Stochastic parareal: a novel application of probabilistic methods to time-parallelisation

Kamran Pentland[1], Massimiliano Tamborrino[2], Debasmita Samaddar[3], and Lynton Appel[3]

[1]Mathematics Institute, University of Warwick

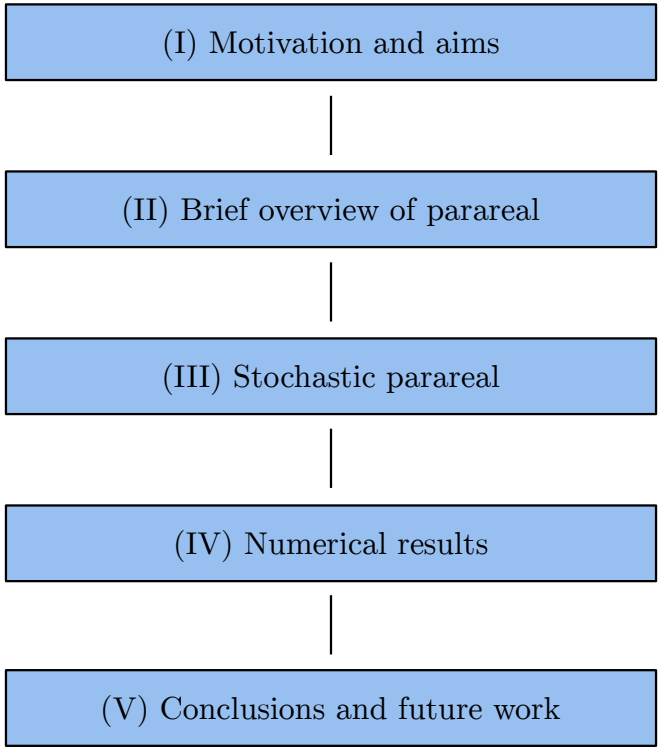
[2]Department of Statistics, University of Warwick

[3]Culham Centre for Fusion Energy, UKAEA

CCFE Seminar - 30/03/21

WARWICK

Structure of the talk



(I) Motivation and aims

- Want to numerically integrate large-scale **initial value problems** (IVPs).
  - → Systems of ordinary/partial differential equations.
  - → High **wallclock runtimes,** even using existing parallel methods.

- To extract further parallel speed-up, we must consider **parallelising in time.**
  - → Counter-intuitive (i.e. future solutions states depend upon previous states).
  - → Number of existing methods (multiple shooting type, multigrid etc).
  - → All of which are **deterministic** (i.e. provide fixed speed-up).

(I) Motivation and aims

- **Parareal**[1] is a one such time-parallel algorithm.
  - → Works on range of IVPs (i.e. molecular, fluid dynamics).
  - → Uses **two integrators** + **predictor-corrector** to converge iteratively.
  - → Locates solution in **fixed number of iterations**, yielding **fixed parallel speed-up**.
  - → Provides speed-up of approx. x10 vs. serial solvers – **including plasma dynamics**[2].

- Many variants of parareal exist to reduce wallclock time:
  - → Using adaptive time-stepping.
  - → Processor scheduling.

- None, however, **reduce the number of iterations** taken by parareal (**fewer iterations = larger speed-up**).

[1] - J. L. Lions, Y. Maday, and G. Turinici, ***Résolution d'EDP par un schéma en temps pararéel***, C. R. Math. Acad. Sci. Paris - Series I: Math., 332 (2001), pp. 661–668.
[2] - D. Samaddar, D. E. Newman, and R. Sánchez, ***Parallelization in time of numerical simulations of fully-developed plasma turbulence using the parareal algorithm***, J. Comput. Phys., 229 (2010), pp. 6558–6573.

(I) Motivation and aims

- Aim to extend parareal $\rightarrow$ **stochastic parareal algorithm.**
  - Sample **multiple** potential solutions from **probability distributions.**
  - "Most accurate" fed into parareal's **predictor-corrector.**
  - Converges in **fewer iterations than the original algorithm**, with given probability.

- Illustrate power of method by running **numerical simulations** on IVPs:
  - estimating **discrete distributions** of the **convergence rate** for stochastic parareal.
  - testing different **sampling rules** (i.e. different probability distributions).
  - validating the **accuracy** of stochastic solutions vs. the deterministic solution.

Focus is on **convergence rates**, not wallclock runtimes.

Take home message

Parareal converges **deterministically** (**parallel speed-up is fixed**) $\rightarrow$ stochastic sampling used to **reduce the number of iterations**.

(II) Overview of parareal

The idea

- Solve an IVP **in parallel**: $\frac{du}{dt} = f(u(t), t) \qquad t \in [T_0, T_N] \qquad u(T_0) = u^0$ ← Initial value

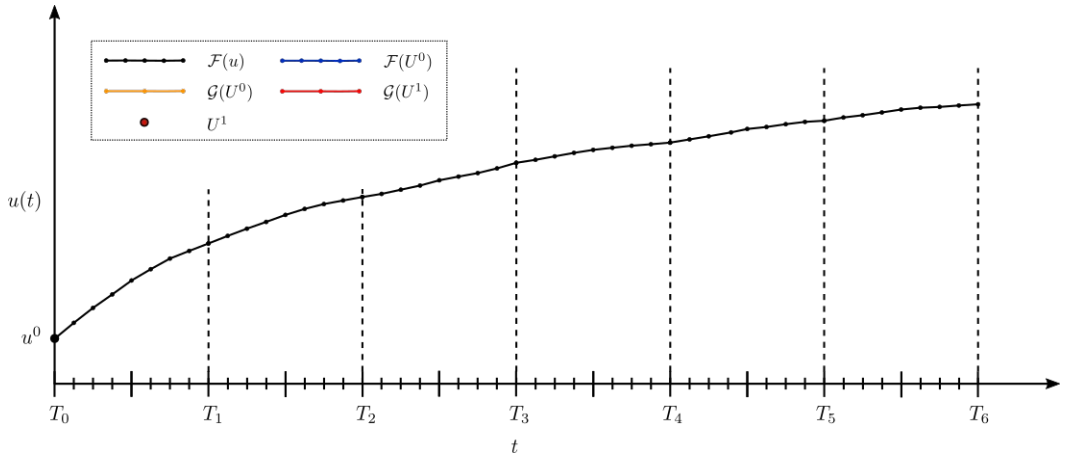
- Parareal discretises into N sub-problems $\rightarrow$ each assigned a processor.

- Integration carried out by:
  - **High accuracy, slow integrator F** $\rightarrow$ runs in parallel.
  - **Lower accuracy, fast integrator G** $\rightarrow$ runs serially.

- N initial values **required** to run F in parallel $\rightarrow$ parareal **combines solvers** to locate solution.

How it works

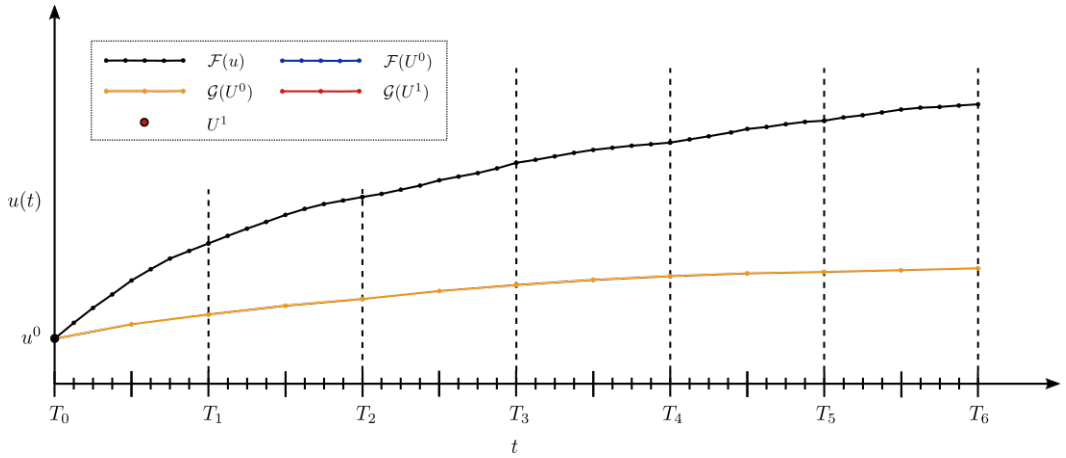
Aiming for F solution (black line).



How it works

Aiming for F solution (black line).

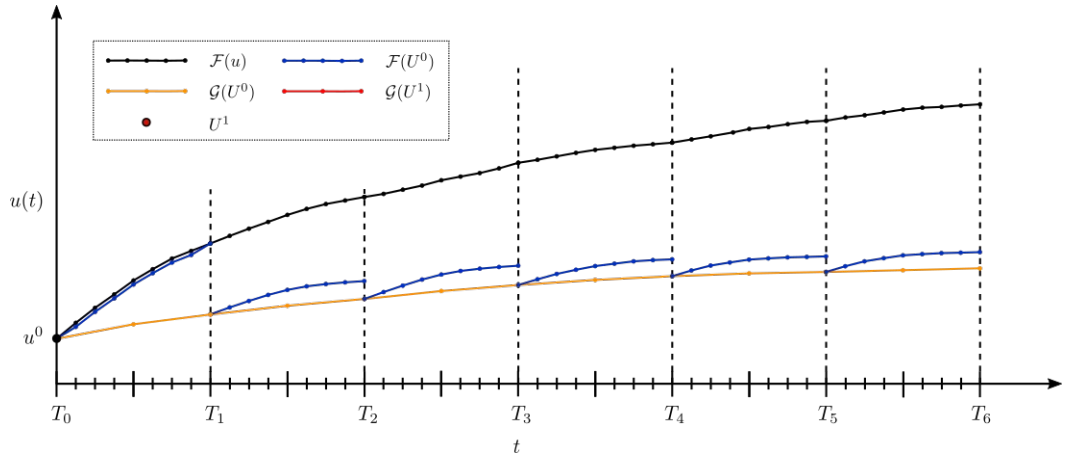
1) Run G **serially** over entire interval (yellow).



How it works

Aiming for F solution (black line).

1) Run G **serially** over entire interval (yellow).
2) Run F **in parallel** to locate more accurate solutions (blue).



How it works

Aiming for F solution (black line).

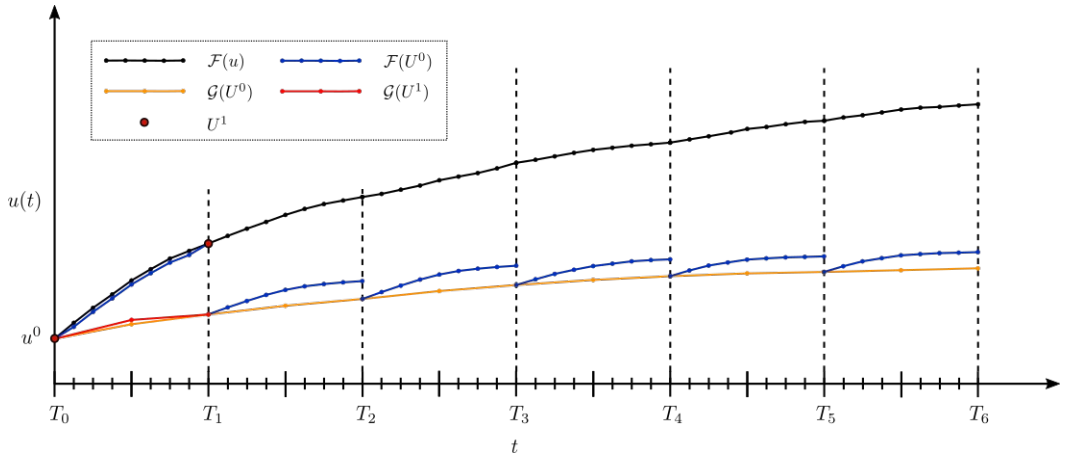
1) Run G **serially** over entire interval (yellow).
2) Run F **in parallel** to locate more accurate solutions (blue).
3) Run G (red) and use **predictor-corrector (PC)** to iteratively improve solution in each sub-interval.

Solution

Iteration number

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

Time step



How it works

Aiming for F solution (black line).

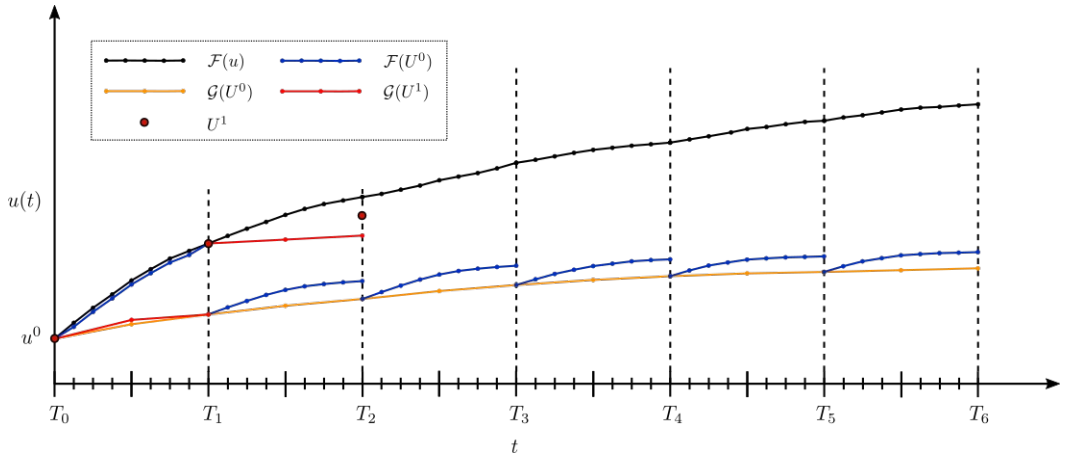
1) Run G **serially** over entire interval (yellow).
2) Run F **in parallel** to locate more accurate solutions (blue).
3) Run G (red) and use **predictor-corrector (PC)** to iteratively improve solution in each sub-interval.

Solution

Iteration number

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

Time step



(II) Overview of parareal

How it works

Aiming for F solution (black line).

1) Run G **serially** over entire interval (yellow).
2) Run F **in parallel** to locate more accurate solutions (blue).
3) Run G (red) and use **predictor-corrector (PC)** to iteratively improve solution in each sub-interval.

Solution

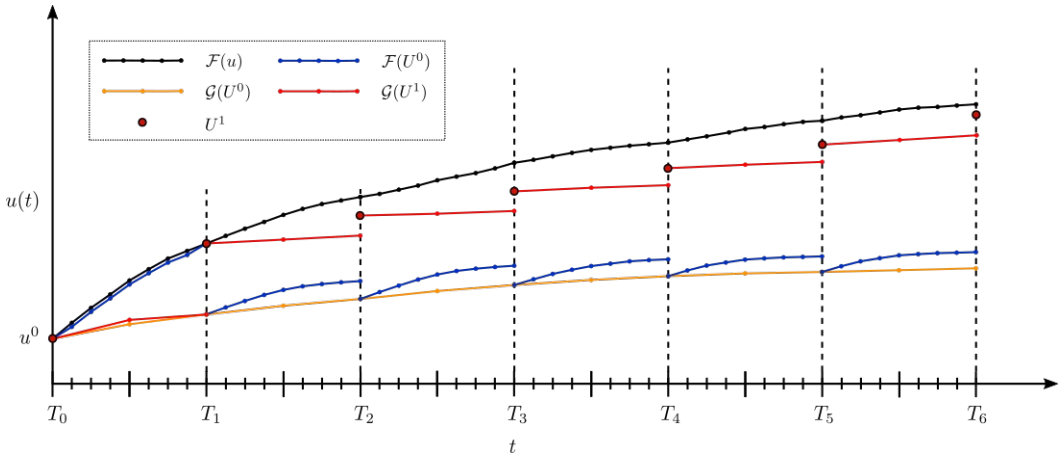
Iteration number

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

Time step

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

$$\|U_i^k - U_i^{k-1}\|_\infty < \varepsilon \ \forall i \in \{1, \ldots, N\}.$$



(II) Overview of parareal

How it works

Converges **deterministically** in $k_d \in \{1, \ldots, N\}$ iterations.

Parallel speed-up $\approx N/k_d$

Aiming for F solution (black line).

1) Run G **serially** over entire interval (yellow).
2) Run F **in parallel** to locate more accurate solutions (blue).
3) Run G (red) and use **predictor-corrector (PC)** to iteratively improve solution in each sub-interval.

Solution

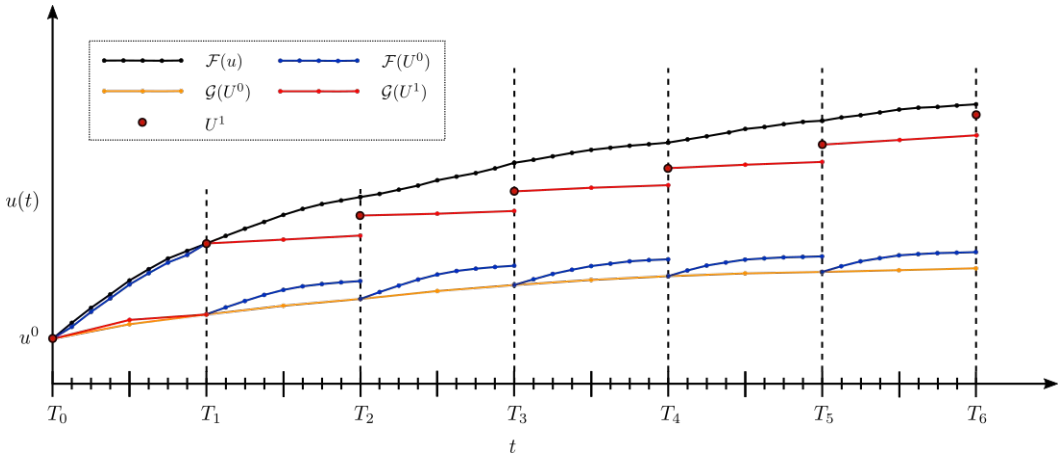
Iteration number

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

Time step

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

$$\|U_i^k - U_i^{k-1}\|_\infty < \varepsilon \ \forall i \in \{1, \ldots, N\}.$$



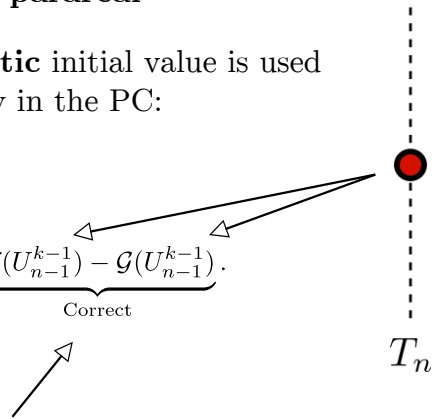
(III) Stochastic parareal

Parareal is already fairly robust...how do we improve its convergence rate?

Classic parareal

A single **deterministic** initial value is used iteratively in the PC:

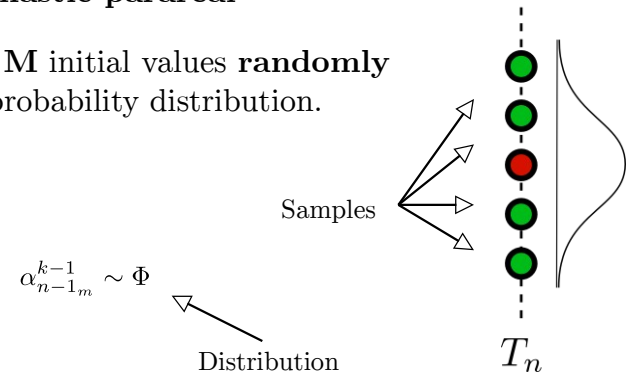
$$U_n^k = \underbrace{\mathcal{G}(U_{n-1}^k)}_{\text{Predict}} + \underbrace{\mathcal{F}(U_{n-1}^{k-1}) - \mathcal{G}(U_{n-1}^{k-1})}_{\text{Correct}}.$$



Want to improve this correction.

Stochastic parareal

Instead, sample **M** initial values **randomly** from some probability distribution.

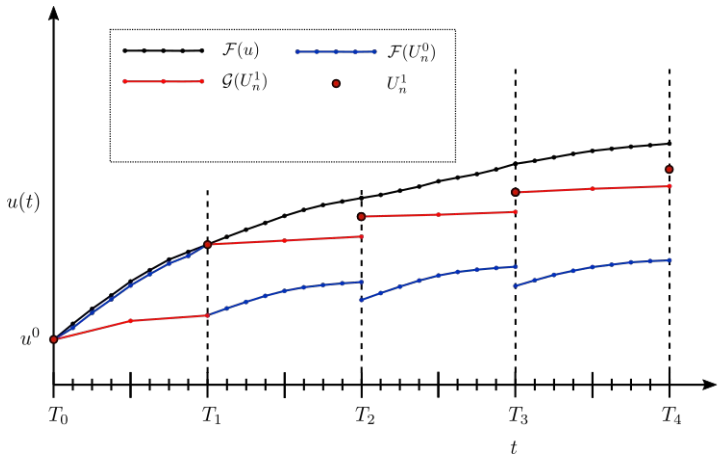


Choose 'optimal sample' from these.

Aim: Improve correction using **optimal sample** → **converge in fewer iterations.**

How it works

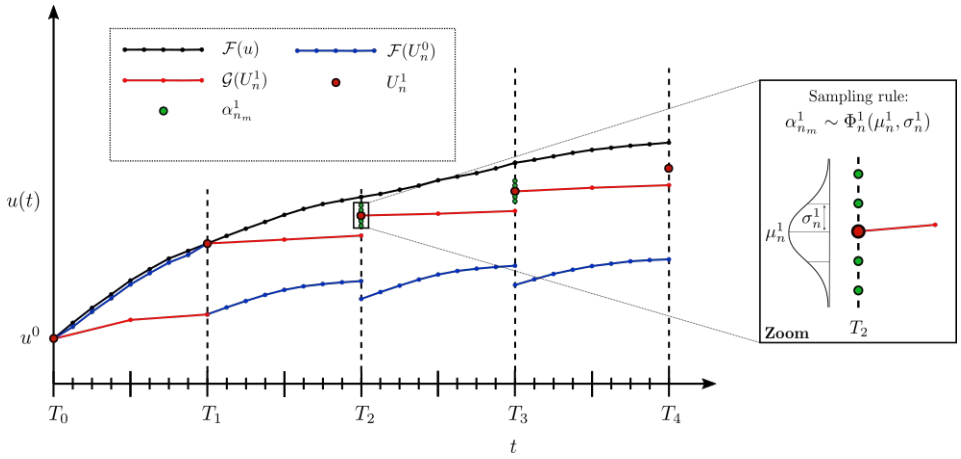
1) First iteration ($k = 1$) identical to parareal. Start with the PC values (red dots).



(III) Stochastic parareal

How it works

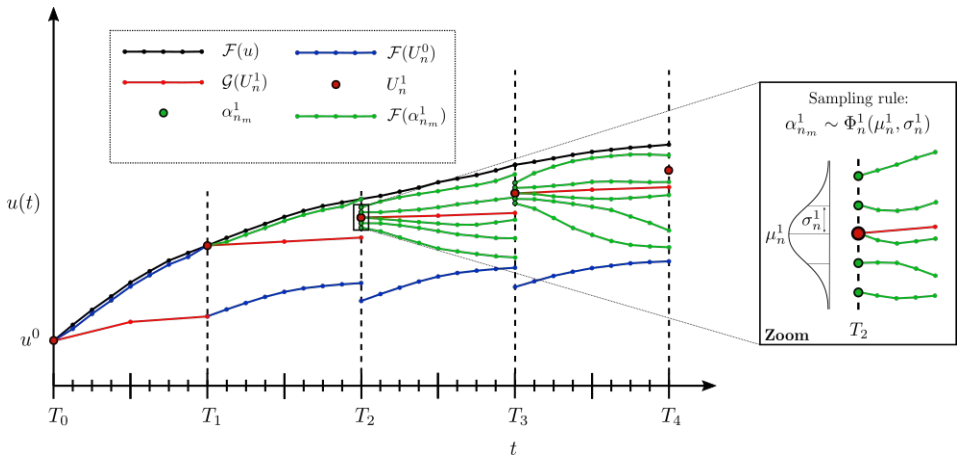
1) First iteration ($k = 1$) identical to parareal. Start with the PC values (red dots).
2) Sample M initial values at unconverged sub-intervals: $\alpha^1_{n_m} \sim \Phi(\mu^1_n, \sigma^1_n)$



(III) Stochastic parareal

How it works

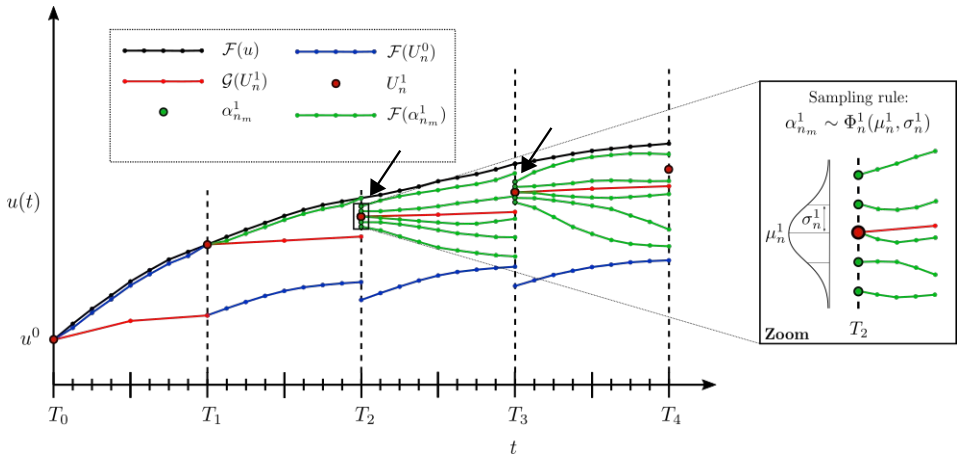
1) First iteration ($k = 1$) identical to parareal. Start with the PC values (red dots).
2) Sample M initial values at unconverged sub-intervals: $\alpha^1_{n_m} \sim \Phi(\mu^1_n, \sigma^1_n)$
3) Run F **in parallel** on each sample (green lines).



(III) Stochastic parareal

How it works

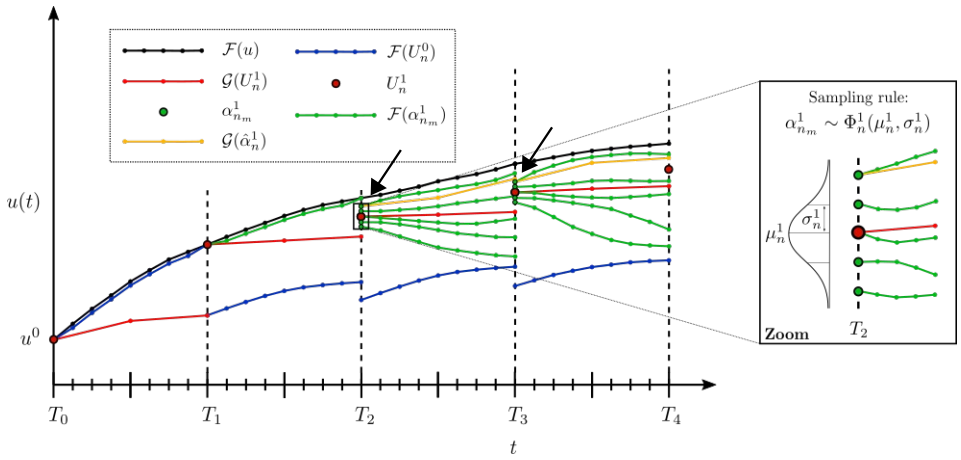
1) First iteration ($k = 1$) identical to parareal. Start with the PC values (red dots).
2) Sample M initial values at unconverged sub-intervals: $\alpha^1_{n_m} \sim \Phi(\mu^1_n, \sigma^1_n)$
3) Run F **in parallel** on each sample (green lines).
4) Select **'optimal sample'** $\hat{\alpha}^1_n$ at each sub-interval by identifying the smoothest F trajectory over the entire time interval.



(III) Stochastic parareal

How it works

1) First iteration ($k = 1$) identical to parareal. Start with the PC values (red dots).
2) Sample M initial values at unconverged sub-intervals: $\alpha^1_{n_m} \sim \Phi(\mu^1_n, \sigma^1_n)$
3) Run F **in parallel** on each sample (green lines).
4) Select **'optimal sample'** $\hat{\alpha}^1_n$ at each sub-interval by identifying the smoothest F trajectory over the entire time interval.
5) Run G on optimal samples (yellow lines).

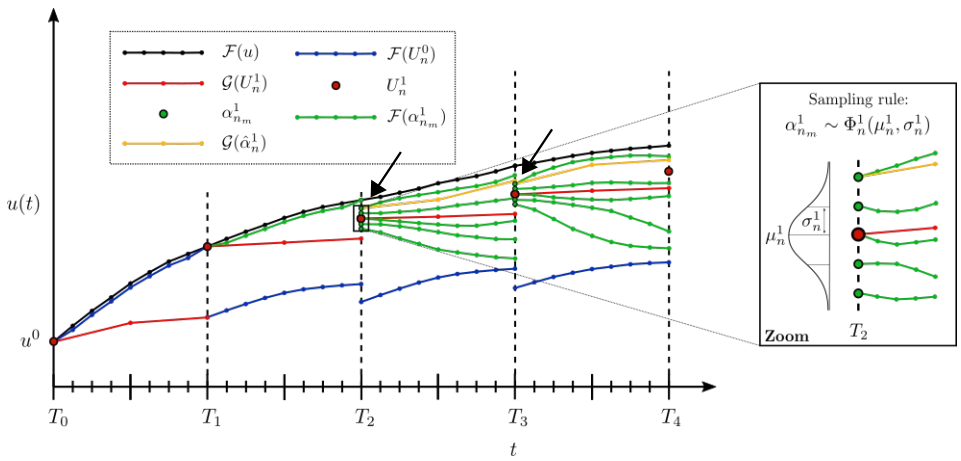


(III) Stochastic parareal

How it works

1) First iteration ($k = 1$) identical to parareal. Start with the PC values (red dots).
2) Sample M initial values at unconverged sub-intervals: $\alpha^1_{n_m} \sim \Phi(\mu^1_n, \sigma^1_n)$
3) Run F **in parallel** on each sample (green lines).
4) Select **'optimal sample'** $\hat{\alpha}^1_n$ at each sub-interval by identifying the smoothest F trajectory over the entire time interval.
5) Run G on optimal samples (yellow lines).
6) Use new F and G values in PC:

$$U^2_n = \underbrace{\mathcal{G}(U^2_{n-1})}_{\text{predict}} + \underbrace{\mathcal{F}(\hat{\alpha}^1_{n-1}) - \mathcal{G}(\hat{\alpha}^1_{n-1})}_{\text{new correction}}.$$



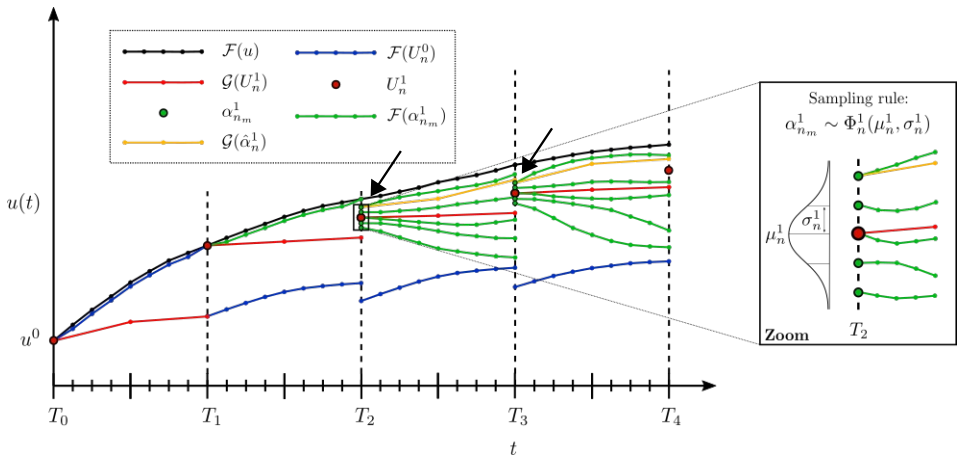
(III) Stochastic parareal

How it works

1) First iteration ($k = 1$) identical to parareal. Start with the PC values (red dots).
2) Sample M initial values at unconverged sub-intervals: $\alpha^1_{n_m} \sim \Phi(\mu^1_n, \sigma^1_n)$
3) Run F **in parallel** on each sample (green lines).
4) Select **'optimal sample'** $\hat{\alpha}^1_n$ at each sub-interval by identifying the smoothest F trajectory over the entire time interval.
5) Run G on optimal samples (yellow lines).
6) Use new F and G values in PC:

$$U^2_n = \underbrace{\mathcal{G}(U^2_{n-1})}_{\text{predict}} + \underbrace{\mathcal{F}(\hat{\alpha}^1_{n-1}) - \mathcal{G}(\hat{\alpha}^1_{n-1})}_{\text{new correction}}.$$

7) Carry out error checks – then do further iterations from step 2 if necessary.



(III) Stochastic parareal

How it works

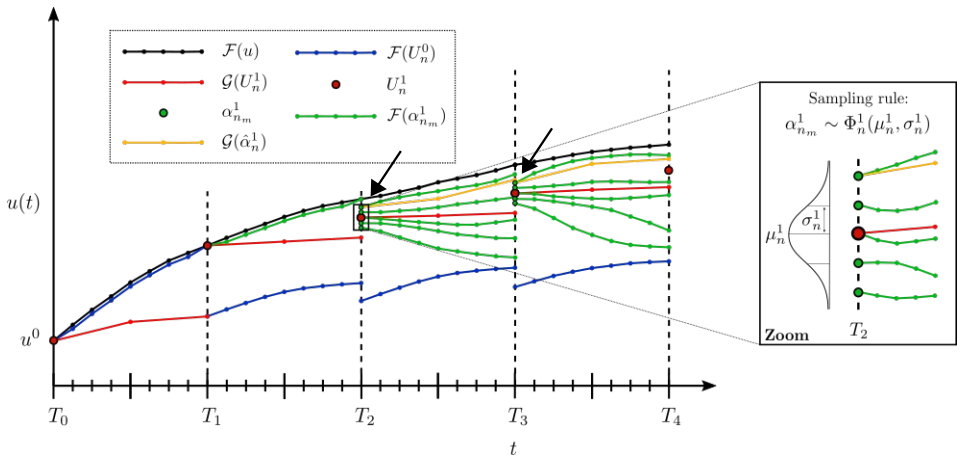
1) First iteration ($k = 1$) identical to parareal. Start with the PC values (red dots).
2) Sample M initial values at unconverged sub-intervals: $\alpha^1_{n_m} \sim \Phi(\mu^1_n, \sigma^1_n)$
3) Run F **in parallel** on each sample (green lines).
4) Select **'optimal sample'** $\hat{\alpha}^1_n$ at each sub-interval by identifying the smoothest F trajectory over the entire time interval.
5) Run G on optimal samples (yellow lines).
6) Use new F and G values in PC:

$$U^2_n = \underbrace{\mathcal{G}(U^2_{n-1})}_{\text{predict}} + \underbrace{\mathcal{F}(\hat{\alpha}^1_{n-1}) - \mathcal{G}(\hat{\alpha}^1_{n-1})}_{\text{new correction}}.$$

7) Carry out error checks – then do further iterations from step 2 if necessary.

Note: M = 1 → stochastic parareal = parareal.

Converges **stochastically** in $k_s \in \{1, \ldots, N\}$ iterations.



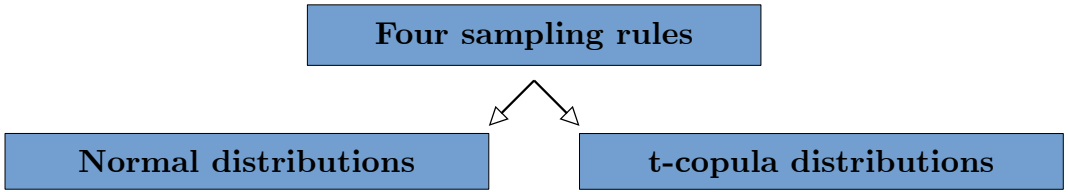
(III) Stochastic parareal

Properties

- Numerical solutions **and** convergence rate **vary stochastically** $\rightarrow$ i.e. probabilistic solutions.
- F is run more frequently $\rightarrow$ requires **M times more processors** than parareal.

Sampling rules

- Known F and G solutions used to **construct probability distributions**.
- Vary with iteration **and** time step.
- **Four rules** tested.
- **Two different means** $\rightarrow$ either F values or the PC values.
- **Standard deviation** fixed $\rightarrow$ difference between G values.
- **Correlations** $\rightarrow$ taken between components of system.

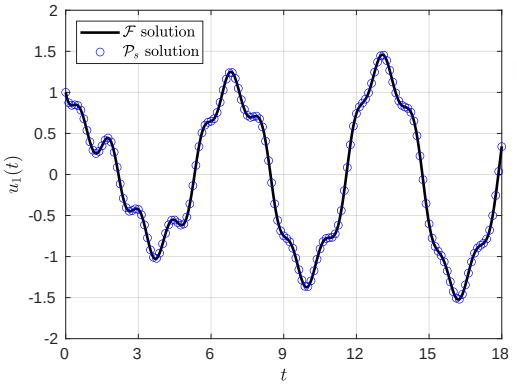


(IV) Numerical results

One-dimensional nonlinear ODE

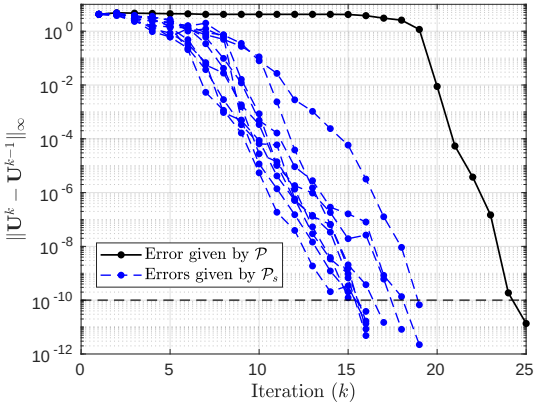
$$\frac{du_1}{dt} = \sin(u_1)\cos(u_1) - 2u_1 + e^{-t/100}\sin(5t) + \ln(1+t)\cos(t)$$

Initial value	Time interval
$u_1(0) = 1$	$t \in [0, 100]$



Numerical solutions in interval [0,18].

- F and G selected as explicit RK4 methods $\rightarrow$ G with much larger time steps.
- Parareal takes $k_d = 25$ iterations (out of 40).
- Taking M = 2 (sampling rule 1) $\rightarrow$ convergence is stochastic and much sooner than parareal!



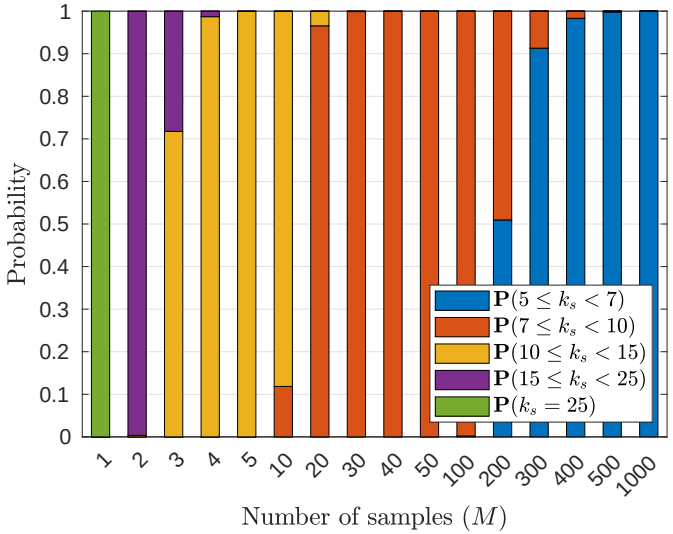
Successive errors generated by both algorithms at each iteration k.

(IV) Numerical results

One-dimensional nonlinear ODE

- Estimate **distributions** for $k_s \rightarrow$ ran algorithm 2000 times for each M.
- If $M > 1 \rightarrow$ we better the parareal result with **certainty**.
- As M increases $\rightarrow$ k_s decreases rapidly!
- Best scenario $k_s = 5 \rightarrow$ 20 iterations sooner than parareal!
- Demonstrates power of using stochastic sampling to reduce convergence rate.
- Other rules perform very similarly.

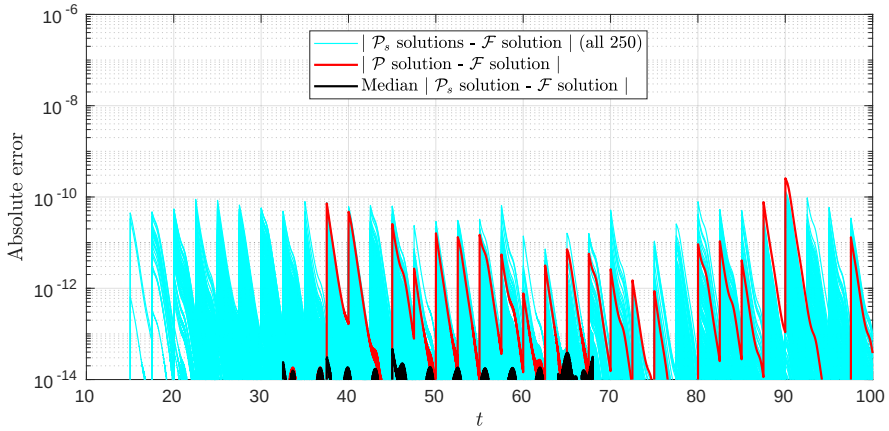
Sampling rule 1 (Normal distrib.)



Distributions of convergence rate k_s for increasing M.

One-dimensional nonlinear ODE

- **Median error** of stochastic solution is **smaller** than deterministic error.
- Standard deviation of stochastic solutions is $\mathcal{O}(10^{-11})$.



Absolute errors of: - 250 independent stochastic solutions vs. the F solver (light blue).
- median stochastic solution vs. F solution (black).
- parareal solution vs. F solution (red).

(IV) Numerical results

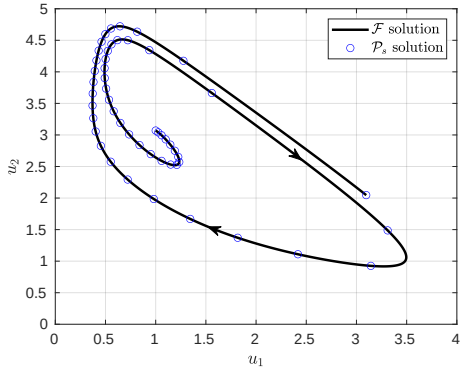
Two-dimensional Brusselator system

- Next, we test performance using **bivariate** probability distributions on **stiff** system.
- Parareal takes $k_d = 7$ iterations (out of 25).
- Sampling close to F values (**rules 1 and 3**) perform best.
- Generating **correlated** samples improves performance.
- Stiffness does however demand more samples to reduce k_s further and further.

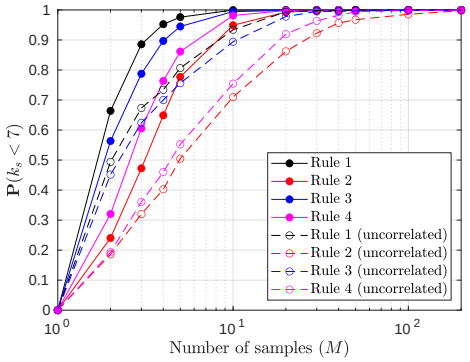
$$\frac{du_1}{dt} = 1 + u_1^2 u_2 - 4u_1$$
$$\frac{du_2}{dt} = 3u_1 - u_1^2 u_2$$

Initial values
$\boldsymbol{u}(0) = (1, 3.07)^{\mathrm{T}}$

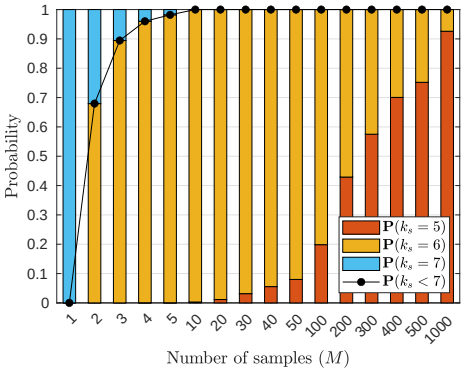
Time interval
$t \in [0, 15.3]$



Numerical solutions in phase space over time.



Estimated probabilities of converging in fewer iterations than deterministic parareal.



Estimated discrete distributions of k_s for sampling rule 1.

(IV) Numerical results

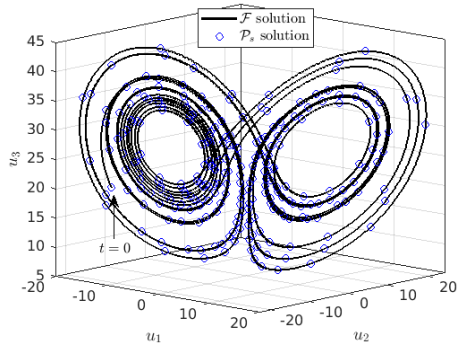
Three-dimensional Lorenz system

- Chaotic system $\rightarrow$ exponentially diverging trajectories.
- Parareal takes $k_d = 20$ iterations (out of 50).
- Sampling close to PC values (**rules 2 and 4**) perform best in this case.
- Generating correlated samples, again, improves performance.
- Best convergence of $k_s = 16$ in small fraction of runs with M = 1000.

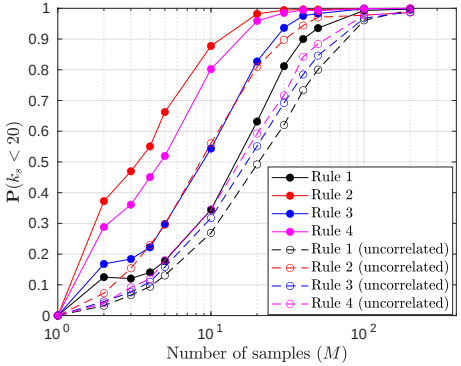
$$\frac{du_1}{dt} = 10(u_2 - u_1)$$
$$\frac{du_2}{dt} = 28u_1 - u_1u_3u_2$$
$$\frac{du_3}{dt} = u_1u_2 - \frac{8}{3}u_3$$

Initial values
$\boldsymbol{u}(0) = (-15, -15, 20)^{\mathrm{T}}$

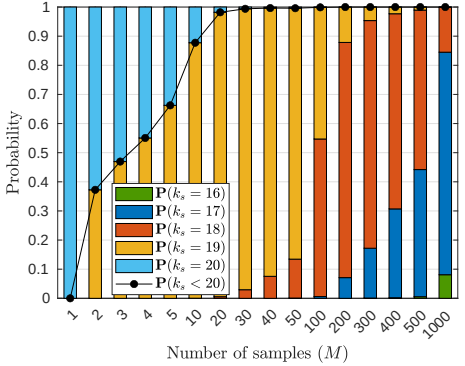
Time interval
$t \in [0, 18]$



Numerical solutions in phase space over time.



Estimated probabilities of converging in fewer iterations than deterministic parareal.



Estimated discrete distributions of k_s for sampling rule 2.

(V) Conclusions and future work

Conclusions

- Developed a **stochastic parareal algorithm** $\rightarrow$ can **better the convergence rate of parareal**.
- Sampling from probability distributions $\rightarrow$ **increases probability** of locating true solution.
- Results illustrate that:
  - Given sufficient samples (M) $\rightarrow$ probability of beating parareal **approaches one**.
  - **Expected value** of k_s decreases for increasing M.
  - **Accuracy** of stochastic solutions is maintained, often better, than parareal.
  - Choice of **marginal means** has more impact than **type of distribution** – however no clear "optimal" sampling rule.
  - Generating **correlated samples** improves performance.

(V) Conclusions and future work

Future work

- Does algorithm scale for much larger systems? $\rightarrow$ long term goal to apply these methods to plasma simulations.
- Number of processors required scales with M $\rightarrow$ problematic if large amount of sampling required.
- Develop sampling rules/alternative methods that adapt to the problem being solved.

Thank you for listening! Questions?