{predict}} + \underbrace{\mathcal{F}(U_n^{k-1}) - \mathcal{G}(U_n^{k-1})}_{\text{correct}} \rightarrow \underbrace{\{U_n^k, \mathcal{F}(U_n^k), \mathcal{G}(U_n^k)\}}_{\text{Lots of data discarded!}}{}_{0\leqslant k,n\leqslant N}$$

Recall $\rightarrow$ lower k = higher speedup.

How? $\rightarrow$ harness data with sampling- and learning-based probabilistic methods.

Thesis overview

Main research question: Can we accelerate convergence (i.e. reduce k) by making better use of the discarded ($\mathcal{F}$ and $\mathcal{G}$) solution data in Parareal?

$$U_{n+1}^{k} = \underbrace{\mathcal{G}(U_n^k)}_{\text{predict}} + \underbrace{\mathcal{F}(U_n^{k-1}) - \mathcal{G}(U_n^{k-1})}_{\text{correct}} \;\rightarrow\; \underbrace{\{U_n^k, \mathcal{F}(U_n^k), \mathcal{G}(U_n^k)\}}_{\text{Lots of data discarded!}}{}_{0 \leqslant k,n \leqslant N}$$

Recall $\rightarrow$ lower k = higher speedup.

How? $\rightarrow$ harness data with sampling- and learning-based probabilistic methods.

Our aims:

I. derive our own probabilistic PinT algorithms.
II. prove convergence analytically/numerically.
III. demonstrate speedup and scalability vs. Parareal.
IV. generate probabilistic solutions.

Thesis overview

Main research question: Can we accelerate convergence (i.e. reduce k) by making better use of the discarded ($\mathcal{F}$ and $\mathcal{G}$) solution data in Parareal?

$$U_{n+1}^k = \underbrace{\mathcal{G}(U_n^k)}_{\text{predict}} + \underbrace{\mathcal{F}(U_n^{k-1}) - \mathcal{G}(U_n^{k-1})}_{\text{correct}} \;\rightarrow\; \underbrace{\{U_n^k, \mathcal{F}(U_n^k), \mathcal{G}(U_n^k)\}}_{\text{Lots of data discarded!}}{}_{0\leqslant k,n\leqslant N}$$

Recall $\rightarrow$ lower k = higher speedup.

How? $\rightarrow$ harness data with sampling- and learning-based probabilistic methods.

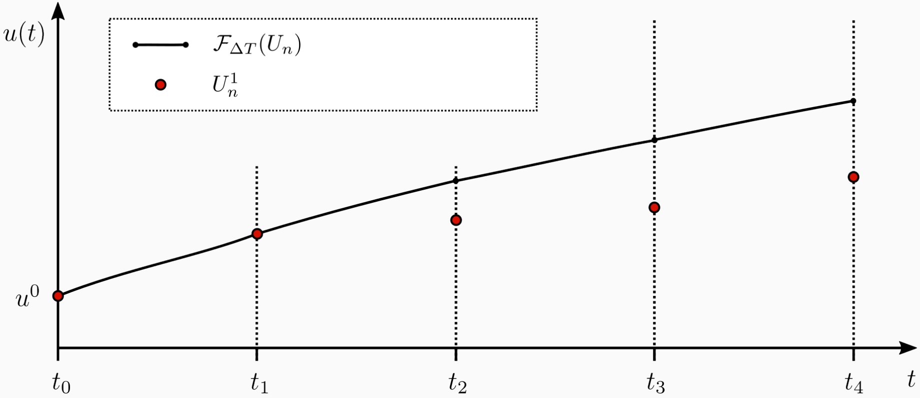
Our aims:

I. derive our own probabilistic PinT algorithms.
II. prove convergence analytically/numerically.
III. demonstrate speedup and scalability vs. Parareal.
IV. generate probabilistic solutions.

Outcome: we developed two algorithms $\rightarrow$ SParareal and GParareal.

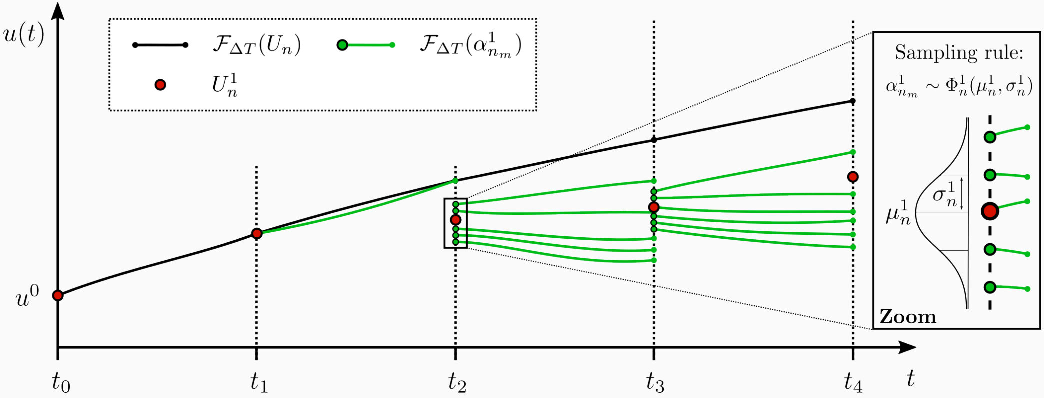
SParareal: a sampling-based PinT algorithm

SParareal: the idea



To begin: Run the first iteration of Parareal.

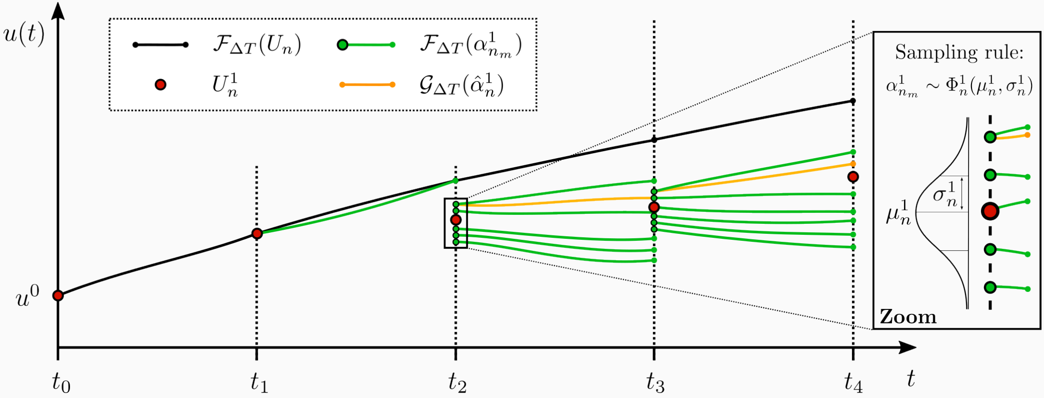
SParareal: the idea



New idea: Sample M solutions from prob. distributions constructed using $\mathcal{F}$ and $\mathcal{G}$ data:

$$\alpha_n^{k-1} \sim \mathcal{N}(U_n^{k-1}, (\mathcal{G}(U_{n-1}^k) - \mathcal{G}(U_{n-1}^{k-1}))^2) \ \rightarrow \ \text{propagate all with } \mathcal{F} \text{ in parallel.}$$

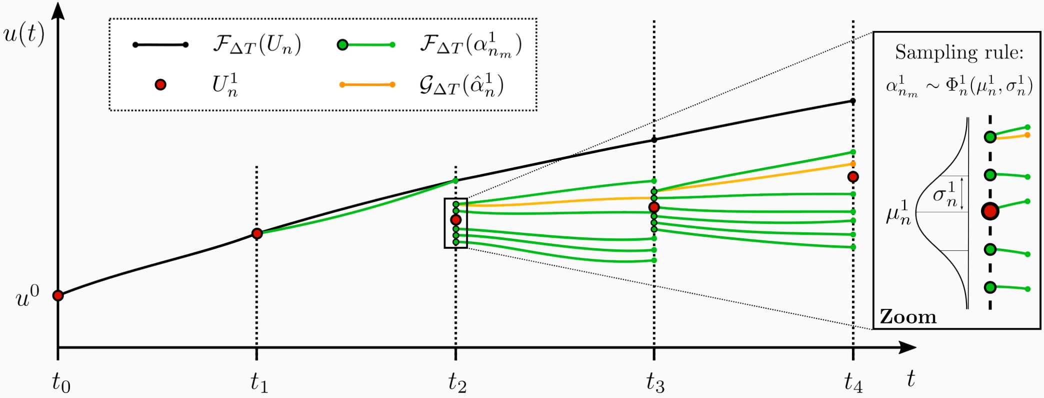
SParareal: the idea



Select "smoothest" trajectory over $[t_0, t_4]$ and use modified PC:

$$U^k_{n+1} = \underbrace{\mathcal{G}(U^k_n)}_{\text{predict}} + \underbrace{\mathcal{F}(\hat{\alpha}^{k-1}_n) - \mathcal{G}(\hat{\alpha}^{k-1}_n)}_{\text{new correction}}$$

SParareal: the idea

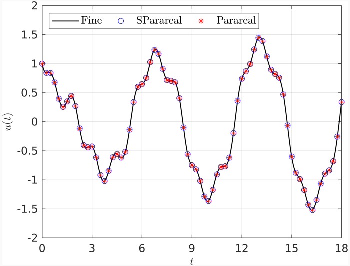


Key aim: Explore solution space more than Parareal can.

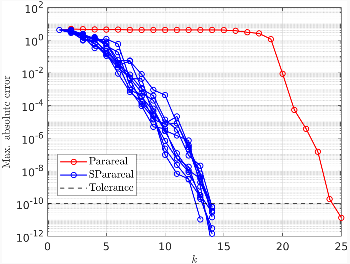
(Different "sampling rules" also available!)

SParareal: results summary

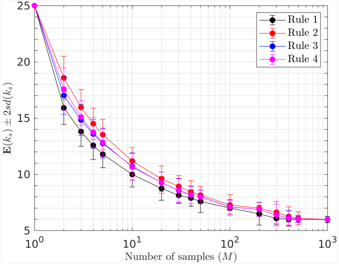
1D system : $$\frac{\mathrm{d}u}{\mathrm{d}t} = \sin(u)\cos(u) - 2u + e^{-t/100}\sin(5t) + \ln(1+t)\cos(t), \quad t \in [0, 100].$$



(a) Solution



(b) Convergence ($M = 3$)

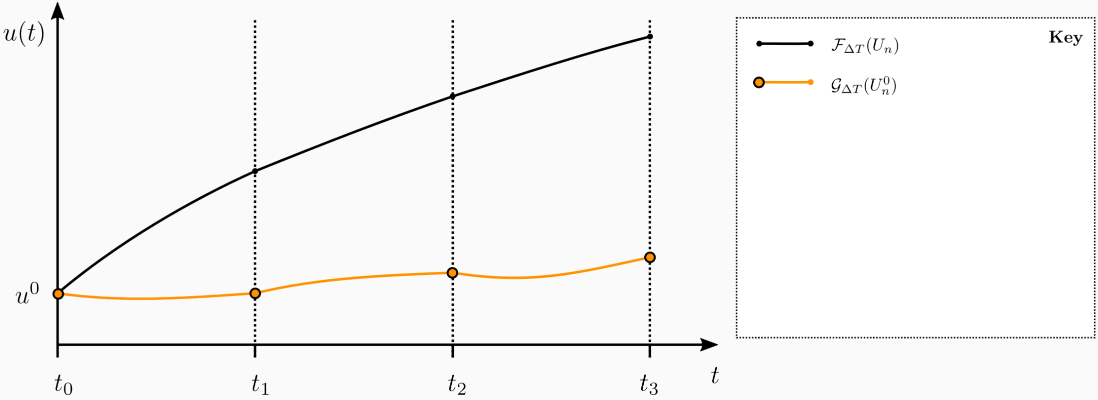


(c) Expected convergence

- Pro: Converges in fewer iterations than Parareal for increasing M.
- Pro: Returns (stochastic) probabilistic solutions → errors $\max_n \mathbb{E}\left[\|\boldsymbol{u}(t_n) - \boldsymbol{U}_n^k\|^2\right]$ bounded.
- Pro: Similar results for systems of ODEs.
- Con: Requires $\mathcal{O}(MN)$ processors vs. N in Parareal.

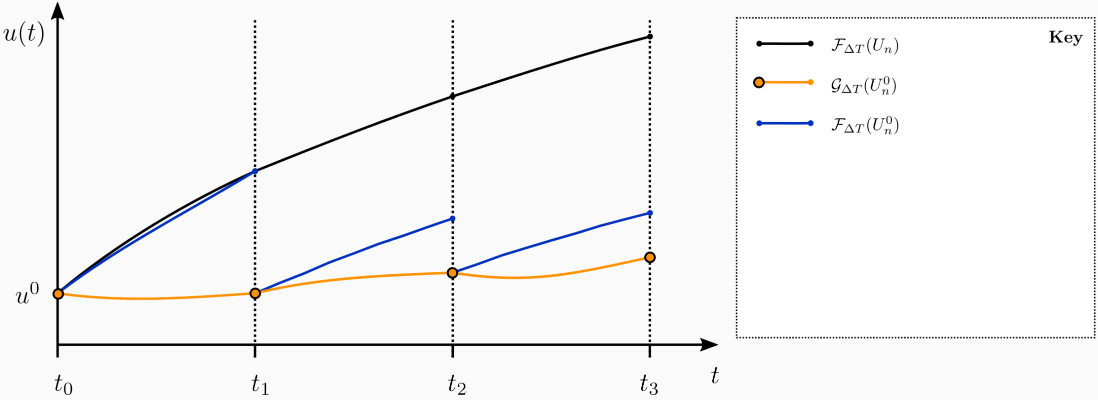
GParareal: a learning-based PinT algorithm

GParareal: the idea



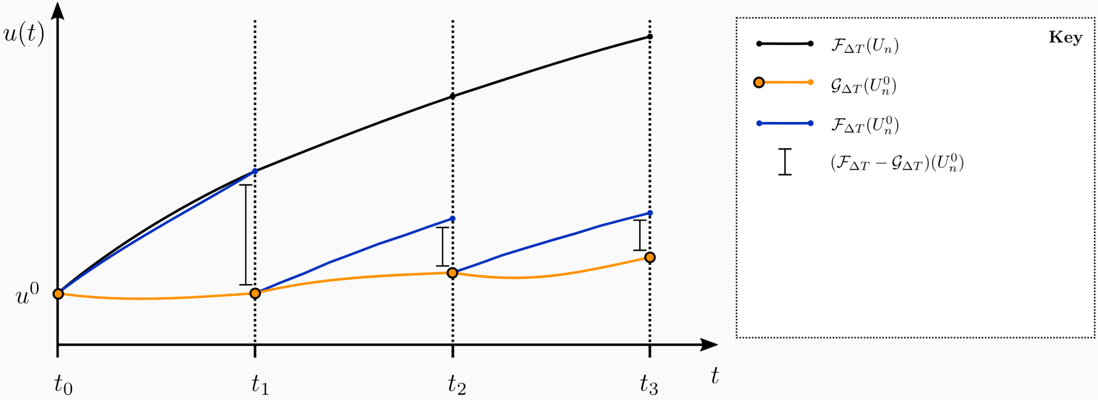
To begin: As before, run $\mathcal{G}$ and then $\mathcal{F}$.

GParareal: the idea



To begin: As before, run $\mathcal{G}$ and then $\mathcal{F}$.

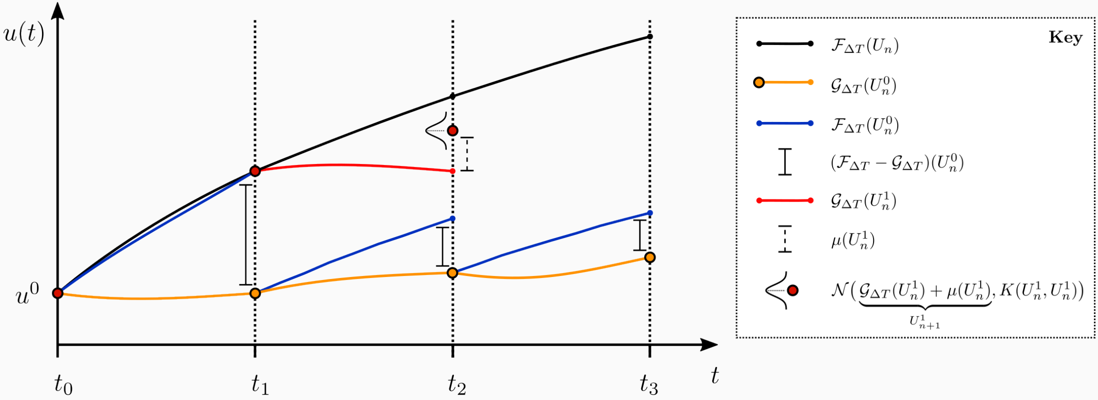
GParareal: the idea



New idea: Use *all* prior $\mathcal{F}$ and $\mathcal{G}$ data to train a GP emulator:

$$(\mathcal{F} - \mathcal{G})(U_n^k) \mid \{(\mathcal{F} - \mathcal{G})(U_n^0), U_n^0\}_{n=0}^{N-1} \sim \mathcal{N}\big(\hat{\mu}(U_n^k), \hat{K}(U_n^k, U_n^k)\big).$$

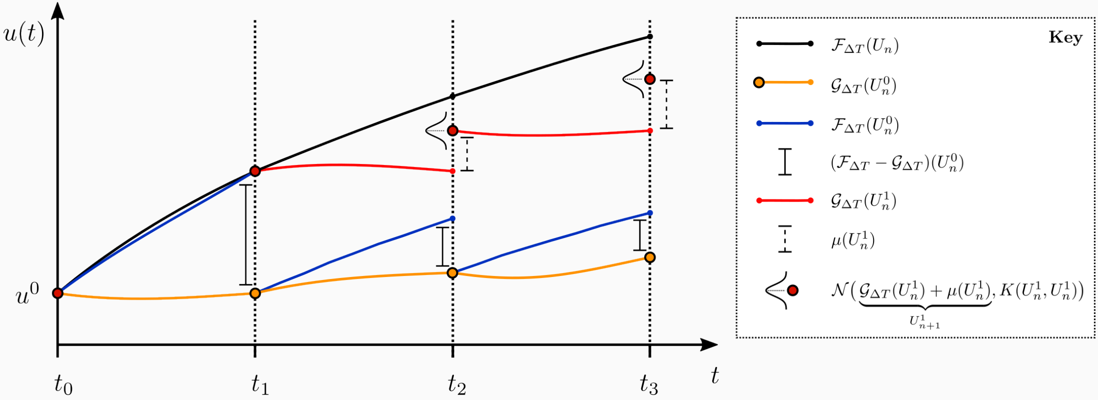
GParareal: the idea



Modified PC: approximate Gaussian by its mean value

$$U_{n+1}^k = \underbrace{\mathcal{G}(U_n^k)}_{\text{prediction}} + \underbrace{(\mathcal{F} - \mathcal{G})(U_n^k)}_{\text{new correction}} \approx \mathcal{G}(U_n^k) + \hat{\mu}(U_n^k)$$

GParareal: the idea

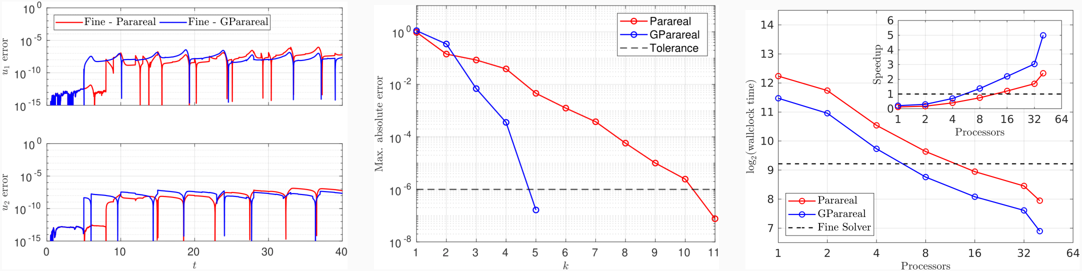


Modified PC: approximate Gaussian by its mean value

$$U_{n+1}^k = \underbrace{\mathcal{G}(U_n^k)}_{\text{prediction}} + \underbrace{(\mathcal{F} - \mathcal{G})(U_n^k)}_{\text{new correction}} \approx \mathcal{G}(U_n^k) + \hat{\mu}(U_n^k)$$

GParareal: results summary

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



(a) Errors **(b)** Convergence **(c)** Speedup

- Pro: Good accuracy and half the iterations $\Rightarrow$ twice the speedup.
- Pro: Error bound $|u(t_n) - U_n^k| \leqslant \Lambda_k \sum_{i=0}^{n-(k+1)} A^i \qquad 1 \leqslant k < n \leqslant N.$
- Pro: Can re-use old simulation (legacy) $\rightarrow$ obtain more speedup.
- Con: GP training time needs to be small, else speedup lost!

So what did we find?

Wrapping up

Many more results (e.g. ODE/PDE systems, speedup tests, convergence/complexity analysis).

Original aims and open problems:

I. derive our own probabilistic PinT algorithms (✓).
   - harnessed data using prob. methods to accelerate convergence.

II. prove convergence analytically/numerically (✓).

III. demonstrate speedup and scalability vs. Parareal (speedup ✓, scalability ✗).
   - SParareal hindered by CoD/processor counts → weighted samples?
   - GParareal hindered by GP training time → use "best" data?

IV. generate probabilistic solutions (SParareal ✓, GParareal ✗)
   - Unsure how to interpret uncertainty from SParareal solutions.
   - Could sample solutions in GParareal → untested.

Acknowledgements

Massimiliano Tamborrino

Tim Sullivan

Lynton Appel (CCFE)

James Buchanan (CCFE)

Debasmita Samaddar (ex. CCFE)

Huge thanks to everyone for their help (and those who I've forgotten):

- MathSys management team.
- Everyone else who made it such a great time.
- Funders: EPSRC, Culham Centre for Fusion Energy (CCFE), and Euratom.

Thanks for listening, any questions?

Additional results

Parareal: complexity

Wallclock time:

$$T_{\text{para}} \approx \underbrace{N T_{\mathcal{G}}}_{\text{Iteration } 0} + \underbrace{\sum_{i=1}^{k} \big(T_{\mathcal{F}} + (N-i) T_{\mathcal{G}} \big)}_{\text{Iterations } 1 \text{ to } k}$$

$$= k T_{\mathcal{F}} + (k+1)\big(N - \frac{k}{2}\big) T_{\mathcal{G}}.$$

Parallel speedup:

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

Parallel efficiency:

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

SPararreal: complexity

Wallclock time

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

Note: summation term includes additional cost of running $\mathcal{G}$ for optimal samples (as well as runs of carried out in PC step).

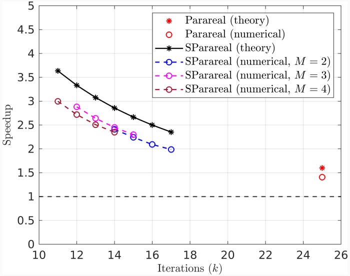
Parallel speedup:

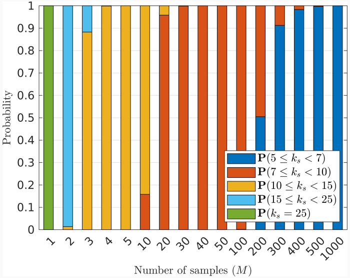
$$S_{\text{SPara}} \approx \frac{T_{\text{serial}}}{T_{\text{SPara}}} = \left[\frac{k}{N} + \left(2k - \frac{k}{N}(k+1) + \frac{1}{N}\right)\frac{T_{\mathcal{G}}}{T_{\mathcal{F}}}\right]^{-1},$$

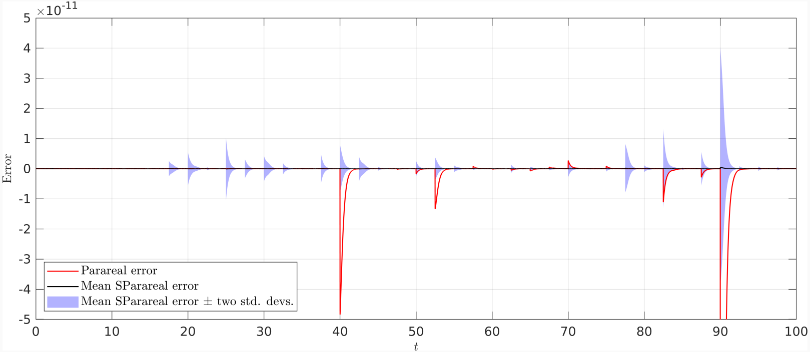
Parallel efficiency:

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

SParareal: further results





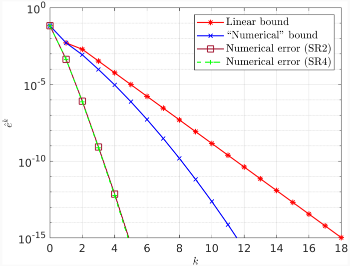


Theorem 1 (Linear error bound for sampling rules)

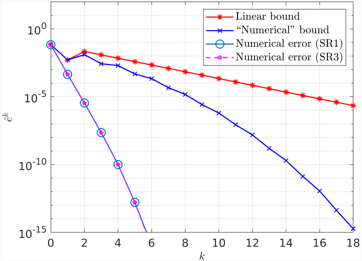
Suppose SParareal satisfies assumptions 1-3 (see thesis). Then, the maximal mean-square error of the solution to a nonlinear ODE system satisfies

$$\max_n \mathbb{E}\left[\|\boldsymbol{u}(t_n) - \boldsymbol{U}_n^k\|^2\right] \leqslant \hat{e}^0 \left[\frac{A + \Lambda_1 + \sqrt{(A+\Lambda_1)^2 + 4\Lambda_2(1-B)}}{2(1-B)}\right]^k, \quad \text{if} \quad B < 1,$$

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



(a) Rules 2 and 4



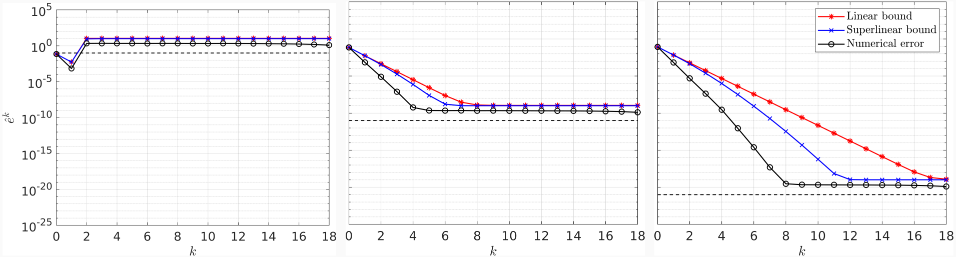
(b) Rules 1 and 3

Theorem 2 (Superlinear error bound for state-independent perturbations)

Suppose the SParareal scheme satisfies assumptions 1-4 (see thesis). Then, the mean-square error of the solution to a nonlinear ODE system at iteration k and time t_n satisfies

$$\mathbb{E}\left[\|\boldsymbol{u}(t_n) - \boldsymbol{U}_n^k\|^2\right] \leqslant DA^{k-1}\sum_{\ell=0}^{n-k}\binom{\ell+k-1}{\ell}B^\ell + \Lambda\sum_{j=0}^{k-2}\sum_{\ell=0}^{n-(j+1)}\binom{\ell+j}{\ell}A^j B^\ell,$$

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



GParareal: complexity

Wallclock time:

$$T_{\text{GPara}} \approx NT_{\mathcal{G}} + \sum_{i=1}^{k}\big(T_{\mathcal{F}} + (N-i)T_{\mathcal{G}} + T_{\text{GP}}(i)\big)$$
$$= kT_{\mathcal{F}} + (k+1)\left(N - \frac{k}{2}\right)T_{\mathcal{G}} + T_{\text{GP}},$$

where $T_{\text{GP}} := \sum_{i=1}^{k} T_{\text{GP}}(i)$.

Parallel speedup:

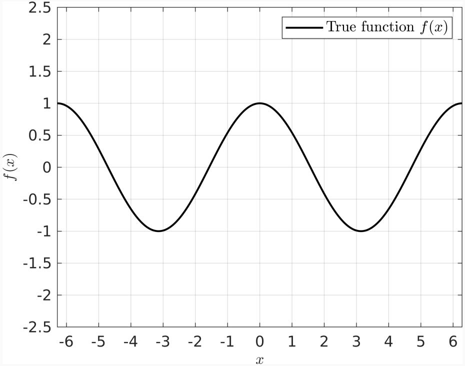
$$S_{\text{GPara}} \approx \left[\frac{k}{N} + (k+1)\left(1 - \frac{k}{2N}\right)\frac{T_{\mathcal{G}}}{T_{\mathcal{F}}} + \frac{1}{N}\frac{T_{\text{GP}}}{T_{\mathcal{F}}}\right]^{-1}.$$

Parallel efficiency:

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

GParareal: what is a GP emulator?

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

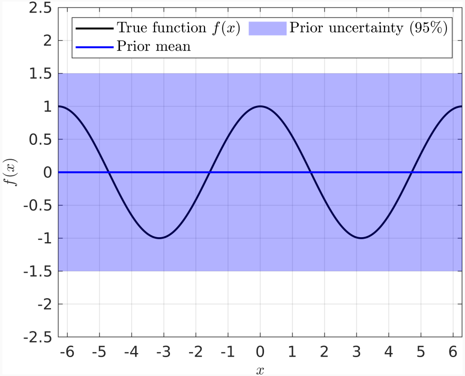


GParareal: what is a GP emulator?

GP emulation: **statistically modelling** 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)

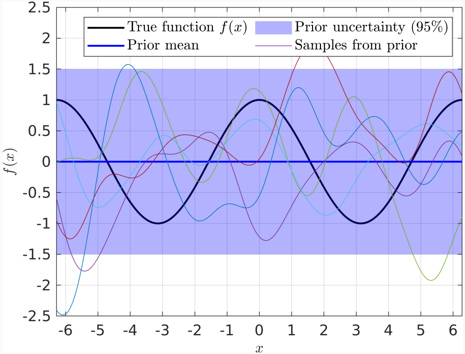
$$f(\boldsymbol{x}) \sim \mathcal{N}\left(\mu(\boldsymbol{x}), K(\boldsymbol{x}, \boldsymbol{x})\right).$$



GP emulation: **statistically modelling** 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)

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

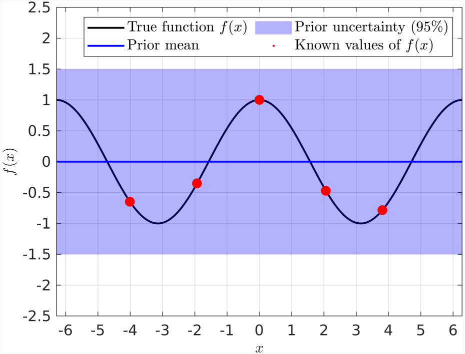


GP emulation: **statistically modelling** an **unknown (expensive-to-evaluate) function** using multivariate **Gaussian** distributions (Rasmussen and Williams, 2006).

- **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): $(\boldsymbol{x}, \boldsymbol{y}) = \big(x_i, f(x_i)\big)_{i=1,\ldots,N}$.



GParareal: what is a GP emulator?

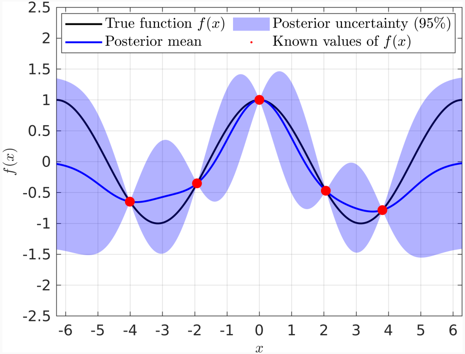
GP emulation: **statistically modelling** an **unknown (expensive-to-evaluate) function** using multivariate **Gaussian** distributions (Rasmussen and Williams, 2006).

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

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

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

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

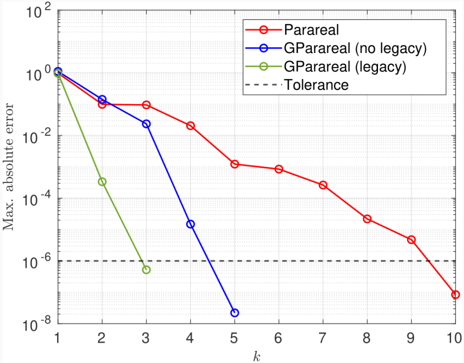


Both $\hat{\mu}(x^*)$ and $\hat{K}(x^*, x^*)$ have nice analytical expressions.

GParareal: results summary

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 $\boldsymbol{u}^0 = (0.75, 0.25)^\intercal$.

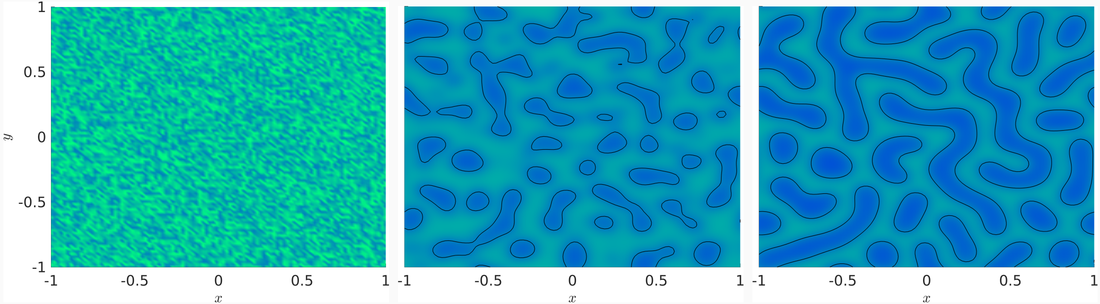


- Takeaway: Re-use $\mathcal{F} - \mathcal{G}$ data in future GParareal simulations to pre-train GP and gain additional speedup.
- Downside: Training time scales with quantity of data!

GParareal: 2D FitzHugh–Nagumo PDE

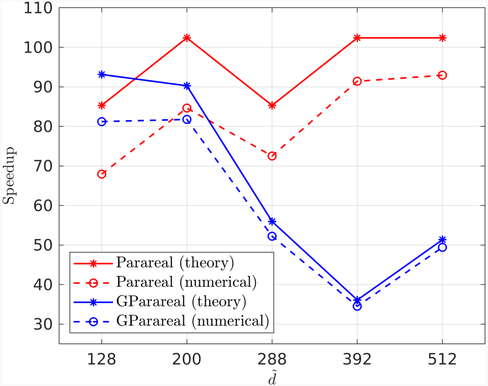
Consider the 2D spatially-dependent FHN model given by

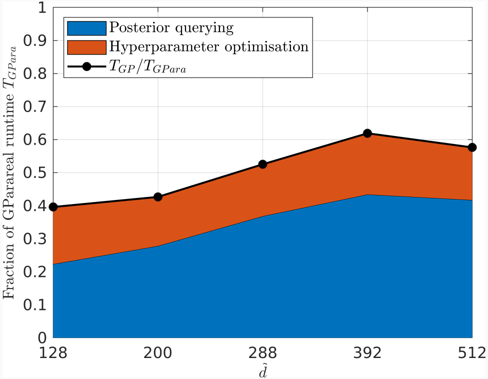
$$v_t = a\nabla^2 v + v - v^3 - w - c, \quad w_t = \tau\left(b\nabla^2 w + v - w\right), \quad (\boldsymbol{x}, t) \in [-1, 1]^2 \times [0, 100].$$



Takeaway: Lots of spatial points $\Rightarrow$ large ODE system to solve.

GParareal: 2D FitzHugh–Nagumo PDE





Takeaway: Cost of emulation hinders scalability.

References 1

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

J. Nievergelt. Parallel methods for integrating ordinary differential equations. *Communications of the ACM*, 7:731–733, 1964. doi:10.1145/355588.365137.

C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning*. Adaptive Computation and Machine Learning. MIT Press, 2006.