RWPararealℓ: a time-parallel PDE solver using Random Weights Neural Networks

Guglielmo Gattiglio
University of Warwick

May 1, 2024

Joint work with:
Lyudmila Grigoryeva, University of St. Gallen
Massimiliano Tamborrino, University of Warwick

Parareal [LMT01].

- Sketch of the procedure
- Computational cost

GParareal [Pen+23].

- Empirical results
- Computational cost

Nearest Neighbor GParareal

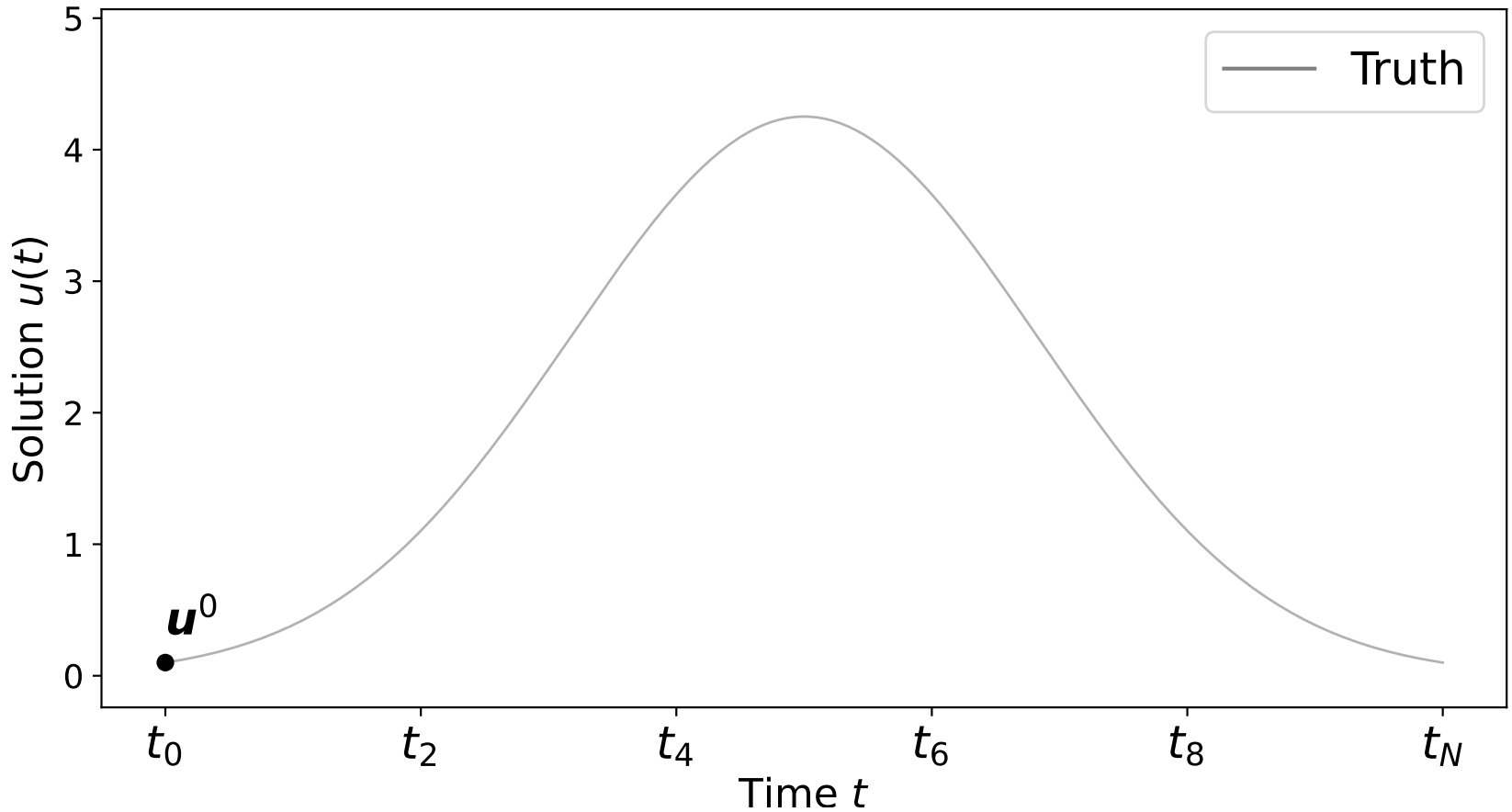
- Empirical results
- Computational cost

Random Weights Neural Networks Parareal New!

- Random Weights Neural Networks Theory
- Empirical results
- Computational cost

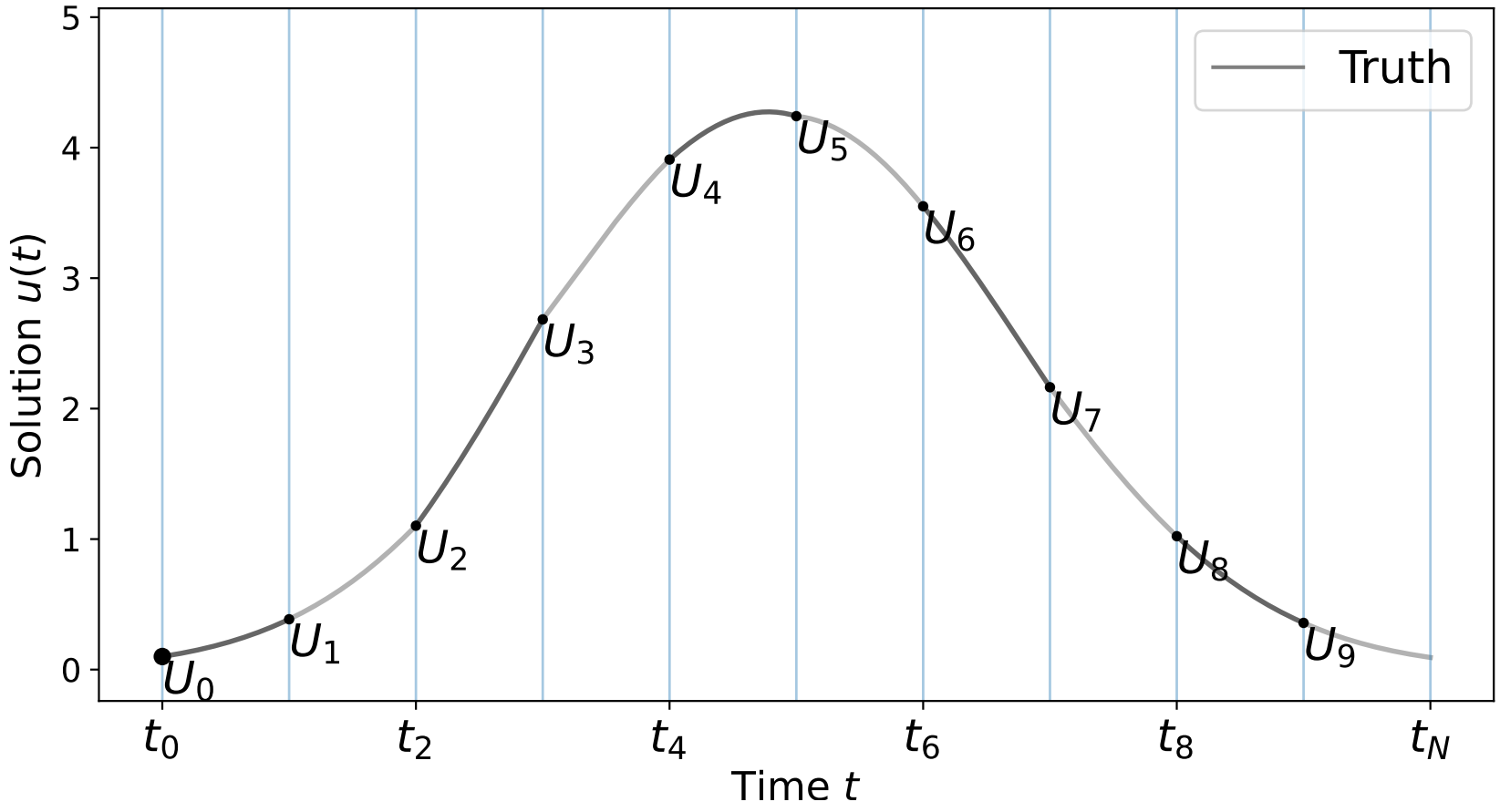
Parareal (Lions et al. [LMT01])

Parareal - Sketch of behavior - 1D System



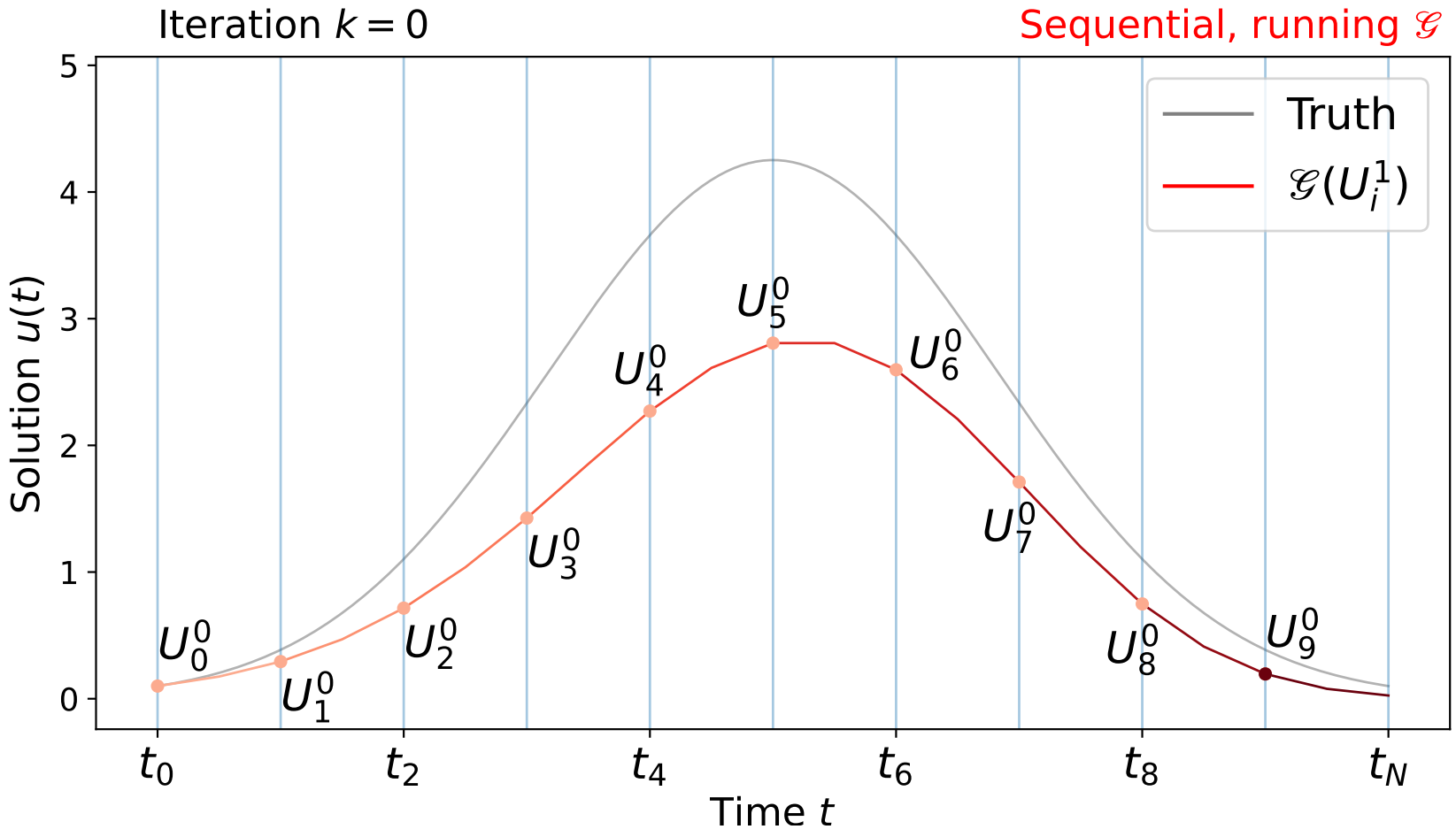
We wish to solve the d-dimensional ODE $\frac{d\boldsymbol{u}}{dt} = h(\boldsymbol{u}(t), t)$ on $t \in [t_0, t_N]$, with $\boldsymbol{u}(t_0) = \boldsymbol{u}^0$, $N \in \mathbb{N}$, and $\boldsymbol{u}^0 \in \mathbb{R}^d$; here $d = 1$.

Parareal - Sketch of behavior - 1D System



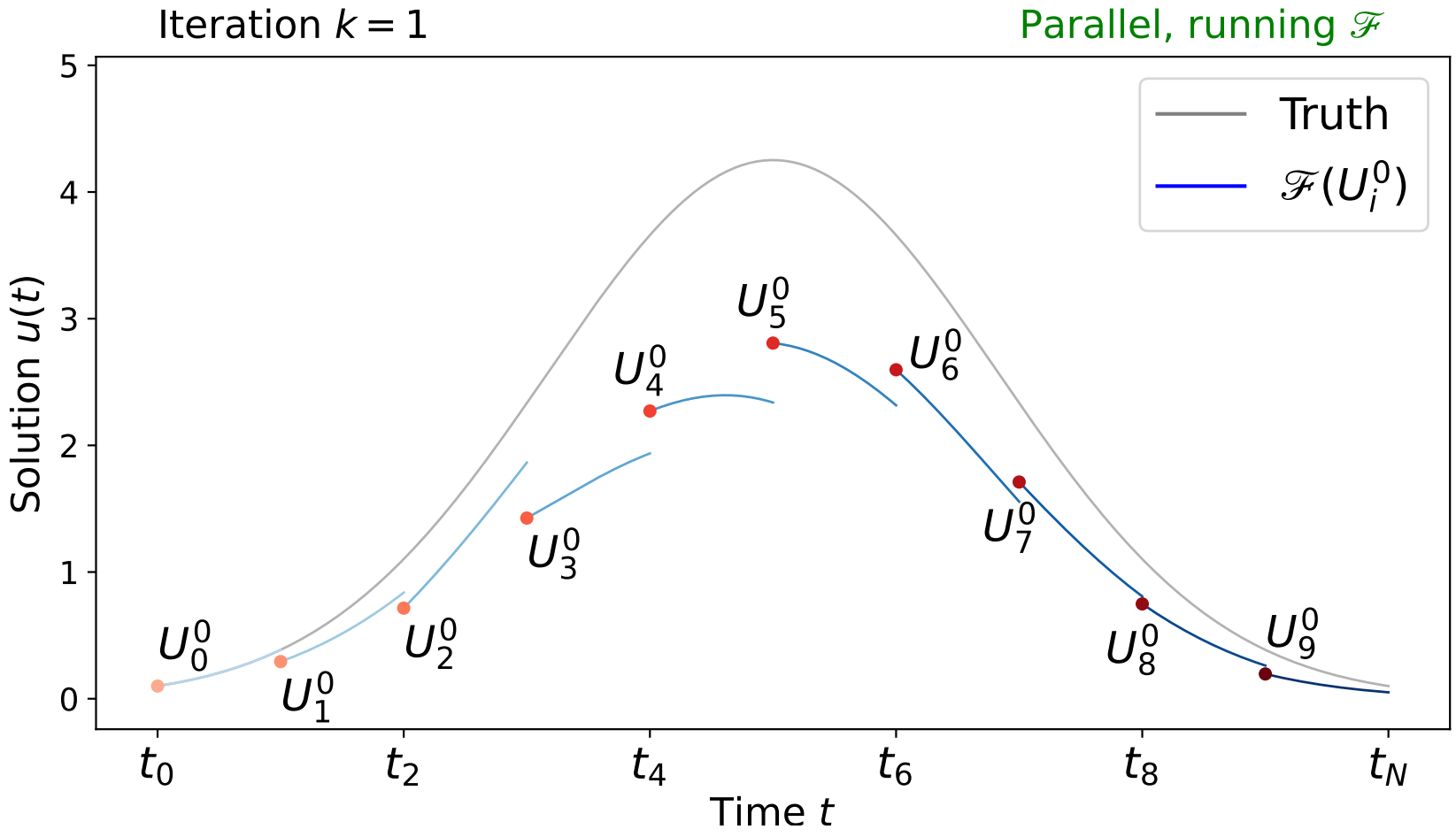
We divide the time interval in N sub-intervals and assign one processor for each. The initial conditions $\boldsymbol{U}_i$, $i = 1, ..., N-1$ are unknown and need to be estimated.

Parareal - Sketch of behavior - 1D System



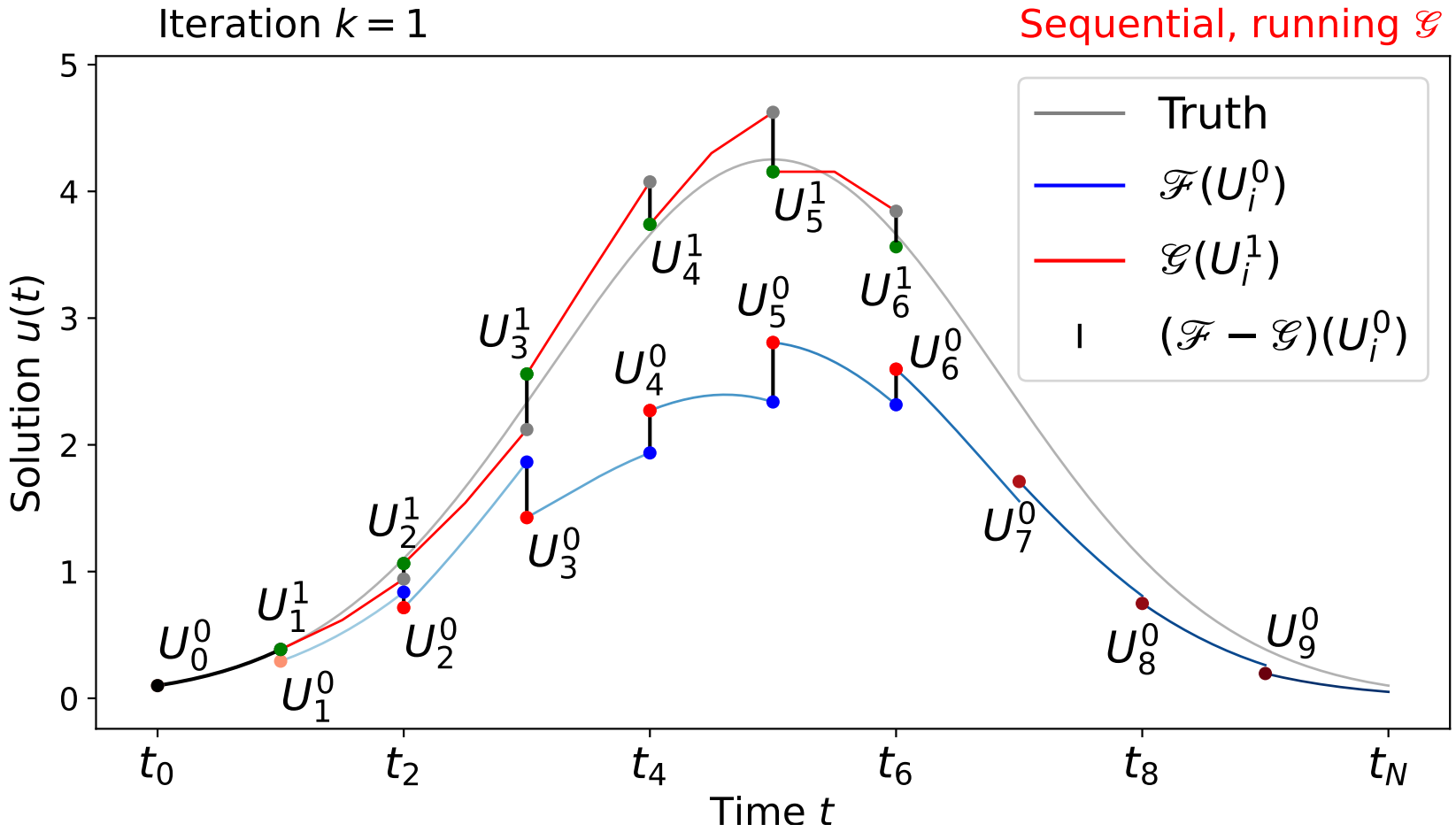
We run a cheap (fast), inaccurate coarse solver $\mathscr{G}$ to provide approximate initial conditions $\boldsymbol{U}_i^k$ for iteration $k = 0$ (initialization) and all intervals $i = 1, \ldots, N-1$.

Parareal - Sketch of behavior - 1D System



Once *some* initial conditions are available, we can run a precise (slow) fine solver $\mathscr{F}$ in *parallel* over the N processors, each started from an initial condition U_i^0.

Parareal - Sketch of behavior - 1D System



The initial conditions are then *sequentially* updated to satisfy a continuity condition, using the Parareal predictor-corrector rule

$$\boldsymbol{U}_i^k = \mathscr{G}(\boldsymbol{U}_{i-1}^k) + \mathscr{F}(\boldsymbol{U}_{i-1}^{k-1}) - \mathscr{G}(\boldsymbol{U}_{i-1}^{k-1})$$

A generic Parareal algorithm

The Parareal predictor-corrector rule can be generalized to a) use data from the current iteration $k+1$ and b) account for different ways of computing the discrepancy $\mathscr{F} - \mathscr{G}$

$$\boldsymbol{U}_i^k = \mathscr{G}(\boldsymbol{U}_{i-1}^k) + \widehat{f}(\boldsymbol{U}_{i-1}^k), \tag{1}$$

where $\widehat{f} : \mathbb{R}^d \to \mathbb{R}^d$ models $\mathscr{F} - \mathscr{G}$. Parareal uses

$$\widehat{f}_{\text{Para}}(\boldsymbol{U}_{i-1}^k) = (\mathscr{F} - \mathscr{G})(\boldsymbol{U}_{i-1}^{k-1}).$$

To evaluate Parareal's performance we use the *parallel speed-up* $S_{\text{alg}} := T_{\text{Serial}}/T_{\text{alg}}$, where

- T_{Serial} is the cost of running $\mathscr{F}$ *sequentially* over $[t_0, t_N]$
- T_{alg} is the runtime ofthe parallel procedure (Parareal)

A generic speed-up calculation

Assume that running $\mathscr{F}$ over one interval $[t_i, t_{i+1}]$ takes $T_{\mathscr{F}}$ time, and similarly for $\mathscr{G}$, and let K_{alg} be the iterations to convergence,

$$S_{\text{alg}} \approx \left(\frac{K_{\text{alg}}}{N} + \text{sequential cost} \left(\frac{T_{\mathscr{G}}}{T_{\mathscr{F}}} \right) + \frac{T_{\text{mdl}}}{N T_{\mathscr{F}}} \right)^{-1} \leq \left(\frac{K_{\text{alg}}}{N} \right)^{-1},$$

where T_{mdl} is the overall cost of evaluating $\widehat{f}$.

Parareal has $T_{\text{Para}} \in O(1)$ and is efficient when $K_{Para} < N$ and $T_{\mathscr{G}} / T_{\mathscr{F}} << 1$.

How can we improve on this?

Keep $\mathscr{G}$ fixed and reduce K_{Para}.

GPararealal (Pentland et al. [Pen+23])

GPararealreal (Pentland et al. [Pen+23])

GPararealreal stores all the discrepancies $(\mathscr{F} - \mathscr{G})$ into a dataset $\mathcal{D}_k$ of cardinality $|\mathcal{D}_k| = NK$ by iteration k

$$\mathcal{D}_k := \{(\boldsymbol{U}_{i-1}^j, (\mathscr{F} - \mathscr{G})(\boldsymbol{U}_{i-1}^j)),\ \ i = 1, \dots, N,\ \ j = 0, \dots, k-1\},$$

and models each coordinate $s = 1, \dots, d$ of $\widehat{f}(\boldsymbol{U}_{i-1}^k)$ through a Gaussian process (GP), using the posterior mean μ

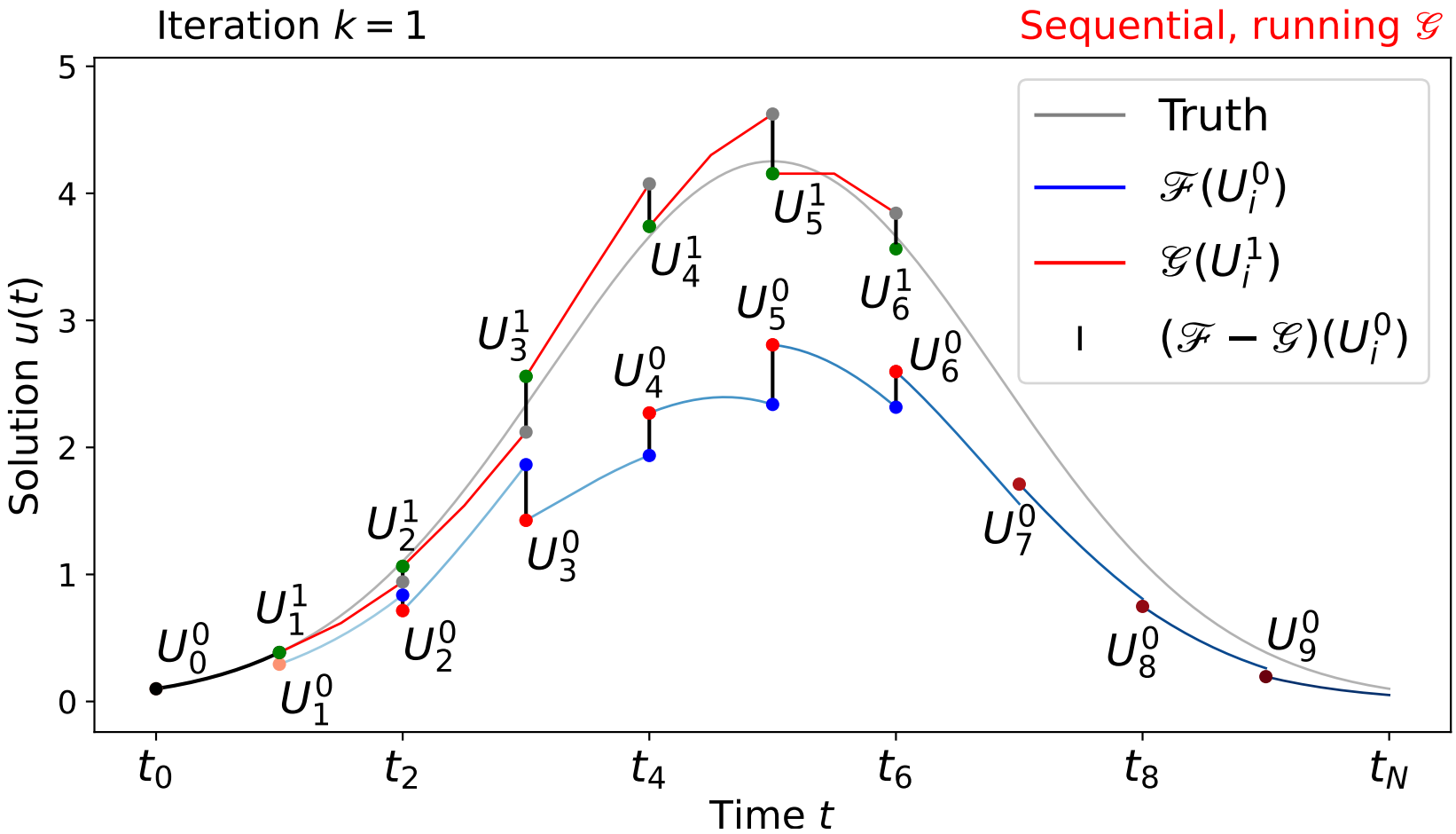
$$\widehat{f}_{\mathrm{GP}}(\boldsymbol{U}_{i-1}^k)_s = \mu_{\mathcal{D}_k}^{(s)}(\boldsymbol{U}_{i-1}^k) \in \mathbb{R},$$

thereby training d different models, one per ODE coordinate.

Since evaluating μ involves inverting a $|\mathcal{D}_k| \times |\mathcal{D}_k|$ matrix, $T_{\mathrm{GP}}(k)$ scales as $O((d/N \vee 1)k^3N^3)$, with $\vee$ the minimum operator, giving

$$T_{\mathrm{GP}} = \sum_{k=1}^{K_{\mathrm{GPara}}} T_{\mathrm{GP}}(k) \in O\left((d/N \vee 1)K_{\mathrm{GPara}}^4 N^3\right) > T_{\mathrm{Para}} \in O(1)$$

Recall



$$S_{\text{alg}} \approx \left(\frac{K_{\text{alg}}}{N} + \text{sequential cost} \left(\frac{T_{\mathscr{G}}}{T_{\mathscr{F}}} \right) + \frac{T_{\text{mdl}}}{N T_{\mathscr{F}}} \right)^{-1} \leq \left(\frac{K_{\text{alg}}}{N} \right)^{-1}$$

GParareal (Pentland et al. [Pen+23])

System	Parareal	GParareal
FitzHugh–Nagumo (FHN)	11	**5**
Rossler	18	**13**
Hopf	19	**10**
Brusselator	**19**	20
Lorenz	15	**11**
Double Pendulum	15	**10**

Comparison of performance for common ODE systems in the literature, described in Slides 50-56.

GPararealal - Performance - Hopf bifurcations

To showcase the empirical performance of GPararealal, consider a non-linear model for the study of Hopf bifurcations ([Sey09, pg. 72]; also Slide 52), defined by the following equations

$$\frac{du_1}{dt} = -u_2 + u_1\left(\frac{t}{T} - u_1^2 - u_2^2\right), \quad \frac{du_2}{dt} = u_1 + u_2\left(\frac{t}{T} - u_1^2 - u_2^2\right), \quad (2)$$

where we note the dependence on time. In practice, we add time as an additional coordinate yielding a $d = 3$ autonomous system.

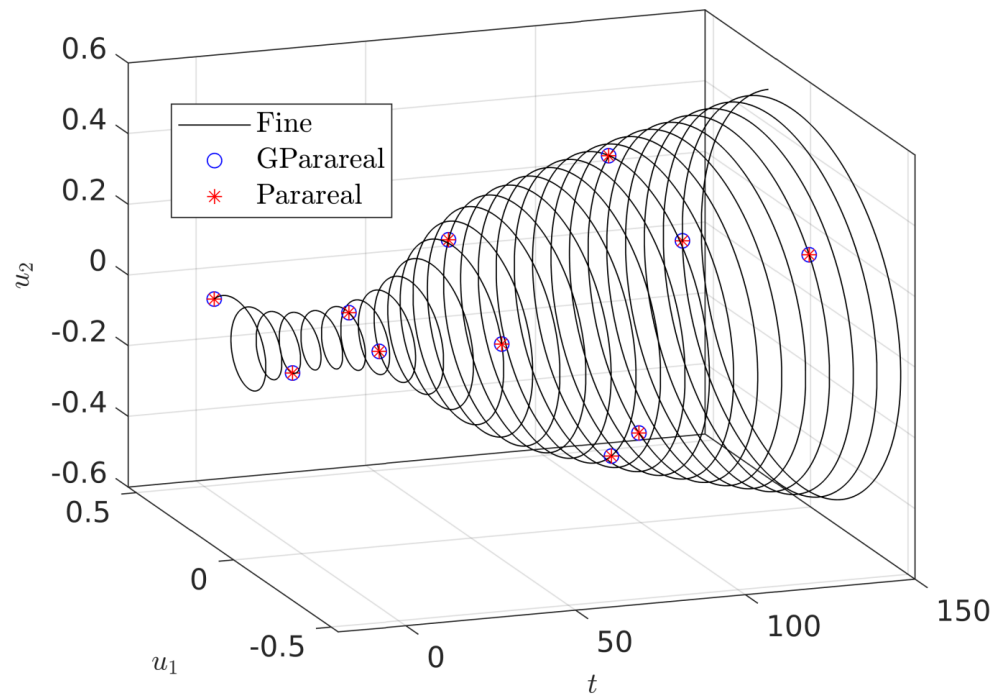
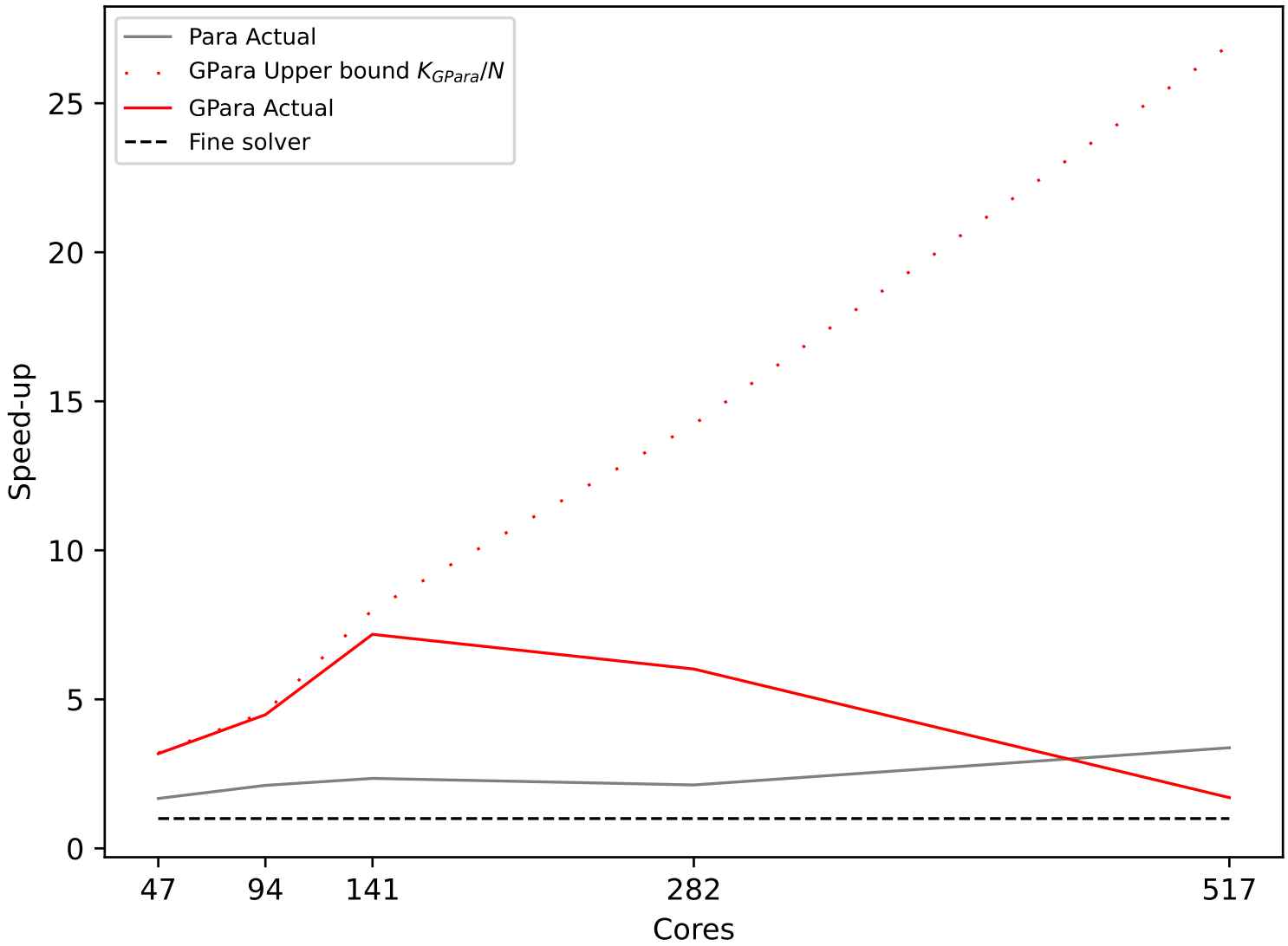


Image taken from Pentland et al. [Pen+23]

GPararealal - Performance - Hopf bifurcations



GPararealreal - Improvements

How can we improve? Maintain $K \leq K_{GPara*}$ while reducing T_{GP*}.

The GP cost comes from the sample size $O(Nk)$ by iteration k.
Can we reduce the sample size without affecting performance?

Yes, we can fit the model using a small subset consisting of the *nearest neighbors* to the prediction point. This is sufficient to smooth locally because very few poin ts are empirically close in Euclidean distance.

Nearest Neighbor GPararealal (NN-GPararealal)

NN-GPareareal

Whereas GPareareal trains the GP once per iteration k on $\mathcal{D}_k$, NN-GPareareal is re-trained every time a prediction $\widehat{f}(\boldsymbol{U}_{i-1}^k)$ is made, on a subset $\mathcal{D}_{i-1,k} \subset \mathcal{D}_k$ with cardinality $|\mathcal{D}_{i-1,k}| = m$

$$\mathcal{D}_{i-1,k} := \{m \text{ nearest neighbors (NN) of } \boldsymbol{U}_{i-1}^k\}.$$

This is known as a nearest neighbor Gaussian process (NNGP).

Using the reduced dataset, NN-GPareareal models $\widehat{f}$ as

$$\widehat{f}_{\mathrm{GP}}(\boldsymbol{U}_{i-1}^k)_s = \mu_{\mathcal{D}_{i-1,k}}^{(s)}(\boldsymbol{U}_{i-1}^k),$$

at a cost of (assuming $K_{\text{NN-GPara}}$ iterations to convergence)

$$T_{\mathrm{NNGP}} \in O\left((d/N \vee 1) K_{\text{NN-GPara}} N(m^3 + \log(K_{\text{NN-GPara}} N)\right),$$

loglinear in N, instead of cubic as GPareareal.

NN-GPararealal - visualizing $\mathcal{D}_{i-1,k}$

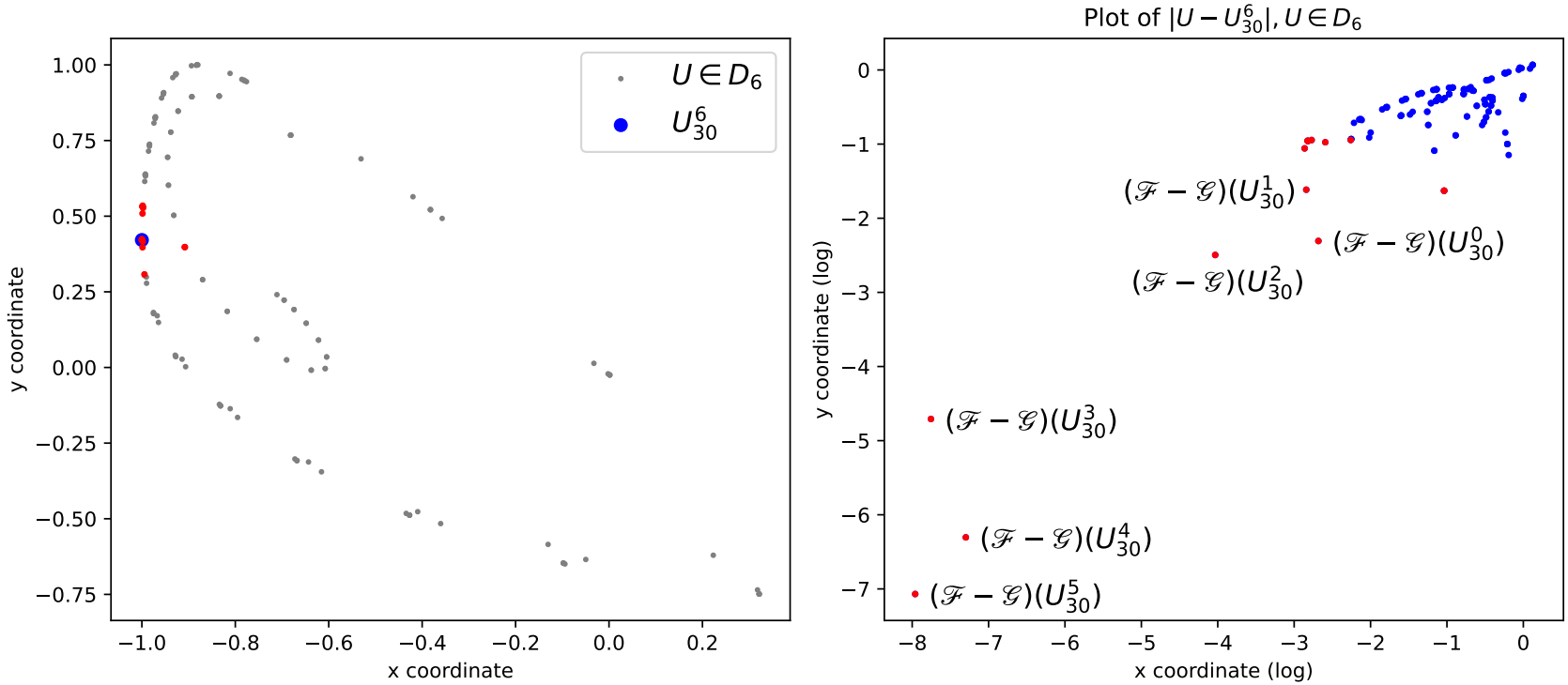


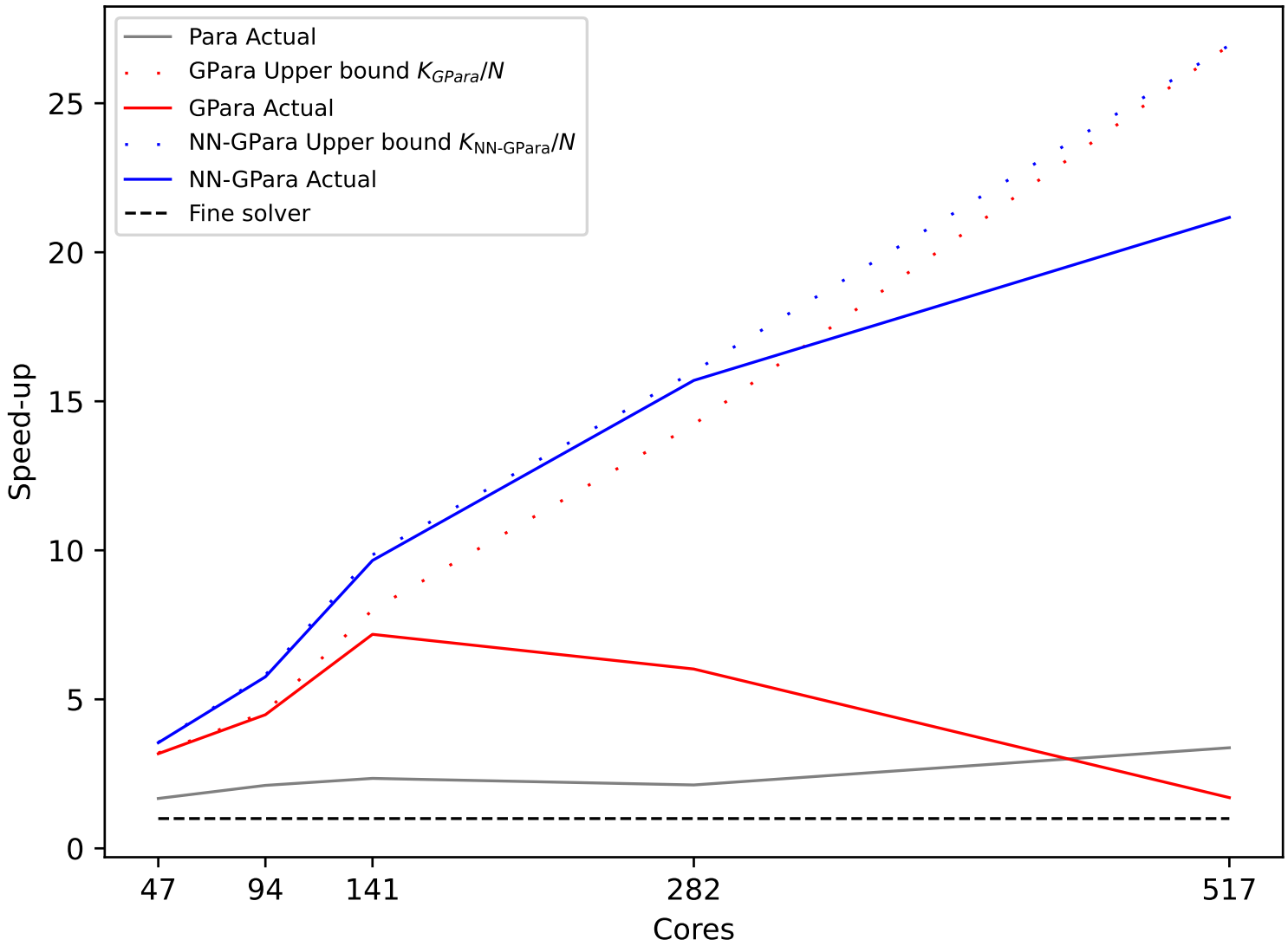
Figure: Visualization of the dataset for the two-dimensional Brusselator system (53) Left, a scatterplot of the observation **U** accumulated by iteration 6 (gray), with the test observation U_{30}^6 (blue). In red are the $m = 15$ nearest neighbors to the test observations. Note how far most points are from U_{30}^6. The right plot makes this easier to see by displaying the absolute (log) distance coordinate-wise between $U \in \mathbf{U}$ and U_{30}^6.

NN-GParareal - Performance

System	Parareal	GParareal	NN-GParareal
FitzHugh–Nagumo	11	**5**	**5**
Rossler	18	13	**12**
Hopf	19	10	**9**
Brusselator	19	20	**17**
Lorenz	15	11	**9**
Double Pendulum	15	**10**	**10**

Comparison of performance for common ODE systems in the literature, described in Slides 50-56.

NN-GParareal - Performance - Hopf bifurcations



NN-GParareal - Performance - FitzHugh-Nagumo PDE

We explore the performance of Parareal and its variants on a high-dimensional system. We use the two-dimensional, non-linear FitzHugh-Nagumo PDE model [AF09]. See also Slide 60.

It represents a set of cells constituted by a small nucleus of pacemakers near the origin immersed among an assembly of excitable cells. The simpler FHN ODE system only considers one cell and its corresponding spike generation behavior.

We discretize both spatial dimensions using finite difference and $\tilde{d}$ equally spaced points, yielding an ODE with $d = 2\tilde{d}^2$ dimensions.

We consider $\tilde{d} = 10, 12, 14, 16$, corresponding to $d = 200, 288, 392, 512$, and set $N = 512$.

NN-GParareal - Performance - FitzHugh-Nagumo PDE

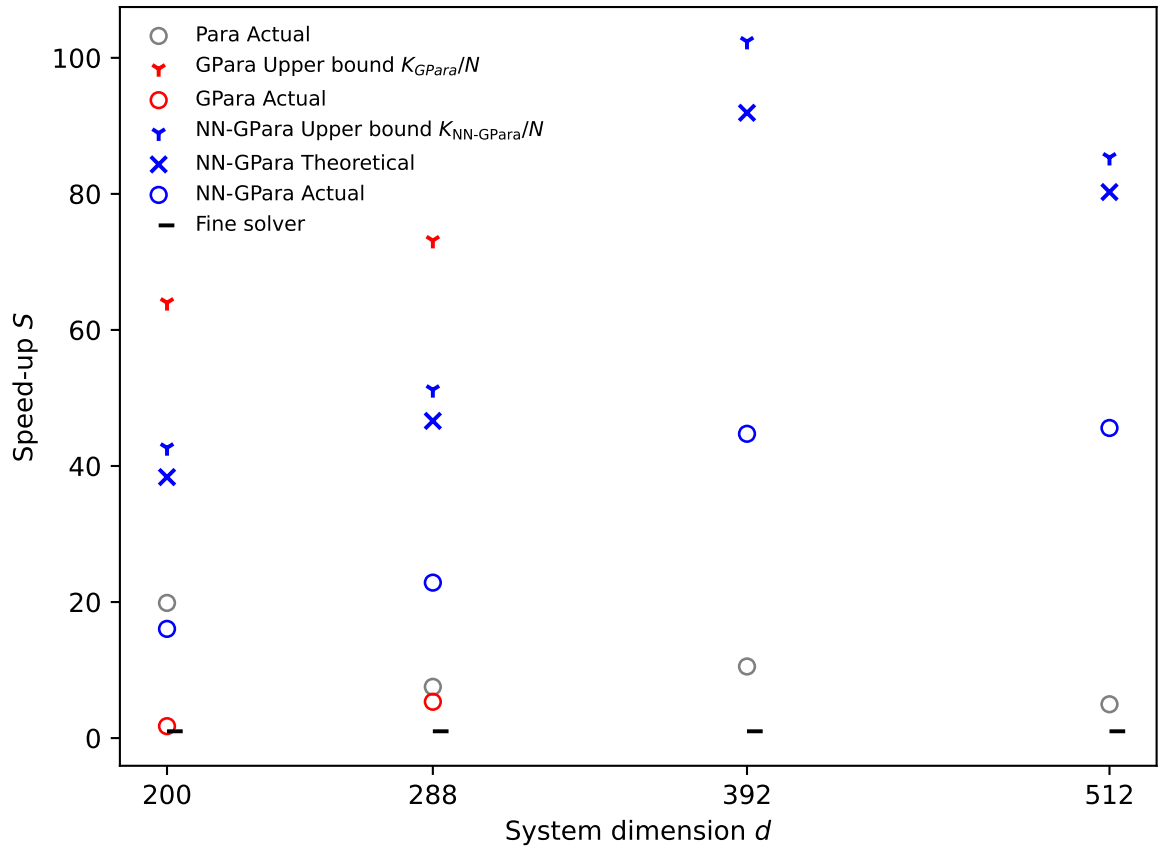


Figure: Plot of speed-ups for Parareal and its variants for the FitzHugh-Nagumo PDE model. The speed-ups are computed according to the formulas above. For $N = 256, 512$, GParareal failed to converge within the computational time budget of 48 hours.

Recap

GParareal:

- Pro: Accelerated convergence compared to Parareal.
- Con: Infeasible for moderate numbers of processors N and ODE dimension d, limiting applicability beyond toy examples.
- Con: It requires one model per ODE dimension.
- Con: Hyperparameter optimization via log-likelihood maximization is very expensive. Usually non-convex.

Recap

Nearest-Neighbors GParareal:

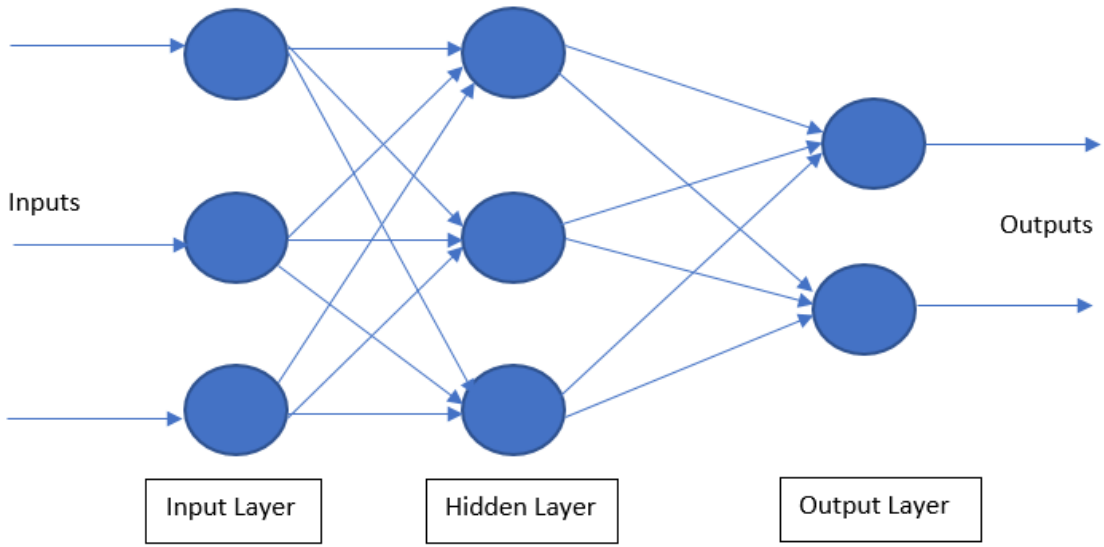
- Pro: Achieves drastic data reduction maintaining or improving performance.
- Pro: The model is re-trained for every prediction, partially relaxing the stationarity assumption.
- Pro: Reduced computational complexity from cubic to loglinear. Verified empirical scalability in N and d (limited).
- Pro: The algorithm runtime can be estimated beforehand.
- Con: It requires one NNGP per ODE dimension.
- Con: Hyperparameter optimization via log-likelihood maximization is still expensive. Usually non-convex.

Random Weights Neural Networks Parareal (RWParareal)

RWParareal

To address the remaining challenges, we deviate from the GP framework and consider a new approach using random weights neural networks (RWNN). RWNN is a learning paradigm for efficiently training single hidden layer feed-forward neural networks, where both input and hidden layer weights are randomly sampled and kept fixed throughout the training procedure.

They can be obtained as an approximation of kernel methods, known as Random Fourier features [RR07] or as a way to avoid back-propagation in neural networks, known as Extreme Learning Machines [HZS04] or, random weights neural networks [Cao+18].



RWParareal

The advantages of RWNNs are their strong empirical performance, theoretical guarantees on their universal approximating abilities [GGO23], and the availability of a closed-form solution for the output layer weights. Keeping the hidden layer weight fixed avoids backpropagation, significantly reducing the runtime.

Let M denote the number of hidden neurons and $H_W^{A,\zeta}(\boldsymbol{U})$ be a single-hidden-layer feed-forward neural networks

$$H_W^{A,\zeta}(\boldsymbol{U}) = W^\top \boldsymbol{\sigma}(A\boldsymbol{U} + \boldsymbol{\zeta}) \in \mathbb{R}^d, \quad \boldsymbol{U} \in \mathbb{R}^d$$

where

- $\boldsymbol{\zeta} \in \mathbb{R}^M$ are random input weights, $\boldsymbol{\zeta} \sim \mathcal{P}_{\boldsymbol{\zeta}}$
- $A \in \mathbb{R}^{M \times d}$ are random hidden weights, $A \sim \mathcal{P}_A$
- $W \in \mathbb{R}^{M \times d}$ are trainable output weights
- $\boldsymbol{\sigma} : \mathbb{R}^M \to \mathbb{R}^M$ is the activation function $\boldsymbol{\sigma}(\boldsymbol{x}) = \max(\boldsymbol{x}, 0))$.

RWPararealal

After observing a dataset $\mathcal{D}_k$, the output weights are estimated by minimizing the λ-penalized squared error loss between the network predictions and the outputs,

$$\widehat{W}_{\lambda}^{\mathcal{D}_k} = \arg \min_{W \in \mathbb{R}^{M \times d}} \left\{ \sum_{(\boldsymbol{U}, \boldsymbol{Y}) \in \mathcal{D}_k} \left\| H_W^{A,\zeta}(\boldsymbol{U}) - \boldsymbol{Y} \right\|^2 + \lambda \sum_{s=1}^{d} \left\| W_{(\cdot,s)} \right\|^2 \right\}.$$

$\widehat{W}_{\lambda}^{\mathcal{D}_k}$ can be computed explicitly in closed form as

$$\widehat{W}_{\lambda}^{\mathcal{D}_k} = \left(X^\top X + \lambda \mathbb{I}_M \right)^{-1} X^\top Y,$$

where $X \in \mathbb{R}^{Nk \times M}$, $X_{(j,\cdot)} := \boldsymbol{\sigma}(AU_{(j,\cdot)} + \boldsymbol{\zeta})$, $j = 1, \ldots, Nk$. Then,

$$\widehat{f}_{\mathrm{R}}(\boldsymbol{U}_{i-1}^k) = H^{A,\zeta}_{\widehat{W}_{\lambda}^{\mathcal{D}_{i-1,k}}}(\boldsymbol{U}_{i-1}^k).$$

Note that the weights $\widehat{W}_{\lambda}^{\mathcal{D}_{i-1,k}}$ are obtained using the reduced dataset $\mathcal{D}_{i-1,k}$ consisting of the nearest neighbors of $\boldsymbol{U}_{i-1}^k$

RWParareal

Several hyperparameters should be tuned to control the performance of RWNN, namely $M, \lambda, \mathcal{P}_A$ and $\mathcal{P}_\zeta$. We took $\mathcal{P}_A = \mathcal{P}_\zeta \sim \text{Uniform}(-1, 1)$, following known approximation bounds for this case [GGO23, Proposition 3].

Since $\mathscr{F} - \mathscr{G}$ is deterministic, we set $\lambda = 0$.

Comparing the model cost *of one prediction* with NN-GParareal

$$T_{\text{NNGP}}(\boldsymbol{U}_{i-1}^k) \approx C_{\text{GP}}(d/N \vee 1)(\underbrace{m^3}_{\text{inversion}} + \underbrace{m^2 d}_{\text{kernel eval.}})$$

$$T_{\text{RW}}(\boldsymbol{U}_{i-1}^k) \approx C_{\text{RW}}(\underbrace{M^3}_{\text{inversion}} + \underbrace{mMd}_{\text{compute } X})$$

However, the constant is much smaller, $C_{\text{RW}} << C_{\text{GP}}$ due to the lack of hyperparameters optimization. Moreover, $d >> N$ is often the case for PDEs.

We now consider several examples.

A comment on the choice of PDEs

We evaluate RWParareal on three increasingly complex PDE systems drawn from an extensive benchmark suite of time-dependent PDEs [Tak+22]:

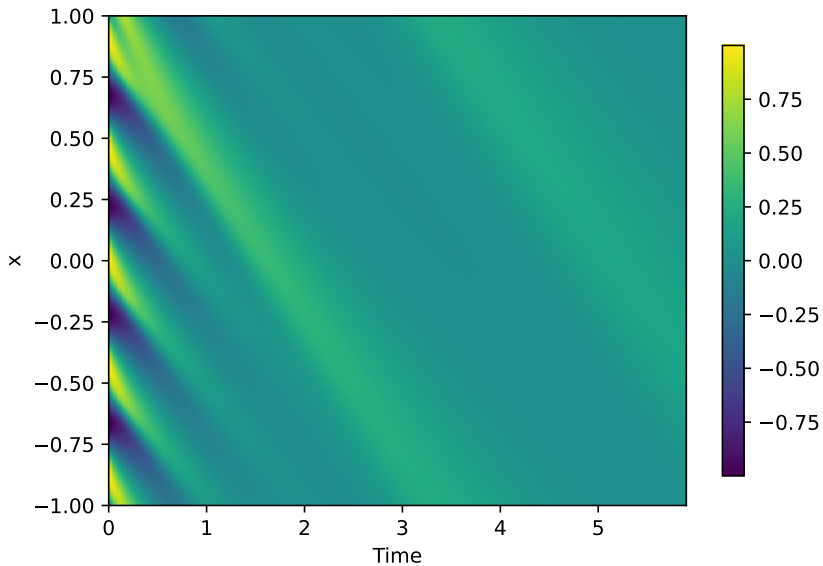
- The 1D Viscous Burgers' equation a non-linear, one-dimensional system exhibiting hyperbolic behavior.
- The 2D Diffusion-Reaction equation, a challenging benchmark used to model biological pattern formation [Tur52].
- The shallow water equations (SWEs), a system of hyperbolic PDEs derived from the compressible Navier-Stokes equations, exhibiting behaviors of real-world significance known to challenge numerical integrators

We intentionally chose two hyperbolic equations (Burgers' and SWE) to address known Parareal issues in solving hyperbolic systems with slow or non-convergent behavior [Bal05; SR05; GV07; AKT16; DM13].

RWPararealal - Viscous Burgers' Equation

Viscous Burgers' Equation is a non-linear, one-dimensional system exhibiting hyperbolic behavior [Sch+18], described by the equation:

$$v_t = \nu v_{xx} - v v_x \quad (x,t) \in (-L, L) \times (t_0, t_N], \tag{3}$$



Numerical solution of viscous Burgers' equation over $(x,t) \in [-1,1] \times [0,5.9]$ with $d = 1128$.

Viscous Burgers' $d = 128$, $N = 128$

Algorithm	K	$NT_{\mathscr{G}}$	$T_{\mathscr{F}}$	Model	Total	Speed-up
Fine	-	-	-	-	13h 10m	1
Parareal	90	0s	6m	0s	8h 54m	1.48
NN-GParareal	14	0s	6m	12m	1h 39m	7.95
RWParareal	10	0s	6m	1s	1h 4m	**12.44**

Viscous Burgers' $d = 1128$, $N = 128$

Algorithm	K	$NT_{\mathscr{G}}$	$T_{\mathscr{F}}$	Model	Total	Speed-up
Fine	-	-	-	-	18h 14m	1
Parareal	91	0s	9m	0s	12h 57m	1.41
NN-GParareal	6	2s	9m	1h 25m	2h 17m	7.98
RWParareal	4	2s	9m	1s	34m	**31.97**

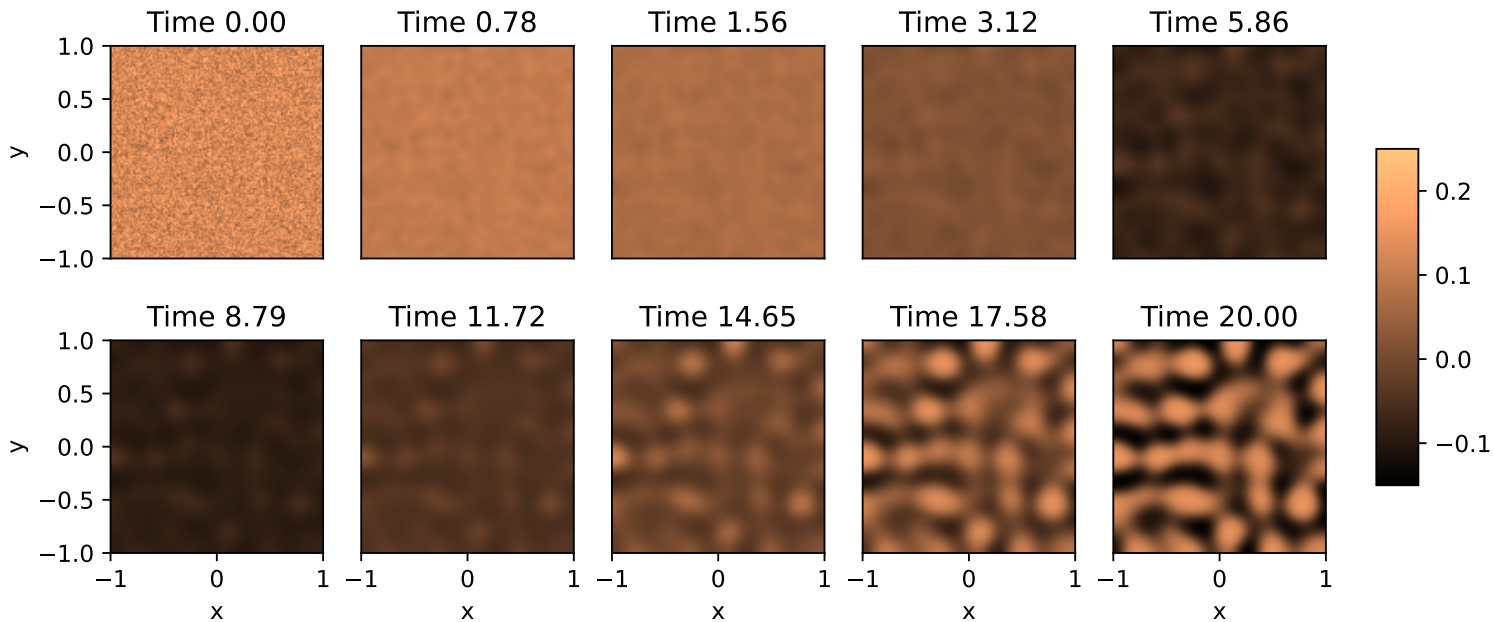
Speed-up scalability of Parareal, NN-GParareal (with $m = 18$), and RWParareal (with $m = 3$ and $M = 20$). $T_{\mathscr{F}}$ and $T_{\mathscr{G}}$ refer to the runtimes per interval of the fine and coarse solvers, respectively. K denotes the number of iterations taken to converge. 'Model' covers all model-related costs, including training and predicting. 'Total' is the overall runtime of the algorithm, while the speed-up is computed with respect to the cost of the sequential fine solver.

RWPararealal - 2D Diffusion-Reaction Equation

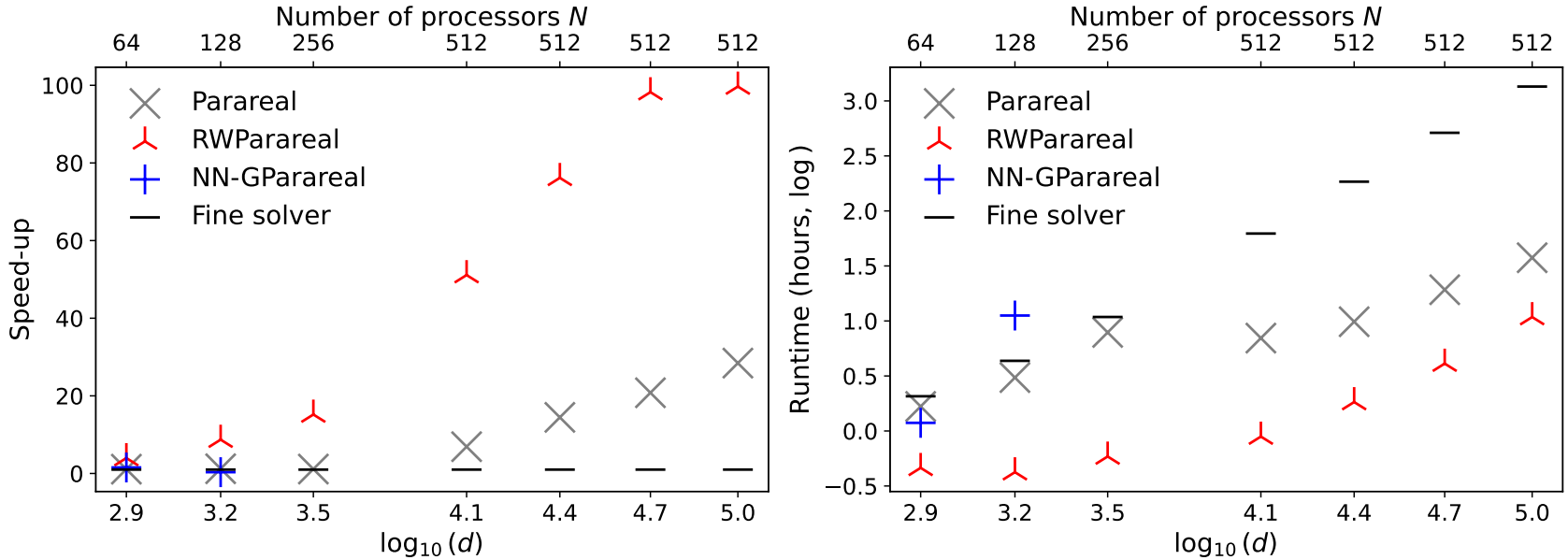
The Diffusion-Reaction equation is a system of two non-linearly coupled variables, the activator $u = u(t, x, y)$ and the inhibitor $v = v(t, x, y)$, defined on a two-dimensional domain

$$\partial_t u = D_u \partial_{xx} u + D_u \partial_{yy} u + R_u, \quad \partial_t v = D_v \partial_{xx} v + D_v \partial_{yy} v + R_v,$$

where D_u and D_v are the diffusion coefficient for the activator and inhibitor, respectively, and $R_u = R_u(u, v)$ and $R_v = R_v(u, v)$ are the activator and inhibitor reaction function, respectively.



Numerical solution of the Diffusion-Reaction equation over $(x, y) \in [-1, 1]^2$ with $N_x = N_y = 235$ for a range of system times t. Only the activator $u(t, x, y)$ is plotted.



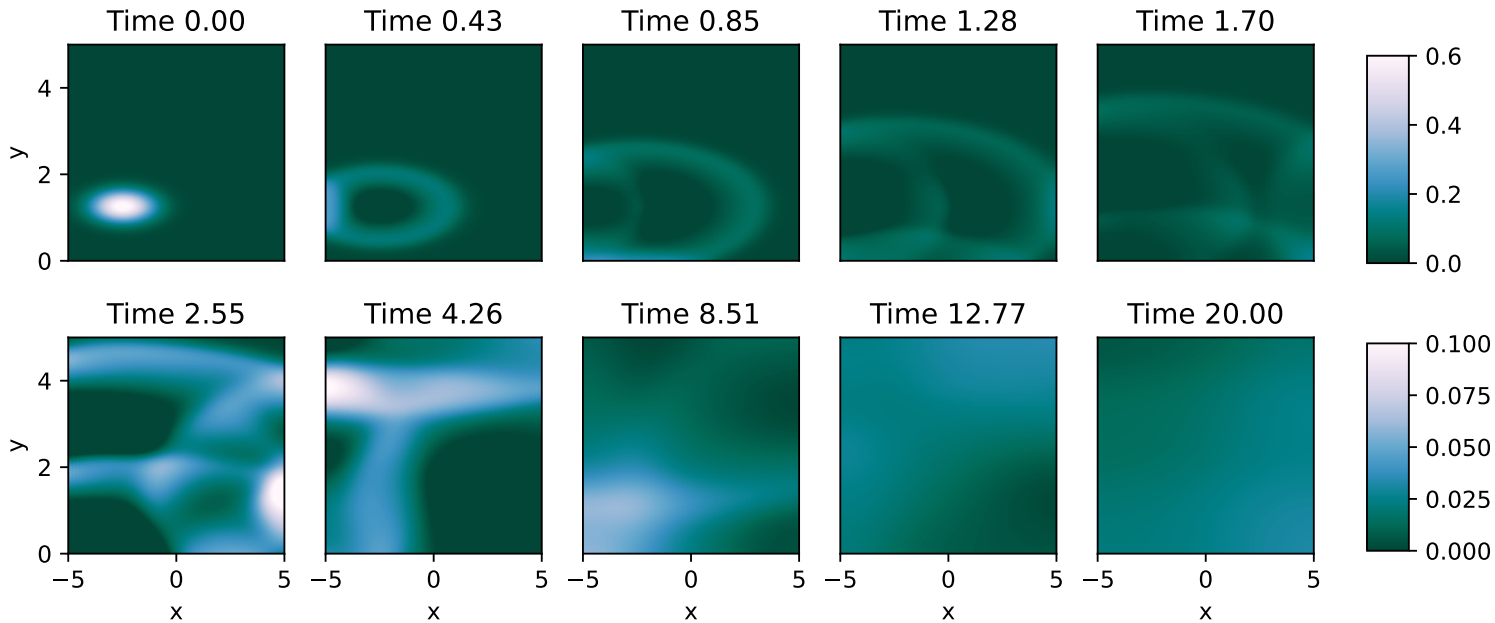
Plot of speed-ups (left) and runtime (right) for Parareal, NN-GParareal (with $m_{\mathrm{NN}} = 20$), and RWParareal (with $m_{\mathrm{R}} = 3$ and $M = 20$) for the 2D Diffusion-Reaction equation as a function of the system dimension d (bottom x-axis) and the number of cores N (top x-axis). N is capped at 512 cores to simulate limited resources.

RWPararealal - Shallow Water Equations (SWEs)

The SWEs are used to model free-surface flow problems. On a two-dimensional domain, they are described by a system of hyperbolic PDEs

$$\partial_t h + \nabla h\mathbf{u} = 0, \quad \partial_t h\mathbf{u} + \nabla \left(u^2 h + \frac{1}{2} g_r h^2 \right) = -g_r h \nabla b,$$

where $\mathbf{u} = u, v$ represents the velocities in the horizontal and vertical direction, respectively. h denotes the water depth, b describes a spatially varying bathymetry, and $h\mathbf{u}$ is interpreted as the directional momentum components.



Numerical solution of the SWEs over $(x, y) \in [-5, 5] \times [0, 5]$ with $N_x = 264$ and $N_y = 133$ for a range of system times t. Only the water depth h (blue) is plotted.

Table: Speed-up analysis for the Shallow Water Equation

d	K_P	K_R	T_F	T_P	T_R	S_P	S_R
15453	52	14	22h 54m	5h 8m	1h 25m	4.47	**16.24**
31104	50	13	3d 2h	15h 43m	4h 9m	4.68	**17.77**
60903	14	9	13d 15h	19h 30m	12h 34m	16.74	**25.97**
105336	8	6	38d 4h	1d 7h	23h 30m	29.35	**38.98**

Speed-up study of the SWEs for the fine solver, Parareal, NN-GParareal (with $m_{\mathrm{NN}} = 20$) and RWParareal (with $m_{\mathrm{R}} = 4$ and $M = 20$), denoted by the subscripts P, NN, F, and R , respectively. d denotes the dimension of the corresponding ODE system.

We have

- K_P, K_R: iterations to convergence for Parareal and RWParareal
- T_F, T_P, T_R: wallclock runtime for $\mathscr{F}$, Parareal, and RWParareal
- S_P, S_R: observed speed-up of Parareal and RWParareal

Wrapping up

What have we learned?

GParareal:

- Pro: Accelerated convergence compared to Parareal.
- Con: Infeasible for moderate numbers of processors N and ODE dimension d, limiting applicability beyond toy examples.
- Con: It requires one model per ODE dimension.
- Con: Hyperparameter optimization via log-likelihood maximization is very expensive. Usually non-convex.

Wrapping up

Nearest-Neighbors GParareal:

- Pro: Achieves drastic data reduction maintaining or improving performance.
- Pro: The model is re-trained for every prediction, partially relaxing the stationarity assumption.
- Pro: Reduced computational complexity from cubic to loglinear. Verified empirical scalability in N and d (limited).
- Pro: The algorithm runtime can be estimated beforehand.
- Con: It requires one NNGP per ODE dimension.
- Con: Hyperparameter optimization via log-likelihood maximization is still expensive. Usually non-convex.

Wrapping up

Random weights neural networks Parareal:

- All NN-GParareal advantages, plus
- Pro: One model learns all d coordinates
- Pro: No hyperparameter optimization needed, training about x1000 faster than GPs
- Pro: Convex optimization problem with closed-form solution
- Pro: Universal approximation guarantees
- Pro: Drastically improved scalability in d, up to 10^5 spatial discretization points
- Con: Overall performance still affected by $\mathscr{G}$.

Further research question:

- Include uncertainty estimation for the algorithm's solution (probabilistic numerics).

References I

[AF09] Benjamin Ambrosio and Jean-Pierre Françoise. "Propagation of bursting oscillations". In: *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 367.1908 (2009), pp. 4863–4875.

[AKT16] Gil Ariel, Seong Jun Kim, and Richard Tsai. "Parareal multiscale methods for highly oscillatory dynamical systems". In: *SIAM Journal on Scientific Computing* 38.6 (2016), A3540–A3564.

[Bal05] Guillaume Bal. "On the convergence and the stability of the parareal algorithm to solve partial differential equations". In: *Domain decomposition methods in science and engineering*. Springer, 2005, pp. 425–432.

[Cao+18] Weipeng Cao et al. "A review on neural networks with random weights". In: *Neurocomputing* 275 (2018), pp. 278–287.

References II

[DM13] Xiaoying Dai and Yvon Maday. "Stable parareal in time method for first-and second-order hyperbolic systems". In: *SIAM Journal on Scientific Computing* 35.1 (2013), A52–A78.

[Dan97] J.M. Anthony Danby. *Computer modeling: from sports to spaceflight... from order to chaos.* 1997.

[DG89] Jean-Louis Deneubourg and Simon Goss. "Collective patterns and decision-making". In: *Ethology Ecology & Evolution* 1.4 (1989), pp. 295–311.

[For88] Bengt Fornberg. "Generation of finite difference formulas on arbitrarily spaced grids". In: *Mathematics of computation* 51.184 (1988), pp. 699–706.

References III

[GV07] Martin J. Gander and Stefan Vandewalle. "Analysis of the parareal time-parallel time-integration method". In: *SIAM Journal on Scientific Computing* 29.2 (2007), pp. 556–578.

[Gil21] William Gilpin. "Chaos as an interpretable benchmark for forecasting and data-driven modelling". In: *arXiv preprint arXiv:2110.05266* (2021).

[GGO23] Lukas Gonon, Lyudmila Grigoryeva, and Juan-Pablo Ortega. "Approximation bounds for random neural networks and reservoir systems". In: *The Annals of Applied Probability* 33.1 (2023), pp. 28–69.

References IV

[HZS04] Guang-Bin Huang, Qin-Yu Zhu, and Chee-Kheong Siew. "Extreme learning machine: a new learning scheme of feedforward neural networks". In: *2004 IEEE international joint conference on neural networks (IEEE Cat. No. 04CH37541)*. Vol. 2. Ieee. 2004, pp. 985–990.

[Kau93] Stuart A. Kauffman. *The origins of order: Self-organization and selection in evolution*. Oxford University Press, USA, 1993.

[LN71] René Lefever and Grégoire Nicolis. "Chemical instabilities and sustained oscillations". In: *Journal of theoretical Biology* 30.2 (1971), pp. 267–284.

[LMT01] Jacques-Louis Lions, Yvon Maday, and Gabriel Turinici. "Résolution d'EDP par un schéma en temps pararéel". In: *Comptes Rendus de l'Académie des Sciences-Series I-Mathematics* 332.7 (2001), pp. 661–668.

References V

[Lor63] Edward N. Lorenz. "Deterministic nonperiodic flow". In: *Journal of atmospheric sciences* 20.2 (1963), pp. 130–141.

[NAY62] Jinichi Nagumo, Suguru Arimoto, and Shuji Yoshizawa. "An active pulse transmission line simulating nerve axon". In: *Proceedings of the IRE* 50.10 (1962), pp. 2061–2070.

[Pen+23] Kamran Pentland et al. "GParareal: a time-parallel ODE solver using Gaussian process emulation". In: *Statistics and Computing* 33.1 (2023), p. 23.

[RR07] Ali Rahimi and Benjamin Recht. "Random features for large-scale kernel machines". In: *Advances in neural information processing systems* 20 (2007).

References VI

[Ras+90] Steen Rasmussen et al. "The coreworld: Emergence and evolution of cooperative structures in a computational chemistry". In: *Physica D: Nonlinear Phenomena* 42.1-3 (1990), pp. 111–134.

[Rös76] Otto E. Rössler. "An equation for continuous chaos". In: *Physics Letters A* 57.5 (1976), pp. 397–398.

[Sch+18] Andreas Schmitt et al. "A numerical study of a semi-Lagrangian Parareal method applied to the viscous Burgers equation". In: *Computing and Visualization in Science* 19.1 (2018), pp. 45–57.

[Sey09] Rüdiger Seydel. *Practical bifurcation and stability analysis*. Vol. 5. Springer Science & Business Media, 2009.

References VII

[SR05] Gunnar Andreas Staff and Einar M Rønquist. "Stability of the parareal algorithm". In: *Domain decomposition methods in science and engineering*. Springer, 2005, pp. 449–456.

[Tak+22] Makoto Takamoto et al. "Pdebench: An extensive benchmark for scientific machine learning". In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 1596–1611.

[Tho99] René Thomas. "Deterministic chaos seen in terms of feedback circuits: Analysis, synthesis," labyrinth chaos"". In: *International Journal of Bifurcation and Chaos* 9.10 (1999), pp. 1889–1905.

[Tur52] A. Turing. "The chemical basis of morphogenesis". In: *Philosophical Transactions of the Royal Society B* 237 (1952), pp. 37–72.

ODE/PDE Systems

Systems: FitzHugh–Nagumo

The FitzHugh-Nagumo (FHN) is a model for an animal nerve axon [NAY62]. It is a reasonably easy system to learn, does not exhibit chaotic behavior, and is commonly used throughout the literature. It is described by the following equations

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

with $(a, b, c) = 0.2, 0.2, 3$. We integrate over $t \in [0, 40]$ using $N = 40$ intervals, taking $u_0 = (-1, 1)$ as the initial condition. We use Runge-Kutta 2 with 160 steps for the coarse solver $\mathscr{G}$, and Ruge-Kutta 4 with $1.6e^5$ steps for the fine solver $\mathscr{F}$. This is the same setting as Pentland et al. [Pen+23], which allows almost direct comparison, although our system is the normalized version of the above, which also applies to u_0. We use a (normalized) error $\epsilon = 5e^{-7}$.

Systems: Rossler

The Rossler is a model for turbulence [Rös76]

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

When $(\hat{a}, \hat{b}, \hat{c}) = (0.2, 0.2, 5.7)$, it exhibits chaotic behavior. This configuration is commonly used throughout the literature. We integrate over $t \in [0, 340]$ using $N = 40$ intervals, taking $u_0 = (0, -6.78, 0.02)$ as initial condition. We use Runge-Kutta 1 with $9e^4$ steps for the coarse solver $\mathscr{G}$, and Ruge-Kutta 4 with $4.5e^8$ steps for the fine solver $\mathscr{F}$. This is the same setting as Pentland et al. [Pen+23], although, like above, we use the normalized version and set a normalized $\epsilon = 5e^{-7}$.

Systems: Non-linear Hopf bifurcation

This is a non-linear model for the study of Hopf bifurcations, see Seydel [Sey09, pg. 72] for a detailed explanation. The model is defined by the following equations

$$\frac{du_1}{dt} = -u_2 + u_1(\frac{t}{T} - u_1^2 - u_2^2), \quad \frac{du_2}{dt} = u_1 + u_2(\frac{t}{T} - u_1^2 - u_2^2), \quad (4)$$

where we note the dependence on time. To counter that, we add time as an additional coordinate, thus yielding a $d = 3$ system. We integrate over $t \in [-20, 500]$ using $N = 32$ intervals, taking $u_0 = (0.1, 0.1, 500)$ as initial condition. We use Runge-Kutta 1 with 2048 steps for the coarse solver $\mathscr{G}$, and Ruge-Kutta 8 with $5.12e^5$ steps for the fine solver $\mathscr{F}$. This is the same setting as Pentland et al. [Pen+23], although, like above, we use the normalized version and set a normalized $\epsilon = 5e^{-7}$.

Systems: Brusselator

The Brusselator models an autocatalytic chemical reaction [LN71]. It is a stiff, non-linear ODE, and the following equations govern it

$$\frac{du_1}{dt} = A + u_1^2 u_2 - (B+1)u_1,$$
$$\frac{du_2}{dt} = Bu_1 - u_1^2 u_2,$$

where $(A, B) = (1, 3)$. We integrate over $t \in [0, 100]$ using $N = 32$ intervals, taking $u_0 = (1, 3.7)$ as initial condition. We use Runge-Kutta 4 with $2.5e^2$ steps for the coarse solver $\mathscr{G}$, and Ruge-Kutta 4 with $2.5e^4$ steps for the fine solver $\mathscr{F}$. We use the normalized version and set a normalized $\epsilon = 5e^{-7}$.

Systems: Double pendulum

This is a model for a double pendulum, adapted from Danby [Dan97]. It consists of a simple pendulum of mass m and rod length ℓ connected to another simple pendulum of equal mass m and rod length ℓ, acting under gravity g. The model is defined by the following equations

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

where

$$\begin{aligned}
f_1(u_1,u_2) &= \sin(u_1-u_2)\cos(u_1-u_2), \\
f_2(u_1,u_2) &= 2 - \cos^2(u_1-u_2).
\end{aligned}$$

Systems: Double pendulum

In the above, m, ℓ, and g have been scaled out of the system by letting $\ell = g$. The variables u_1 and u_2 measure the angles between each pendulum and the vertical axis, while u_3 and u_4 measure the corresponding angular velocities.

The system exhibits chaotic behavior and is commonly used in the literature. Based on the initial condition, it can be difficult to learn.

We integrate over $t \in [0, 80]$ using $N = 32$ intervals, taking $u_0 = (-0.5, 0, 0, 0)$ as initial condition. We use Runge-Kutta 1 with 3104 steps for the coarse solver $\mathscr{G}$, and Ruge-Kutta 8 with $2.17e^5$ steps for the fine solver $\mathscr{F}$. This is a similar setting as Pentland et al. [Pen+23, Figure 4.10], although, like above, we use the normalized version and set a normalized $\epsilon = 5e^{-7}$.

Systems: Lorenz

The Lorenz system is a simplified model for weather prediction [Lor63]. With the following parameters, it is a chaotic system governed by the equations

$$\begin{aligned}
\frac{du_1}{dt} &= \gamma_1 (u_2 - u_1), \\
\frac{du_2}{dt} &= \gamma_2 u_1 - u_1 u_3 - u_2, \\
\frac{du_3}{dt} &= u_1 u_2 - \gamma_3 u_3,
\end{aligned}$$

with $(\gamma_1, \gamma_2, \gamma_3) = (10, 28, 8/3)$. We integrate over $t \in [0, 18]$ using $N = 50$ intervals, taking $u_0 = (-15, -15, 20)$ as initial condition. We use Runge-Kutta 4 with $3e^2$ steps for the coarse solver $\mathscr{G}$, and Ruge-Kutta 4 with $2.25e^4$ steps for the fine solver $\mathscr{F}$. We use the normalized version and set a normalized $\epsilon = 5e^{-7}$.

Systems: Thomas labyrinth

Thomas [Tho99] has proposed a particularly simple three-dimensional system representative of a large class of auto-catalytic models that occur frequently in chemical reactions [Ras+90], ecology [DG89], and evolution [Kau93]. It is described by the following equations

$$\begin{cases} \frac{dx}{dt} = b \sin y - ax, \\ \frac{dy}{dt} = b \sin z - ay, \\ \frac{dz}{dt} = b \sin x - az, \end{cases} \tag{5}$$

where $(a, b) = (0.5, 10)$. We integrate over $t \in [0, 10]$ for $N = 32, 64$, $t \in [0, 40]$ for $N = 128$, and $t \in [0, 100]$ for $N = 256, 512$ intervals. Following Gilpin [Gil21], we take

$$u_0 = (4.6722764, 5.2437205e^{-10}, -6.4444208e^{-10})$$

as initial condition, for which the system exhibits chaotic dynamics. Further, we use Runge-Kutta 1 with $10N$ steps for the coarse solver $\mathscr{G}$ and Ruge-Kutta 4 with $1e^9$ steps for the fine solver $\mathscr{F}$.

Systems: Viscous Burgers' equation

The viscous Burgers' equation is a fundamental PDE describing convection-diffusion occurring in various areas of applied mathematics. It is one-dimensional and defined as

$$v_t = \nu v_{xx} - v v_x \quad (x,t) \in (-L, L) \times (t_0, t_N], \tag{6}$$

with initial condition $v(x, t_0) = v_0(x), x \in [-L, L]$, and boundary conditions

$$v(-L, t) = v(L, t), \quad v_x(-L, t) = v_x(L, t), \quad t \in [t_0, T_N].$$

In the above, ν is the diffusion coefficient. We discretize the spatial domain using finite difference [For88] and $d + 1$ equally spaced points $x_{j+1} = x_j + \Delta x$, where $\Delta x = 2L/d$ and $j = 0, ..., d$.

Systems: Viscous Burgers' equation

In the numerical experiments, we consider two values for the time horizon, $t_N = 5$ and $t_N = 5.9$, with $t_0 = 0$. We set $N = d = 128$ and take $L = 1$ and $\nu = 1/100$. The discretization and finite difference formulation imply that it is equivalent to solving a d-dimensional system of ODEs.

We take $v_0(x) = 0.5(\cos(\frac{9}{2}\pi x) + 1)$ as the initial condition. We use Runge-Kutta 1 with $4N$ steps for the coarse solver $\mathscr{G}$ and Ruge-Kutta 8 with $5.12e^6$ steps for the fine solver $\mathscr{F}$.

We use the normalized version with a normalized $\epsilon = 5e^{-7}$.

Systems: FitzHugh-Nagumo PDE

The two-dimensional, non-linear FitzHugh-Nagumo PDE model [AF09] is an extension of the ODE system in Slide 50. It represents a set of cells constituted by a small nucleus of pacemakers near the origin immersed among an assembly of excitable cells. The simpler FHN ODE system only considers one cell and its corresponding spike generation behavior.

It is defined as

$$\begin{aligned} v_t &= a\nabla^2 v + v - v^3 - w - c, \quad (x,t) \in (-L,L)^2 \times (t_0, t_N] \\ w_t &= \tau\left(b\nabla^2 w + v - w\right), \end{aligned} \tag{7}$$

with initial conditions

$$v(x,t_0) = v_0(x), w(x,t_0) = w_0(x), \quad x \in [-L,L],$$

Systems: FitzHugh-Nagumo PDE

and boundary conditions

$$
\begin{aligned}
v((x,-L),t) &= v((x,L),t) \\
v((-L,y),t) &= v((L,y),t) \\
v_y((x,-L),t) &= v_y((x,L),t) \\
v_x((-L,y),t) &= v_x((L,y),t), \quad t \in [t_0, t_N].
\end{aligned}
$$

The boundary conditions for w are equivalent and not repeated. We discretize both spatial dimensions using finite difference and $\tilde{d}$ equally spaced points, yielding an ODE with $d = 2\tilde{d}^2$ dimensions.

Systems: FitzHugh-Nagumo PDE

In the numerical experiments, we consider four values for $\tilde{d} = 10, 12, 14, 16$, corresponding to $d = 200, 288, 392, 512$. We set $N = 512$, $L = 1$, $t_0 = 0$, and take $v_0(x), w(0)$ randomly sampled from $[0, 1]^d$ as the initial condition.

We use Ruge-Kutta 8 with 10^8 steps for the fine solver $\mathscr{F}$. We use the normalized version with a normalized $\epsilon = 5e^{-7}$.

The time span and coarse solvers depend on $\tilde{d}$, Table 2 describes their relation. This is to provide a realistic experiment where the user would need to adjust the coarse solver based on $t_N - t_0$.

Systems: FitzHugh-Nagumo PDE

d	$\mathscr{G}$	$\mathscr{G}$ steps	t_N
200	RK2	$3N$	$t_N = 150$
288	RK2	$12N$	$t_N = 550$
392	RK2	$25N$	$t_N = 950$
512	RK4	$25N$	$t_N = 1100$

Table: Simulation setup for the two-dimensional FitzHugh-Nagumo PDE. Adjusting the coarse solver based on the time horizon t_N makes the simulation more realistic.