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

K. Pentland[1], M. Tamborrino[2], T. J. Sullivan[1,3], J. Buchanan[4], and L. C. Appel[4]

[1]Mathematics Institute, University of Warwick

[2]Department of Statistics, University of Warwick

[3]Alan Turing Institute, London

[4]Culham Centre for Fusion Energy, UKAEA

WARWICK

What are we doing?

- We seek numerical solutions $\boldsymbol{U}_j \approx \boldsymbol{u}(t_j)$ to a system of $d \in \mathbb{N}$ (nonlinear) ordinary differential equations (ODEs):

$$\boxed{\frac{d\boldsymbol{u}}{dt} = \boldsymbol{f}\big(\boldsymbol{u}(t), t\big) \qquad \boldsymbol{u}(t_0) = \boldsymbol{u}^0 \qquad t \in [t_0, t_J]}$$

What are we doing?

- We seek numerical solutions $\boldsymbol{U}_j \approx \boldsymbol{u}(t_j)$ to a system of $d \in \mathbb{N}$ (nonlinear) ordinary differential equations (ODEs):

$$\boxed{\frac{d\boldsymbol{u}}{dt} = \boldsymbol{f}\big(\boldsymbol{u}(t), t\big) \qquad \boldsymbol{u}(t_0) = \boldsymbol{u}^0 \qquad t \in [t_0, t_J]}$$

- We want to integrate problems where any one (or more) of the following hold:
  - (i) the interval of integration, $[t_0, t_J]$
  - (ii) the number of mesh points, $J+1$
  - (iii) the wallclock time to evaluate the vector field, $\boldsymbol{f}$

  is so **large**, that locating a solution takes **hours, days, or even weeks** using **sequential** methods (i.e. Runge-Kutta).

(I) Motivation and aims

What are we doing?

- We seek numerical solutions $\boldsymbol{U}_j \approx \boldsymbol{u}(t_j)$ to a system of $d \in \mathbb{N}$ (nonlinear) ordinary differential equations (ODEs):

$$\boxed{\frac{d\boldsymbol{u}}{dt} = \boldsymbol{f}\big(\boldsymbol{u}(t), t\big) \qquad \boldsymbol{u}(t_0) = \boldsymbol{u}^0 \qquad t \in [t_0, t_J]}$$

- We want to integrate problems where any one (or more) of the following hold:
  - (i) the interval of integration, $[t_0, t_J]$
  - (ii) the number of mesh points, $J+1$
  - (iii) the wallclock time to evaluate the vector field, $\boldsymbol{f}$

  is so **large**, that locating a solution takes **hours, days, or even weeks** using **sequential** methods (i.e. Runge-Kutta).

- For example: IVPs for simulating magnetically confined fusion plasmas **over one second** can take **100 days to run**...

Take home message

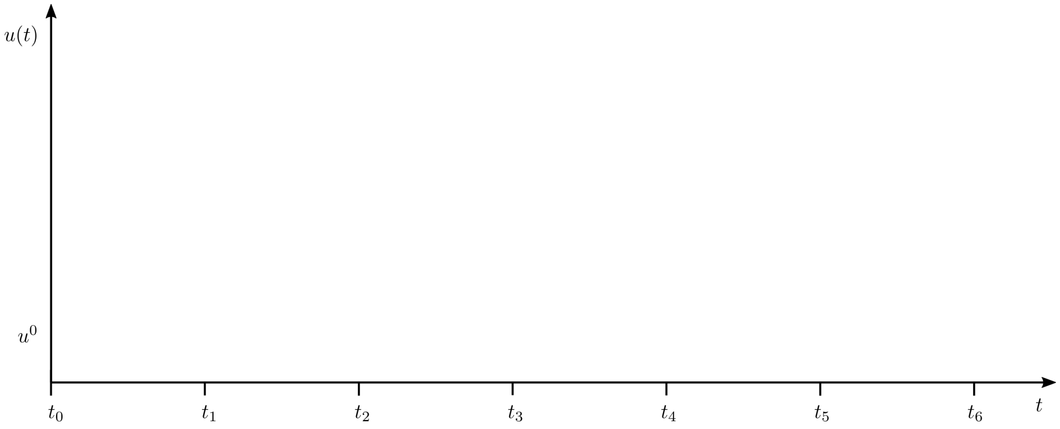
Our goal is to develop **new time-parallel algorithms** using **probabilistic methods** to solve IVPs **faster**.

What will we cover today?

- What is a **time-parallel method**? $\rightarrow$ **parareal** (an existing method).
- Introduce **GParareal** (our method) that combines **parareal** and **Gaussian process emulation**.
- Illustrate that GParareal **performs favourably** compared to **parareal** $\rightarrow$ **additional parallel speedup**.
- Highlight some **open problems** surrounding GParareal.

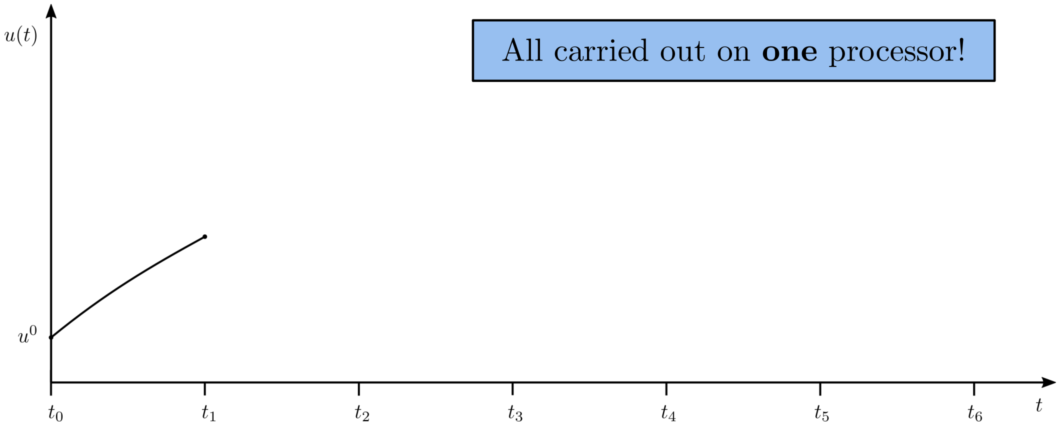
- **Consider:** scalar ODE $\rightarrow$ usually we integrate **sequentially** with some RK method, call it $\mathbf{F}$, on a **single processor**.

$$\boxed{\frac{du}{dt} = f\big(u(t), t\big) \qquad u(t_0) = u^0 \qquad t \in [t_0, t_J]}$$



- **Consider:** scalar ODE $\rightarrow$ usually we integrate **sequentially** with some RK method, call it **F**, on a **single processor**.
  - Time is inherently sequential $\rightarrow$ solution at future steps depend on the past!

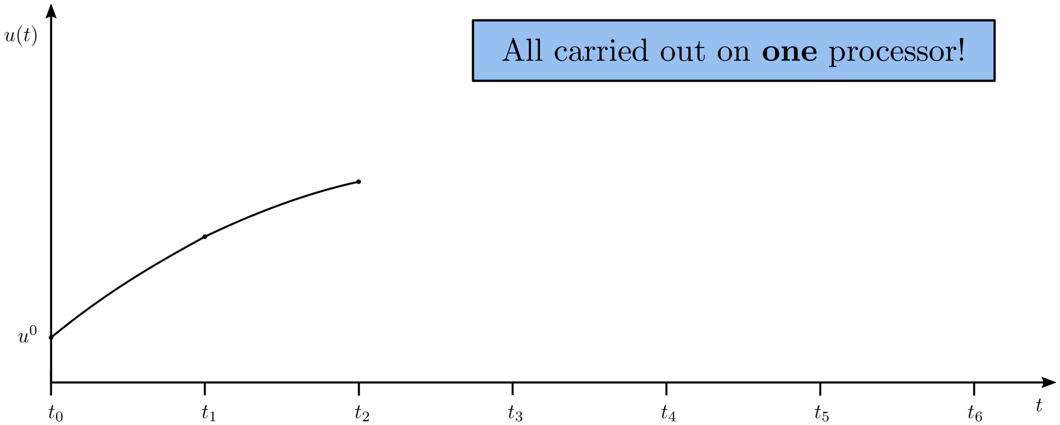
$$\boxed{\frac{du}{dt} = f\big(u(t), t\big) \qquad u(t_0) = u^0 \qquad t \in [t_0, t_J]}$$



All carried out on **one** processor!

- **Consider:** scalar ODE $\rightarrow$ usually we integrate **sequentially** with some RK method, call it $\mathbf{F}$, on a **single processor**.
  - Time is inherently sequential $\rightarrow$ solution at future steps depend on the past!

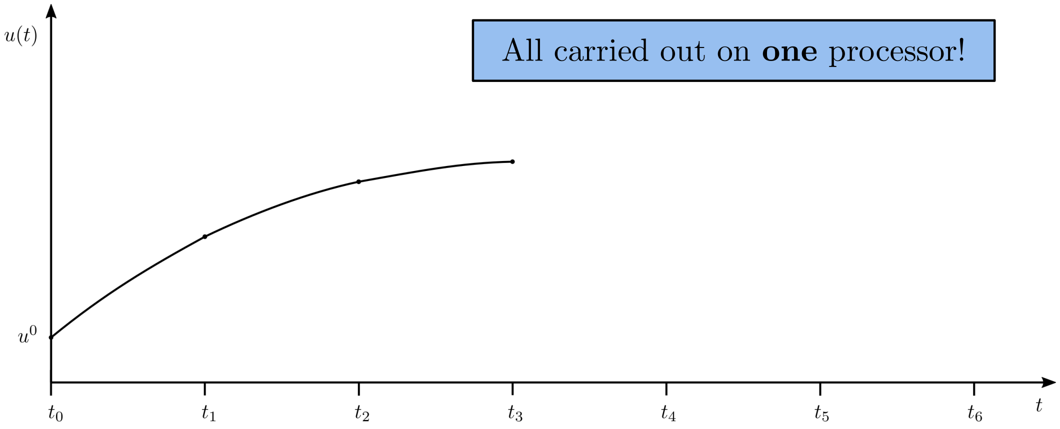
$$\boxed{\frac{du}{dt} = f\big(u(t), t\big) \qquad u(t_0) = u^0 \qquad t \in [t_0, t_J]}$$



All carried out on **one** processor!

- **Consider:** scalar ODE $\rightarrow$ usually we integrate **sequentially** with some RK method, call it **F**, on a **single processor**.
  - Time is inherently sequential $\rightarrow$ solution at future steps depend on the past!

$$\boxed{\frac{du}{dt} = f\big(u(t), t\big) \qquad u(t_0) = u^0 \qquad t \in [t_0, t_J]}$$

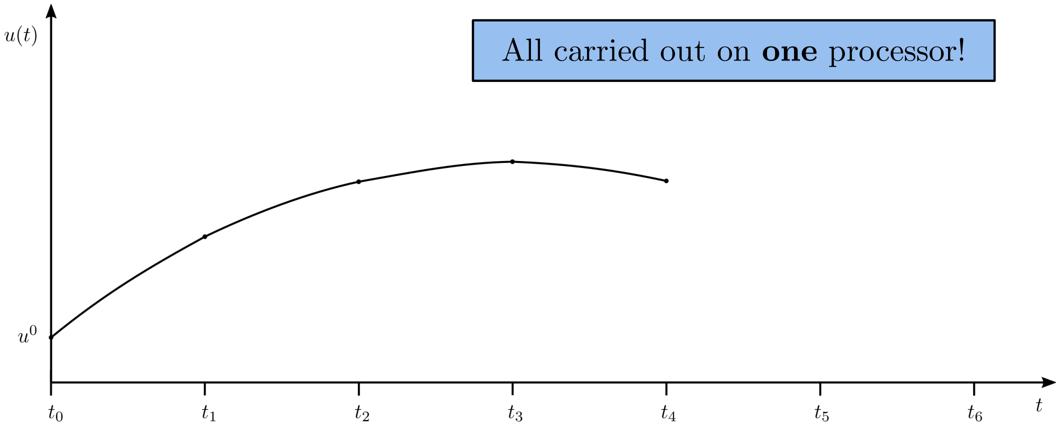


All carried out on **one** processor!

(II) Time-parallel methods

- **Consider:** scalar ODE $\rightarrow$ usually we integrate **sequentially** with some RK method, call it **F**, on a **single processor**.
  - Time is inherently sequential $\rightarrow$ solution at future steps depend on the past!

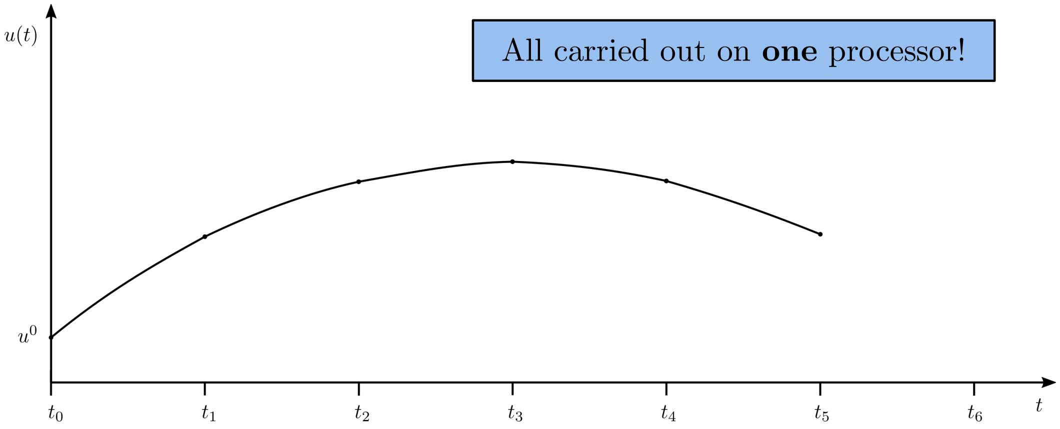
$$\boxed{\frac{du}{dt} = f\big(u(t), t\big) \qquad u(t_0) = u^0 \qquad t \in [t_0, t_J]}$$



All carried out on **one** processor!

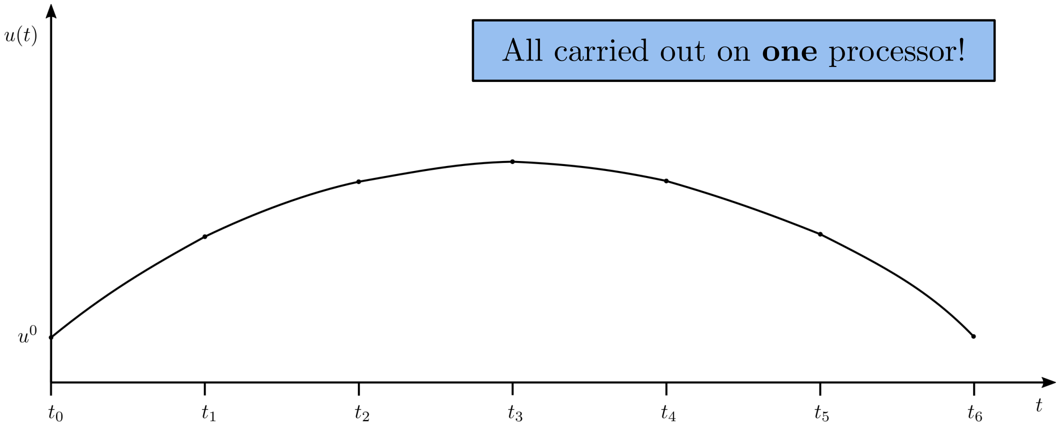
- **Consider:** scalar ODE $\rightarrow$ usually we integrate **sequentially** with some RK method, call it **F**, on a **single processor**.
  - Time is inherently sequential $\rightarrow$ solution at future steps depend on the past!

$$\boxed{\frac{du}{dt} = f\big(u(t), t\big) \qquad u(t_0) = u^0 \qquad t \in [t_0, t_J]}$$



- **Consider:** scalar ODE $\rightarrow$ usually we integrate **sequentially** with some RK method, call it $\mathbf{F}$, on a **single processor**.
  - Time is inherently sequential $\rightarrow$ solution at future steps depend on the past!

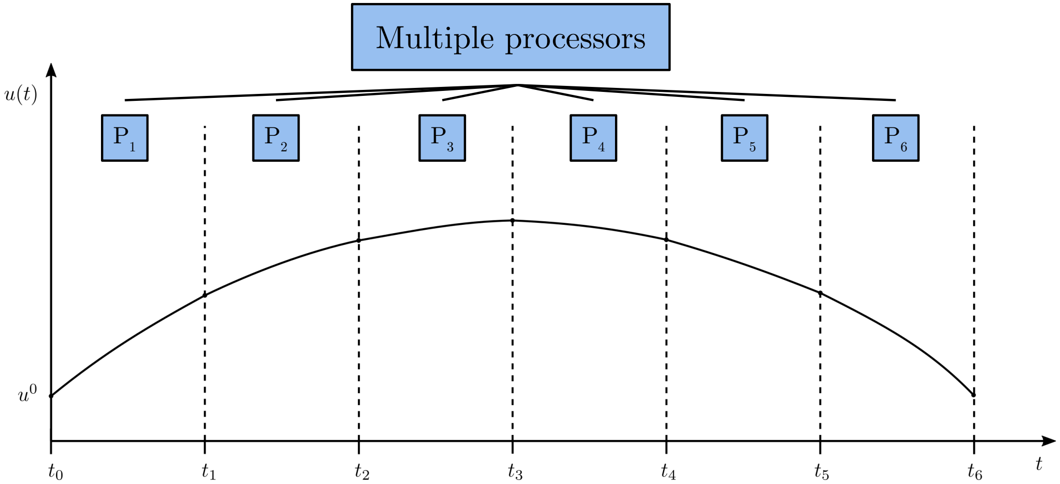
$$\boxed{\frac{du}{dt} = f\big(u(t), t\big) \qquad u(t_0) = u^0 \qquad t \in [t_0, t_J]}$$



(II) Time-parallel methods

- **Consider:** scalar ODE $\rightarrow$ usually we integrate **sequentially** with some RK method, call it **F**, on a **single processor**.
  - Time is inherently sequential $\rightarrow$ solution at future steps depend on the past!
  - With $\boldsymbol{J}$ **processors** we could partition into $\boldsymbol{J}$ sub-problems and solve **in parallel** directly.

$$\boxed{\frac{du}{dt} = f\big(u(t), t\big) \qquad u(t_0) = u^0 \qquad t \in [t_0, t_J]} \longrightarrow \boxed{\frac{du_j}{dt} = f\big(u_j(t), t\big) \quad u_j(t_j) = U_j \qquad t \in [t_j, t_{j+1}]}$$

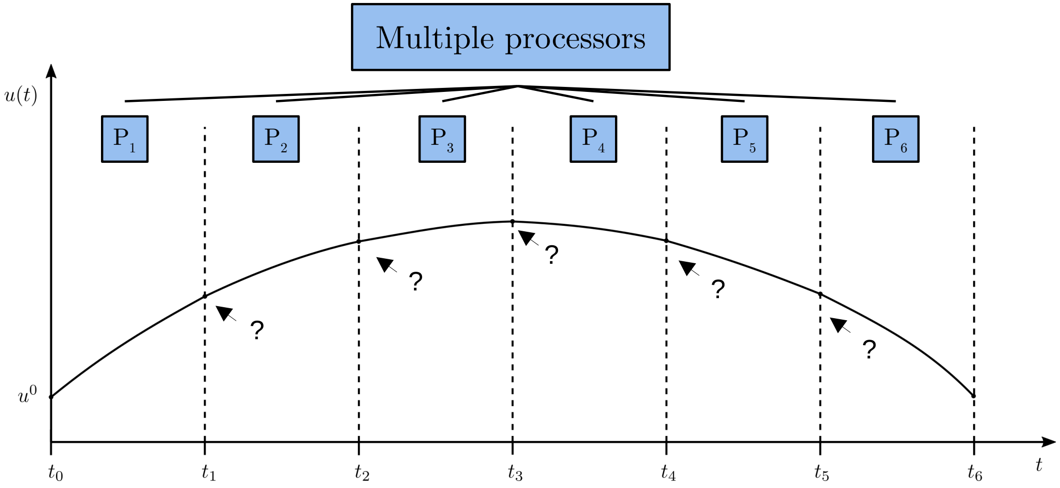


(II) Time-parallel methods

- **Consider:** scalar ODE $\rightarrow$ usually we integrate **sequentially** with some RK method, call it $\mathbf{F}$, on a **single processor**.
  - Time is inherently sequential $\rightarrow$ solution at future steps depend on the past!
  - With $\boldsymbol{J}$ **processors** we could partition into $\boldsymbol{J}$ sub-problems and solve **in parallel** directly.

$$\boxed{\frac{du}{dt} = f\big(u(t), t\big) \qquad u(t_0) = u^0 \qquad t \in [t_0, t_J]} \longrightarrow \boxed{\frac{du_j}{dt} = f\big(u_j(t), t\big) \quad u_j(t_j) = U_j \qquad t \in [t_j, t_{j+1}]}$$

- However, only **one initial condition** is known!
- How do we find the others?

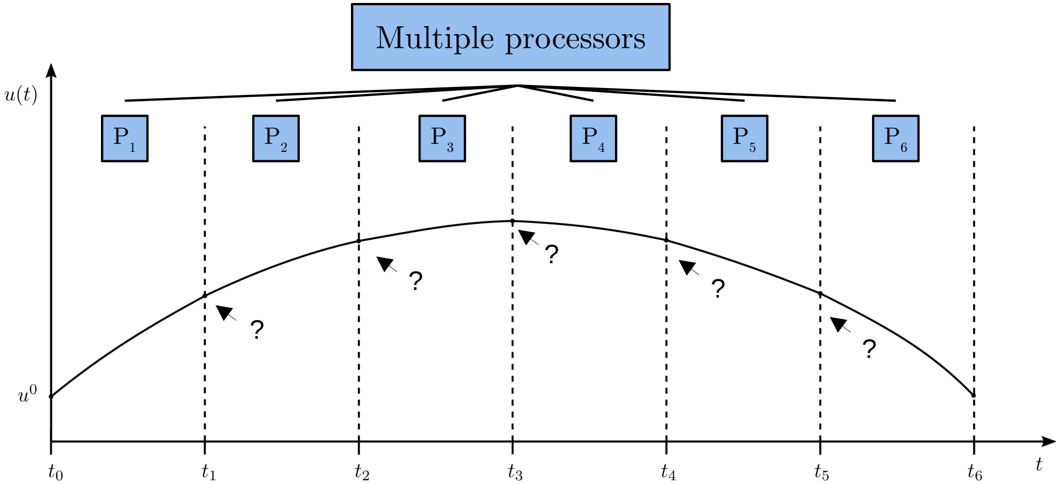


- **Consider:** scalar ODE $\rightarrow$ usually we integrate **sequentially** with some RK method, call it $\mathbf{F}$, on a **single processor**.
  - Time is inherently sequential $\rightarrow$ solution at future steps depend on the past!
  - With $\boldsymbol{J}$ **processors** we could partition into $\boldsymbol{J}$ sub-problems and solve **in parallel** directly.

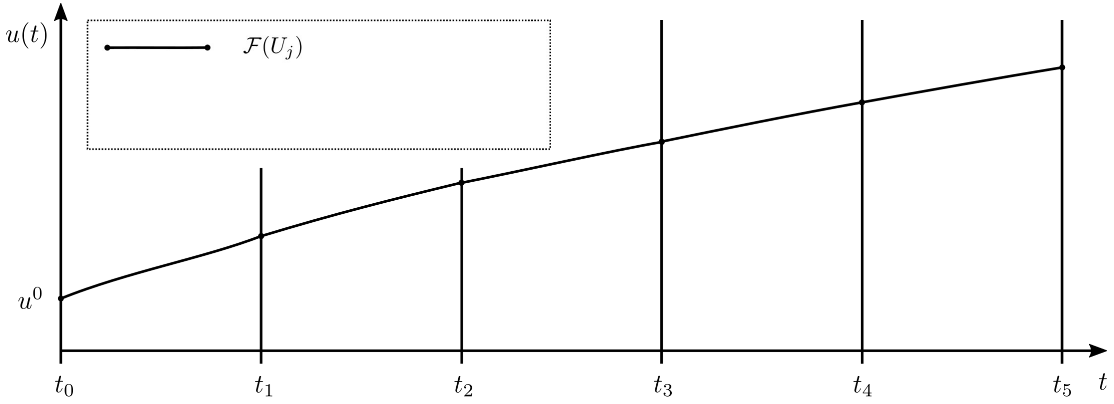
$$\boxed{\frac{du}{dt} = f\big(u(t), t\big) \quad u(t_0) = u^0 \qquad t \in [t_0, t_J]} \longrightarrow \boxed{\frac{du_j}{dt} = f\big(u_j(t), t\big) \quad u_j(t_j) = U_j \qquad t \in [t_j, t_{j+1}]}$$

- However, only **one initial condition** is known!
- How do we find the others?

- Many time-parallel methods:
  - Direct.
  - Waveform-relaxation.
  - Multigrid/multiple shooting (our focus).



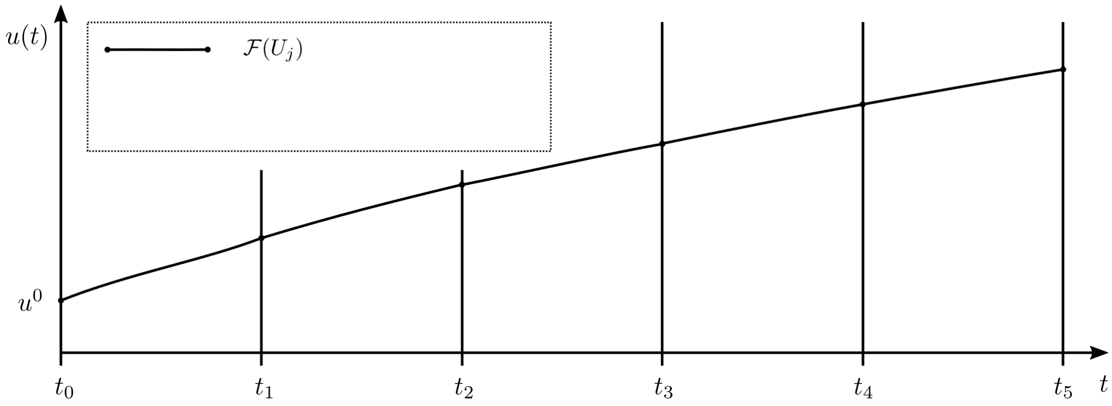
Aim: locate the J initial values (from before) **iteratively.**



Aim: locate the J initial values (from before) **iteratively.**

How? using **two sequential integrators**:

- Fine solver $\mathbf{F}$ (high accuracy/slow execution)
- Coarse solver $\mathbf{G}$ (low accuracy/fast execution)

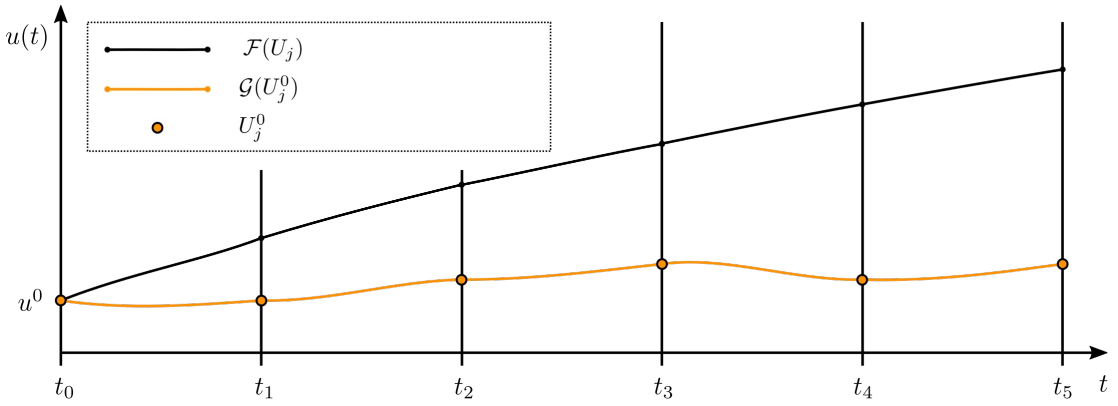


Aim: locate the J initial values (from before) **iteratively.**

1) Run **G serially** (yellow).

How? using **two sequential integrators**:

- Fine solver **F** (high accuracy/slow execution)
- Coarse solver **G** (low accuracy/fast execution)



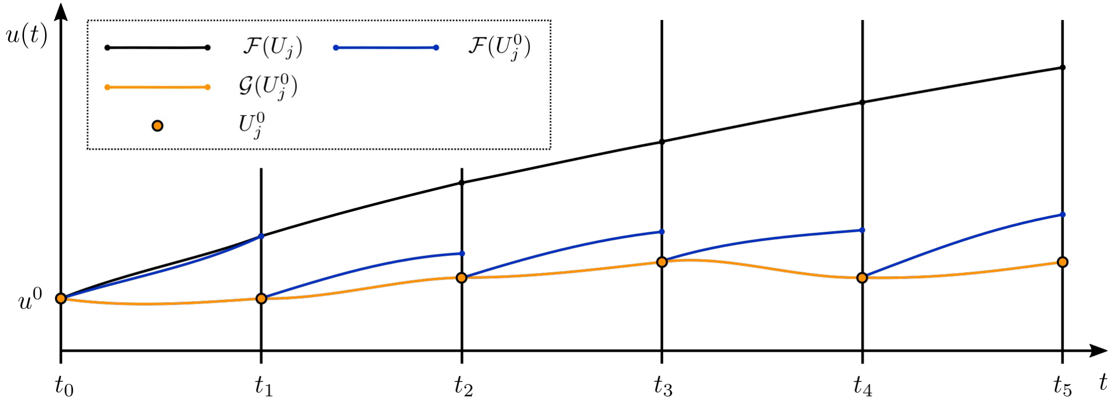
Aim: locate the J initial values (from before) **iteratively.**

1) Run **G serially** (yellow).

2) Using these conditions, run **F in parallel** (blue).

How? using **two sequential integrators**:

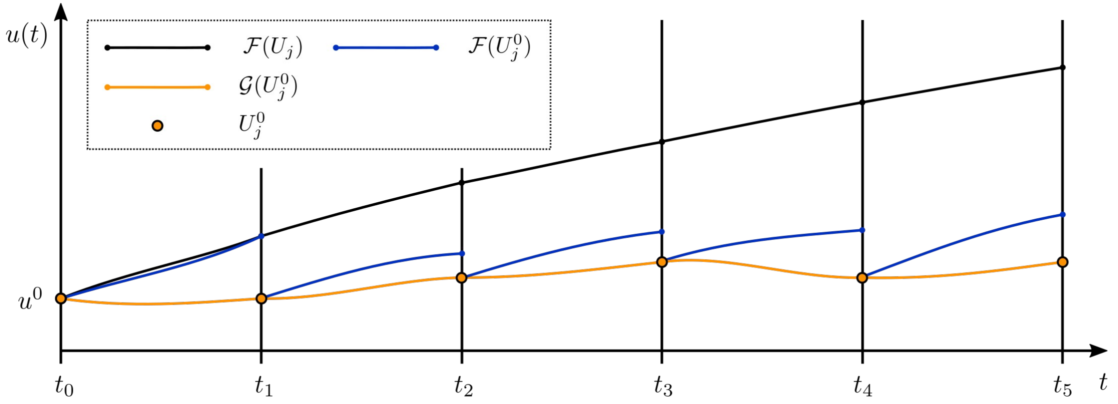
- Fine solver **F** (high accuracy/slow execution)
- Coarse solver **G** (low accuracy/fast execution)



Aim: locate the J initial values (from before) **iteratively.**

1) Run **G serially** (yellow).
2) Using these conditions, run **F in parallel** (blue).
3) Predictor-corrector **(PC)** step: **predict with G** (red) and **correct using difference of previous F and G**:

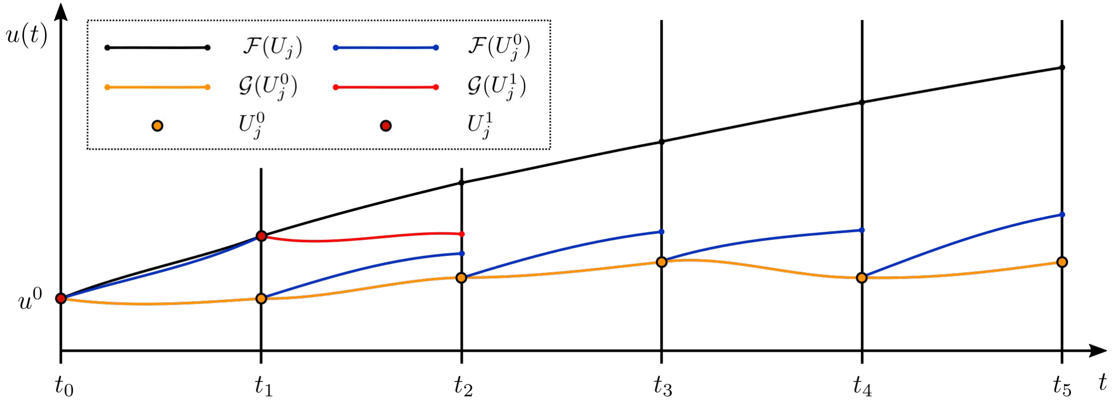
$$\boxed{U_j^k = \underbrace{\mathcal{G}(U_{j-1}^k)}_{\text{Prediction}} + \underbrace{\mathcal{F}(U_{j-1}^{k-1}) - \mathcal{G}(U_{j-1}^{k-1})}_{\text{Correction}}}$$



Aim: locate the J initial values (from before) **iteratively.**

1) Run **G serially** (yellow).
2) Using these conditions, run **F in parallel** (blue).
3) Predictor-corrector **(PC)** step: **predict with G** (red) and **correct using difference of previous F and G**:

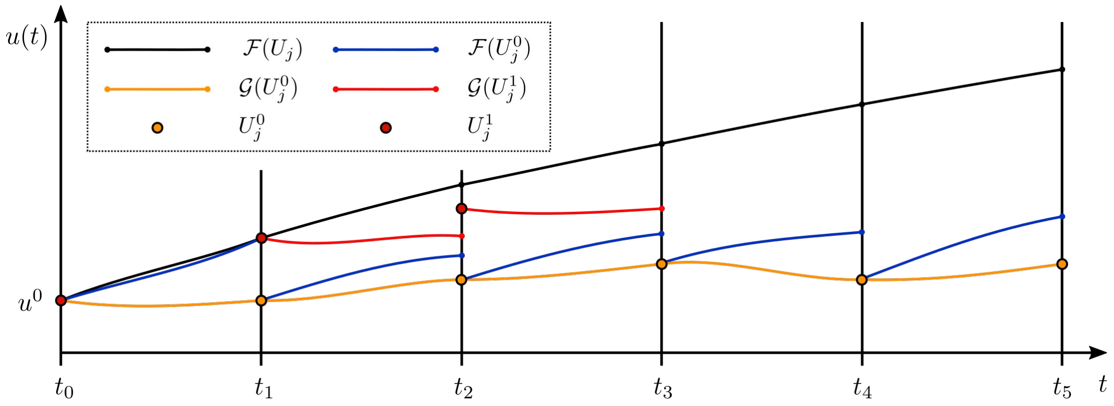
$$\boxed{U_j^k = \underbrace{\mathcal{G}(U_{j-1}^k)}_{\text{Prediction}} + \underbrace{\mathcal{F}(U_{j-1}^{k-1}) - \mathcal{G}(U_{j-1}^{k-1})}_{\text{Correction}}}$$



Aim: locate the J initial values (from before) **iteratively.**

1) Run **G serially** (yellow).
2) Using these conditions, run **F in parallel** (blue).
3) Predictor-corrector **(PC)** step: **predict with G** (red) and **correct using difference of previous F and G**:

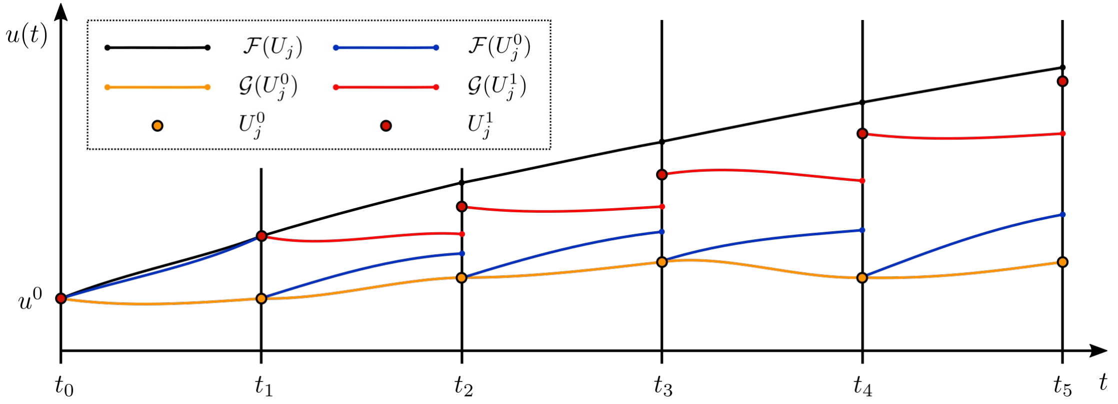
$$\boxed{U_j^k = \underbrace{\mathcal{G}(U_{j-1}^k)}_{\text{Prediction}} + \underbrace{\mathcal{F}(U_{j-1}^{k-1}) - \mathcal{G}(U_{j-1}^{k-1})}_{\text{Correction}}}$$



Aim: locate the J initial values (from before) **iteratively.**

1) Run **G serially** (yellow).
2) Using these conditions, run **F in parallel** (blue).
3) Predictor-corrector **(PC)** step: **predict with G** (red) and **correct using difference of previous F and G**:

$$\boxed{U_j^k = \underbrace{\mathcal{G}(U_{j-1}^k)}_{\text{Prediction}} + \underbrace{\mathcal{F}(U_{j-1}^{k-1}) - \mathcal{G}(U_{j-1}^{k-1})}_{\text{Correction}}}$$



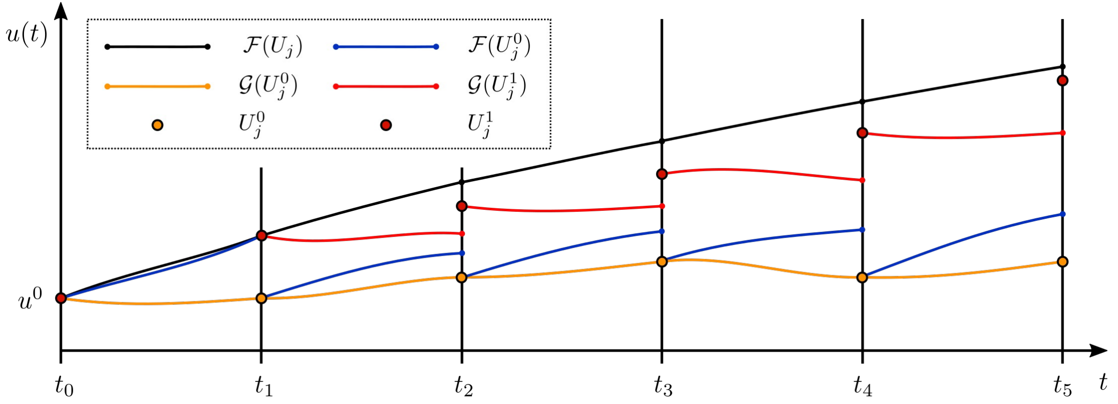
Aim: locate the J initial values (from before) **iteratively.**

1) Run **G serially** (yellow).
2) Using these conditions, run **F in parallel** (blue).
3) Predictor-corrector **(PC)** step: **predict with G** (red) and **correct using difference of previous F and G**:

$$\boxed{U_j^k = \underbrace{\mathcal{G}(U_{j-1}^k)}_{\text{Prediction}} + \underbrace{\mathcal{F}(U_{j-1}^{k-1}) - \mathcal{G}(U_{j-1}^{k-1})}_{\text{Correction}}}$$

4) Solution given by red dots. **Repeat** steps 2 and 3 until desired tolerance reached:

$$\|U_j^k - U_j^{k-1}\|_\infty < \varepsilon \quad \forall j \leq J$$



Aim: locate the J initial values (from before) **iteratively.**

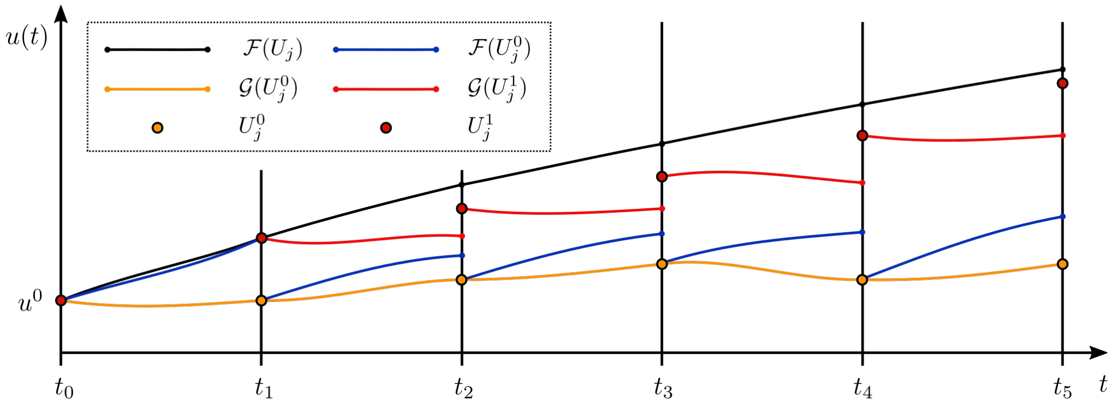
1) Run **G serially** (yellow).
2) Using these conditions, run **F in parallel** (blue).
3) Predictor-corrector **(PC)** step: **predict with G** (red) and **correct using difference of previous F and G**:

$$U_j^k = \underbrace{\mathcal{G}(U_{j-1}^k)}_{\text{Prediction}} + \underbrace{\mathcal{F}(U_{j-1}^{k-1}) - \mathcal{G}(U_{j-1}^{k-1})}_{\text{Correction}}$$

Take home: parareal converges in $k < J$ iterations
$\rightarrow$ approx. speedup $= J/k$

4) Solution given by red dots. **Repeat** steps 2 and 3 until desired tolerance reached:

$$\|U_j^k - U_j^{k-1}\|_\infty < \varepsilon \quad \forall j \leq J$$



The predictor-corrector limitation

- Corrections based on **single previous iteration** $\rightarrow$ all other solution information is ignored.

The predictor-corrector limitation

- Corrections based on **single previous iteration** $\rightarrow$ all other solution information is ignored.

Our research: improve corrections using probabilistic methods to reduce number of iterations k.

The predictor-corrector limitation

- Corrections based on **single previous iteration** $\rightarrow$ all other solution information is ignored.

Our research: improve corrections using probabilistic methods to reduce number of iterations k.

How do we do this?

- We re-formulate the PC such that:

$$U_j^k = \overbrace{\mathcal{F}(\underbrace{U_{j-1}^k}_{\text{Known}})}^{\text{Unknown}} = (\mathcal{F} + \mathcal{G} - \mathcal{G})(U_{j-1}^k) = \underbrace{\mathcal{G}(U_{j-1}^k)}_{\text{Prediction}} + \underbrace{(\mathcal{F} - \mathcal{G})(U_{j-1}^k)}_{\text{Correction}}$$

The predictor-corrector limitation

- Corrections based on **single previous iteration** $\rightarrow$ all other solution information is ignored.

Our research: improve corrections using probabilistic methods to reduce number of iterations k.

How do we do this?

- We re-formulate the PC such that:

$$U_j^k = \overbrace{\mathcal{F}(\underbrace{U_{j-1}^k}_{\text{Known}})}^{\text{Unknown}} = (\mathcal{F}+\mathcal{G}-\mathcal{G})(U_{j-1}^k) = \underbrace{\mathcal{G}(U_{j-1}^k)}_{\text{Prediction}} + \underbrace{(\mathcal{F}-\mathcal{G})(U_{j-1}^k)}_{\text{Correction}}$$

Model using a GP emulator **trained** on **all previous evaluations** of **F** and **G**!

(which returns a Gaussian RV)

The predictor-corrector limitation

- Corrections based on **single previous iteration** $\rightarrow$ all other solution information is ignored.

Our research: improve corrections using probabilistic methods to reduce number of iterations k.

How do we do this?

- We re-formulate the PC such that:

$$U_j^k = \overbrace{\mathcal{F}(\underbrace{U_{j-1}^k}_{\text{Known}})}^{\text{Unknown}} = (\mathcal{F} + \mathcal{G} - \mathcal{G})(U_{j-1}^k) = \underbrace{\mathcal{G}(U_{j-1}^k)}_{\text{Prediction}} + \underbrace{(\mathcal{F} - \mathcal{G})(U_{j-1}^k)}_{\text{Correction}}$$

Model using a GP emulator **trained** on **all previous evaluations** of **F** and **G**!

(which returns a Gaussian RV)

New update rule

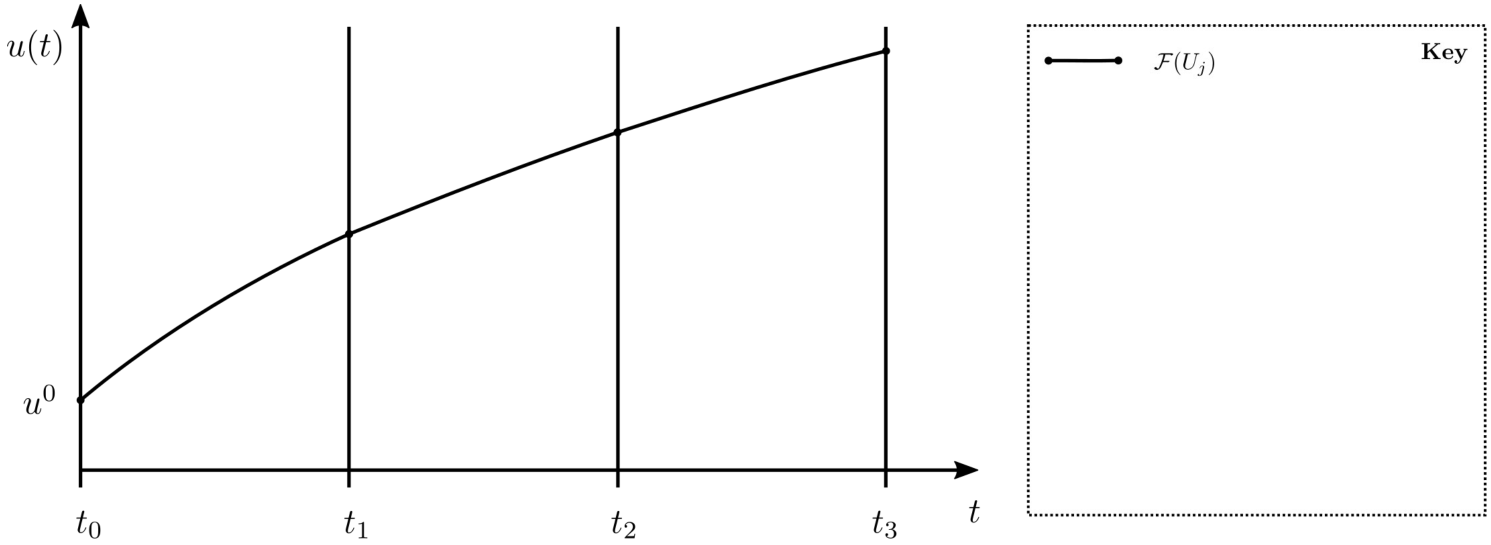
$$U_j^k = \mathcal{G}(U_{j-1}^k) + \mathbb{E}\left[(\mathcal{F} - \mathcal{G})(U_{j-1}^k)\right]$$

We **approximate** the RV by taking its **expected value**.

This ignores uncertainty in the GP $\rightarrow$ **open problem**

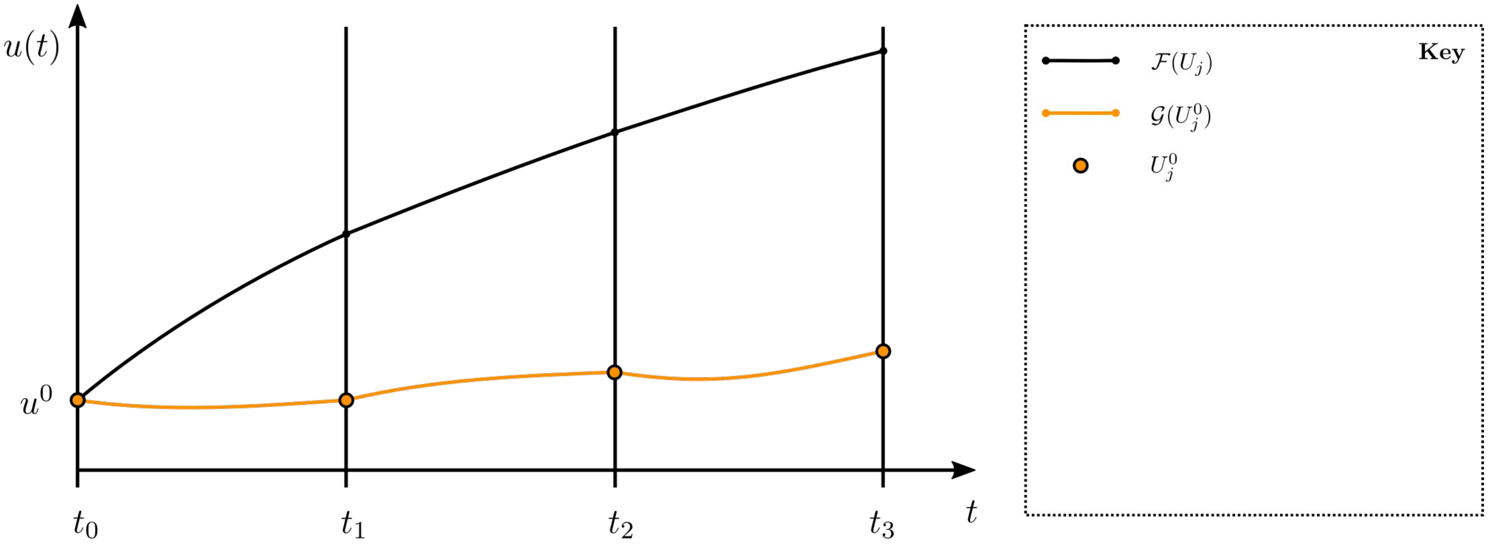
(IV) GParareal

How it works: very similar to parareal.



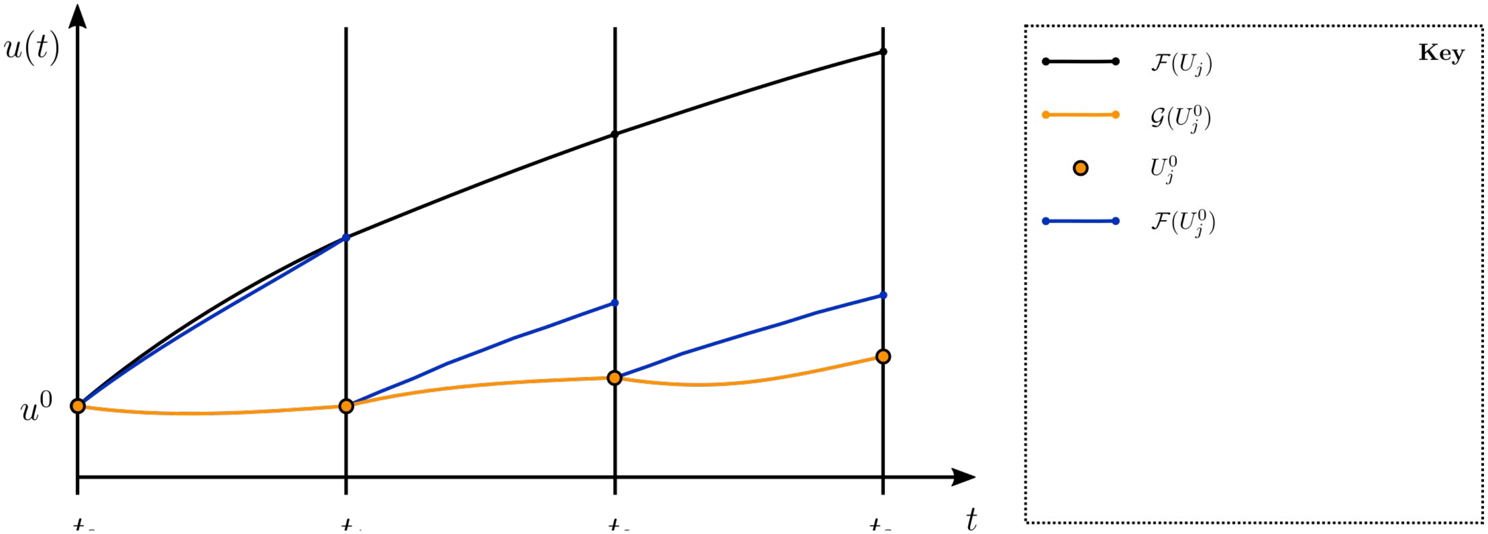
(IV) GParareal

How it works: very similar to parareal.



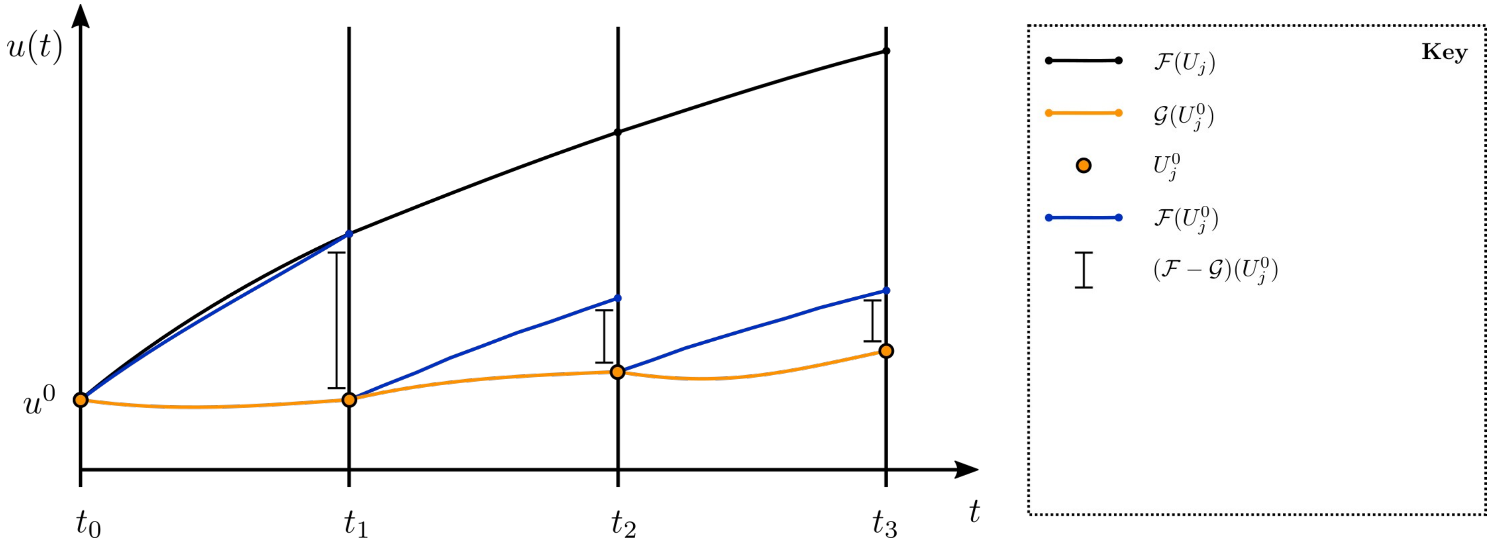
(IV) GParareal

How it works: very similar to parareal.



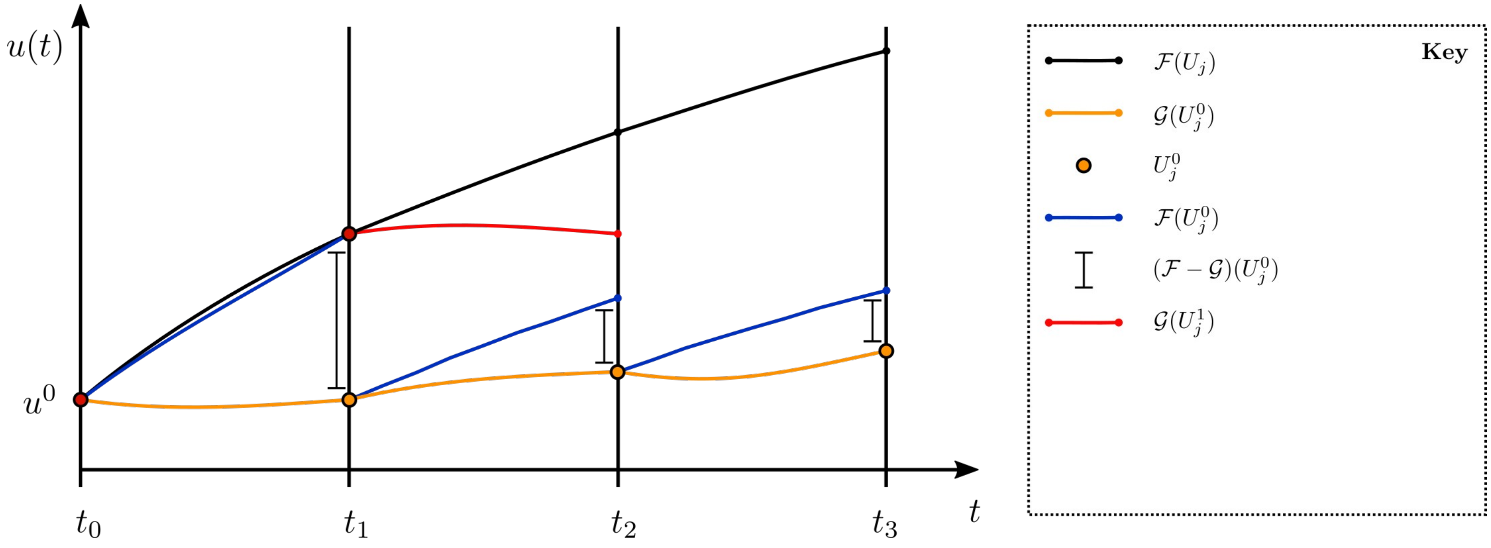
(IV) GParareal

How it works: very similar to parareal.



(IV) GParareal

How it works: very similar to parareal.

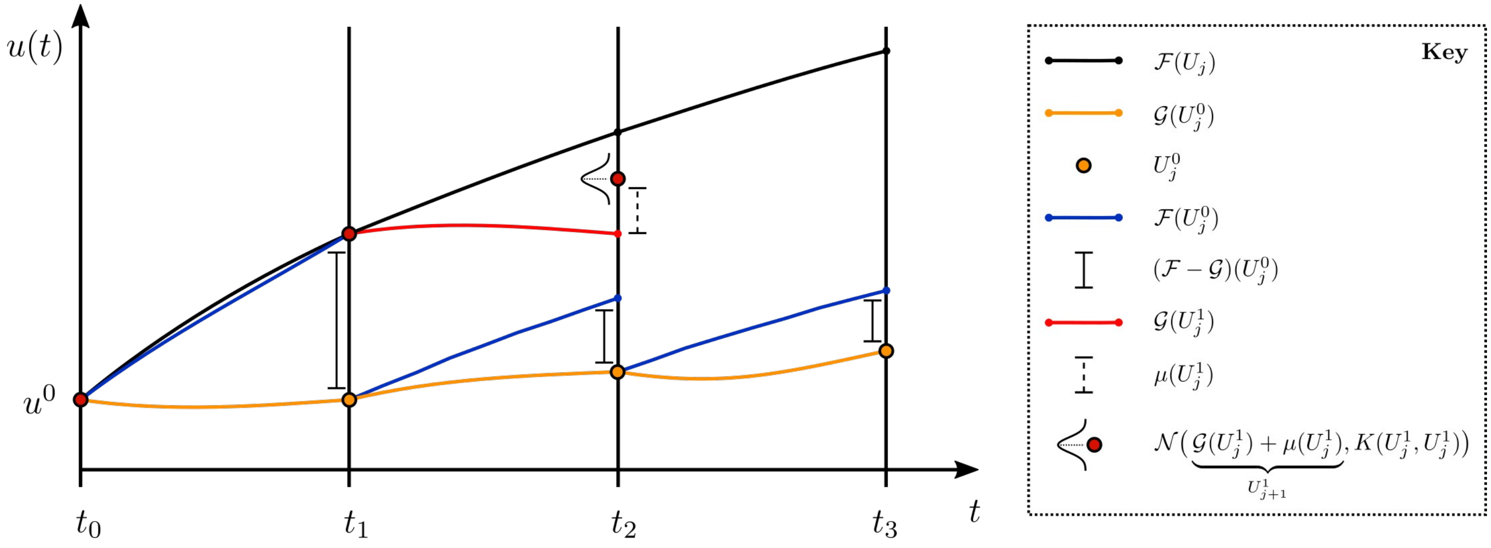


New update rule

$$U_j^k = \mathcal{G}(U_{j-1}^k) + \mathbb{E}[(\mathcal{F}-\mathcal{G})(U_{j-1}^k)]$$

(IV) GParareal

How it works: very similar to parareal.

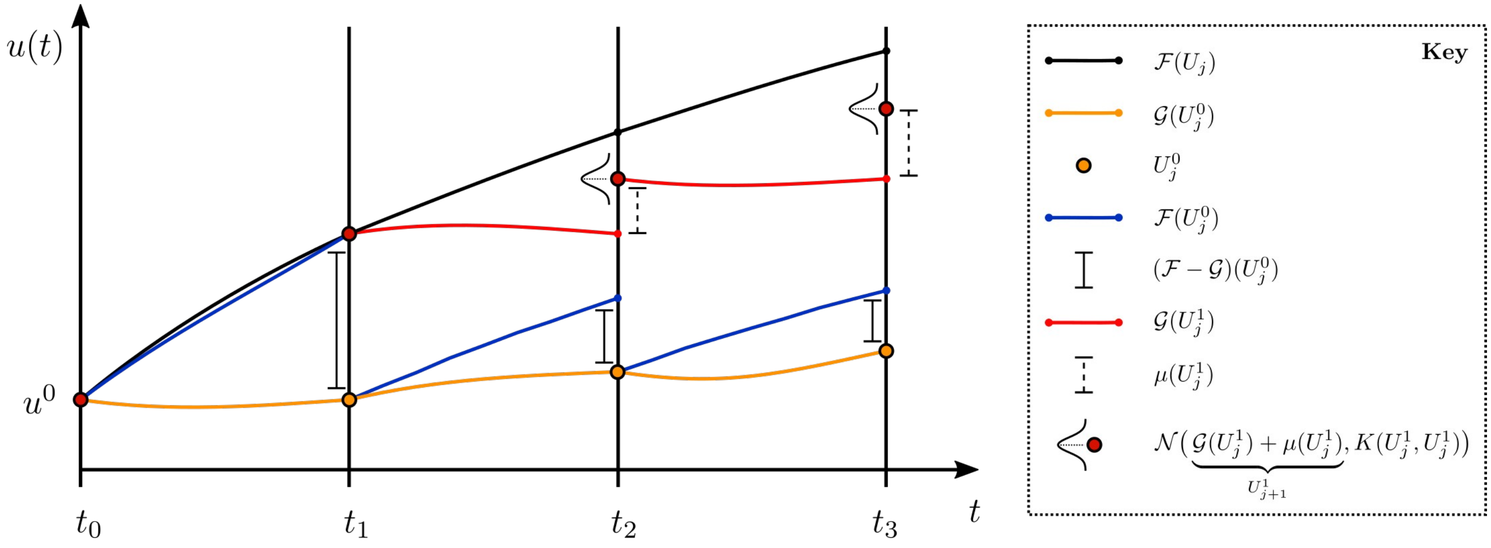


New update rule

$$U_j^k = \mathcal{G}(U_{j-1}^k) + \mathbb{E}\big[(\mathcal{F}-\mathcal{G})(U_{j-1}^k)\big]$$

(IV) GParareal

How it works: very similar to parareal.



New update rule

$$U_j^k = \mathcal{G}(U_{j-1}^k) + \mathbb{E}\big[(\mathcal{F}-\mathcal{G})(U_{j-1}^k)\big]$$

FitzHugh-Nagumo model

$$\frac{du_1}{dt} = c\left(u_1 - \frac{u_1^3}{3} + u_2\right) \qquad t \in [0, 40]$$

$$\frac{du_2}{dt} = -\frac{1}{c}(u_1 - a + bu_2) \qquad \cancel{\boldsymbol{u}(0) = (-1, 1)^\intercal}$$

Iterations until convergence $\boldsymbol{k}$ for various initial values

$$\boldsymbol{u}(0) \in [-1.25, 1.25]^2$$

(IV) GParareal

FitzHugh-Nagumo model

$$\frac{du_1}{dt} = c\left(u_1 - \frac{u_1^3}{3} + u_2\right) \qquad t \in [0, 40]$$

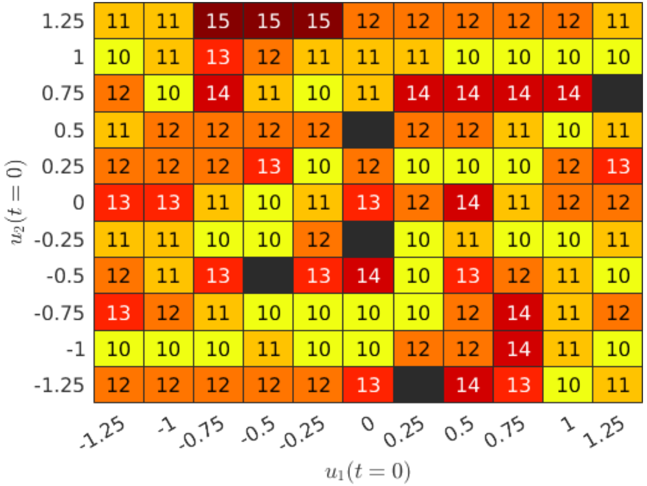
$$\frac{du_2}{dt} = -\frac{1}{c}(u_1 - a + bu_2) \qquad \cancel{\boldsymbol{u}(0) = (-1, 1)^\intercal}$$

Iterations until convergence $\boldsymbol{k}$ for various initial values

$$\boldsymbol{u}(0) \in [-1.25, 1.25]^2$$

Parareal

$\sim 4 \times$ speedup



FitzHugh-Nagumo model

$$\frac{du_1}{dt} = c\left(u_1 - \frac{u_1^3}{3} + u_2\right) \qquad t \in [0, 40]$$

$$\frac{du_2}{dt} = -\frac{1}{c}(u_1 - a + bu_2) \qquad \cancel{\boldsymbol{u}(0) = (-1, 1)^\intercal}$$

Iterations until convergence k for various initial values

$$\boldsymbol{u}(0) \in [-1.25, 1.25]^2$$

Parareal

$\sim 4\times$ speedup

$u_2(t=0)$ \ $u_1(t=0)$	-1.25	-1	-0.75	-0.5	-0.25	0	0.25	0.5	0.75	1	1.25
1.25	11	11	15	15	15	12	12	12	12	12	11
1	10	11	13	12	11	11	11	10	10	10	10
0.75	12	10	14	11	10	11	14	14	14	14	
0.5	11	12	12	12	12		12	12	11	10	11
0.25	12	12	12	13	10	12	10	10	10	12	13
0	13	13	11	10	11	13	12	14	11	12	12
-0.25	11	11	10	10	12		10	11	10	10	11
-0.5	12	11	13		13	14	10	13	12	11	10
-0.75	13	12	11	10	10	10	10	12	14	11	12
-1	10	10	10	11	10	10	12	12	14	11	10
-1.25	12	12	12	12	12	13		14	13	10	11

GParareal

$\sim 8\times$ speedup

$u_2(t=0)$ \ $u_1(t=0)$	-1.25	-1	-0.75	-0.5	-0.25	0	0.25	0.5	0.75	1	1.25
1.25	5	6	5	5	5	5	6	6	5	5	5
1	5	5	5	5	5	5	5	6	5	5	5
0.75	5	5	5	5	5	5	5	5	5	6	5
0.5	5	5	5	5	5	5	5	5	6	5	5
0.25	5	5	5	5	6	5	6	5	5	5	5
0	5	5	5	5	5	5	5	5	5	6	5
-0.25	5	6	5	5	5	6	5	5	5	5	5
-0.5	5	5	5	5	5	5	5	5	5	5	5
-0.75	5	5	5	5	5	5	5	5	5	5	5
-1	5	5	5	5	6	5	5	5	5	5	6
-1.25	5	5	5	5	5	5	5	5	5	5	6

FitzHugh-Nagumo model

$$\frac{du_1}{dt} = c(u_1 - \frac{u_1^3}{3} + u_2) \qquad t \in [0, 40]$$

$$\frac{du_2}{dt} = -\frac{1}{c}(u_1 - a + bu_2) \qquad \boldsymbol{u}(0) = (-1, 1)^\intercal$$

Now we use **legacy data** to pre-train the emulator and solve faster!

FitzHugh-Nagumo model

$$\frac{du_1}{dt} = c\left(u_1 - \frac{u_1^3}{3} + u_2\right) \qquad t \in [0, 40]$$

$$\frac{du_2}{dt} = -\frac{1}{c}(u_1 - a + bu_2) \qquad \boldsymbol{u}(0) = (-1, 1)^\intercal$$

Now we use **legacy data** to pre-train the emulator and solve faster!

Step 1: Run GParareal with the **original initial condition**.

$\boldsymbol{u}(0) = (-1, 1)^\intercal$

Step 2: Store the **F** and **G** solution data (= legacy data).

Step 3: Pre-train emulator using legacy data and then solve for **new initial value**.

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

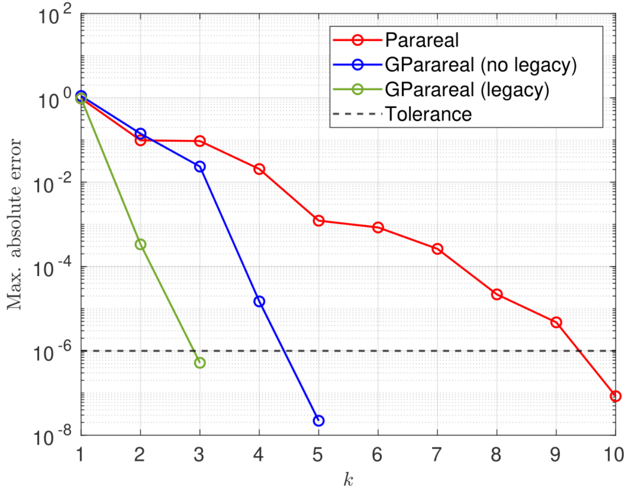
FitzHugh-Nagumo model

$$\frac{du_1}{dt} = c(u_1 - \frac{u_1^3}{3} + u_2) \qquad t \in [0, 40]$$

$$\frac{du_2}{dt} = -\frac{1}{c}(u_1 - a + bu_2) \qquad \boldsymbol{u}(0) = (-1, 1)^\intercal$$

Now we use **legacy data** to pre-train the emulator and solve faster!

Iterations until convergence k (with legacy data)



Step 1: Run GParareal with the **original initial condition**.

$$\boldsymbol{u}(0) = (-1, 1)^\intercal$$

Step 2: Store the **F** and **G** solution data (= legacy data).

Step 3: Pre-train emulator using legacy data and then solve for **new initial value**.

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

(V) Conclusions and future work

Advantages

- can converge in **fewer iterations than parareal** $\rightarrow$ **faster wallclock time**.
- solutions **maintain accuracy wrt parareal**, even for **chaotic systems**.
- GP can be **pre-trained using legacy solution data** $\rightarrow$ **improves speedup further**.
- can solve problems that parareal cannot (i.e. where it does not converge).

Advantages

- can converge in **fewer iterations than parareal → faster wallclock time**.
- solutions **maintain accuracy wrt parareal**, even for **chaotic systems**.
- GP can be **pre-trained using legacy solution data → improves speedup further**.
- can solve problems that parareal cannot (i.e. where it does not converge).

Drawbacks/open problems

- Are standard **out-the-box GP emulators** enough?

  → can we use **better ML/PN methods** to learn the (**high-dimensional**) correction?
- **Approximating the correction** by the **expected value of the GP** ignores all uncertainty.

  → currently we obtain point estimate solutions.

  → can we **quantify uncertainty** to develop a truly **PN method**?
- Is it worth developing developing a **time-parallel PN method?**

Thank you for listening! Questions?

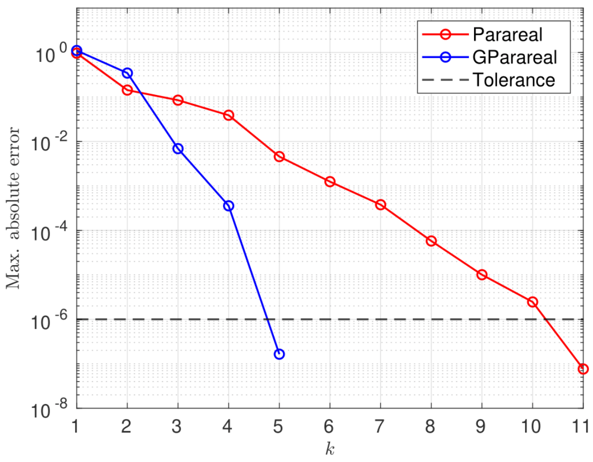
(IV) GParareal: additional results

FitzHugh-Nagumo model

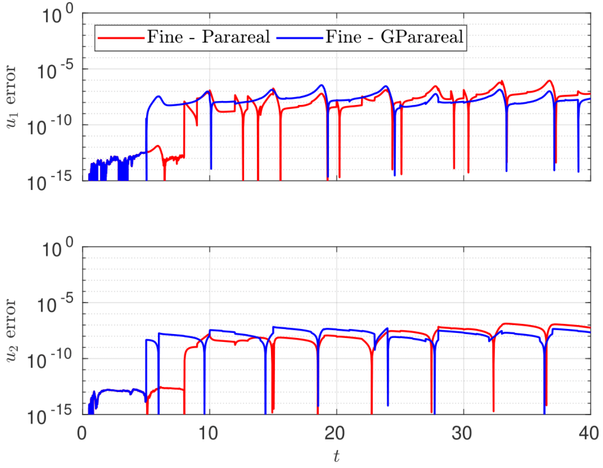
$$\frac{du_1}{dt} = c\left(u_1 - \frac{u_1^3}{3} + u_2\right) \qquad t \in [0, 40]$$

$$\frac{du_2}{dt} = -\frac{1}{c}(u_1 - a + bu_2) \qquad \boldsymbol{u}(0) = (-1, 1)^\intercal$$

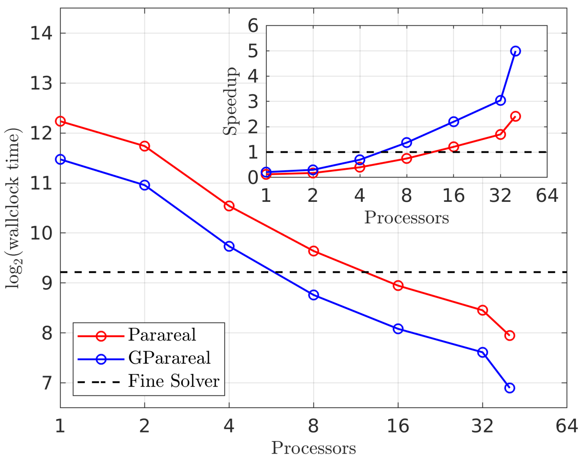
Iterations until convergence k



Accuracy vs. time



Speedup



(IV) GParareal: additional results

FitzHugh-Nagumo model

$$\frac{du_1}{dt} = c\left(u_1 - \frac{u_1^3}{3} + u_2\right) \qquad t \in [0, 40]$$

$$\frac{du_2}{dt} = -\frac{1}{c}(u_1 - a + bu_2) \qquad \boldsymbol{u}(0) = (-1, 1)^\intercal$$

Again we use **legacy data**, but solve over various initial values

Iterations until convergence k for various initial values

$\boldsymbol{u}(0) \in [-1.25, 1.25]^2$

GParareal + legacy data

$u_2(t=0)$ \ $u_1(t=0)$	-1.25	-1	-0.75	-0.5	-0.25	0	0.25	0.5	0.75	1	1.25
1.25	0	-2	-1	0	0	0	-1	-1	0	0	0
1	0	-3	0	0	0	0	0	-2	-1	-1	-1
0.75	0	0	0	0	-1	-2	-1	-1	0	-1	0
0.5	-1	-1	0	0	-1	0	0	0	-1	0	0
0.25	0	0	0	0	-1	0	-1	-1	-2	-1	-1
0	-1	-1	-2	-1	0	-1	-1	0	0	-1	0
-0.25	0	-1	0	0	0	-1	0	0	-1	-1	-1
-0.5	0	0	0	0	-1	-1	-1	0	0	0	0
-0.75	-1	-1	-1	-2	-1	-1	0	0	-1	0	0
-1	0	0	0	0	-1	0	0	0	0	-1	-1
-1.25	0	0	0	0	0	0	0	-1	-1	-1	-1

(IV) GParareal: additional results

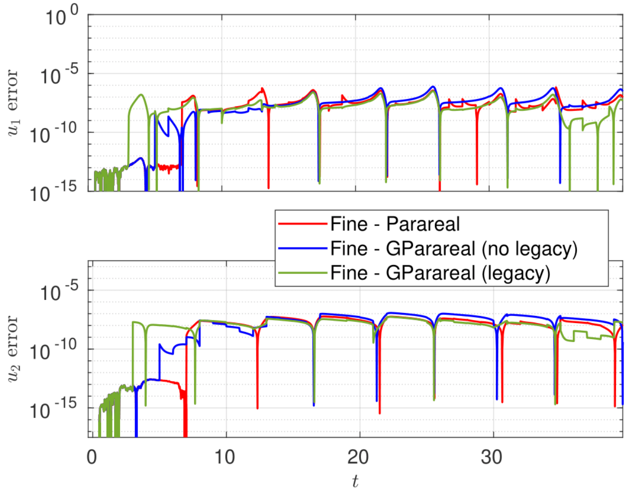
FitzHugh-Nagumo model

$$\frac{du_1}{dt} = c\left(u_1 - \frac{u_1^3}{3} + u_2\right) \qquad t \in [0, 40]$$

$$\frac{du_2}{dt} = -\frac{1}{c}(u_1 - a + bu_2) \qquad \boldsymbol{u}(0) = (-1, 1)^\intercal$$

Now we use **legacy data** to solve for a **new initial condition** faster!

Accuracy vs. time (with legacy data)



Step 1: Run GParareal with on the **original initial condition**.

$$\boldsymbol{u}(0) = (-1, 1)^\intercal$$

Step 2: Store the **F** and **G** solution data (= legacy data).

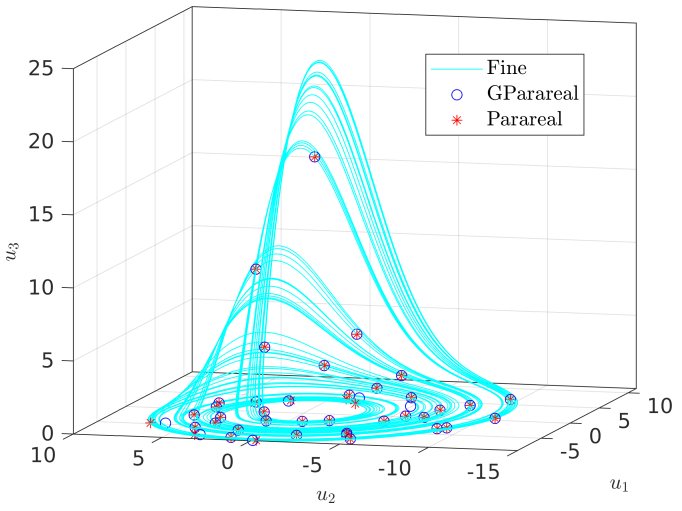
Step 3: Pre-train emulator using legacy data and then solve for **new initial value**.

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

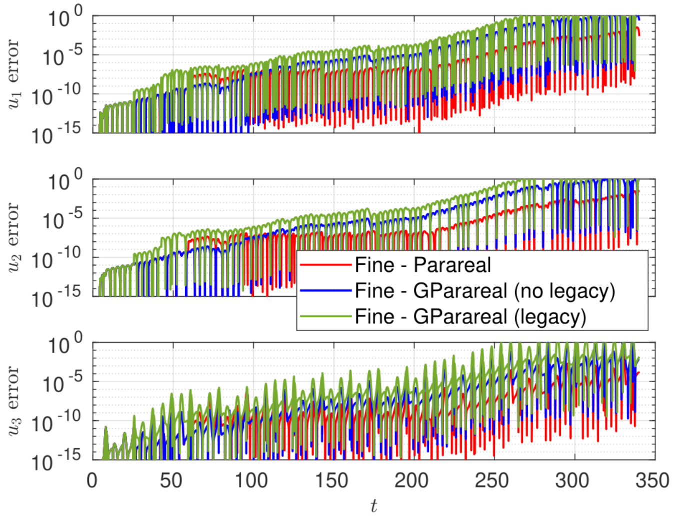
Rossler system

$$\frac{du_1}{dt} = -u_2 - u_3 \qquad \frac{du_2}{dt} = u_1 + au_2 \qquad \frac{du_3}{dt} = b + u_3(u_1 - c) \qquad t \in [0, 340] \qquad \boldsymbol{u}(0) = (0, -6.78, 0.02)^\intercal$$

Solutions in phase space



Accuracy vs. time

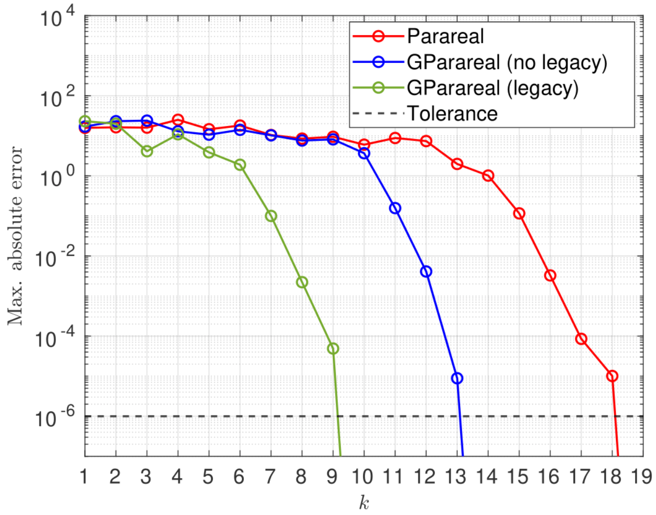


(IV) GParareal: additional results

Rossler system

$$\frac{du_1}{dt} = -u_2 - u_3 \qquad \frac{du_2}{dt} = u_1 + au_2 \qquad \frac{du_3}{dt} = b + u_3(u_1 - c) \qquad t \in [0, 340] \qquad \boldsymbol{u}(0) = (0, -6.78, 0.02)^\intercal$$

Iterations until convergence k (with legacy data)



Speedup (with legacy data)