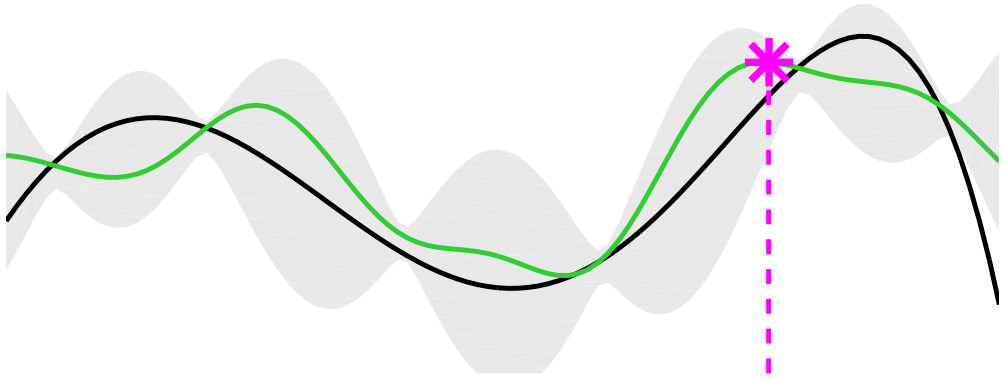
Parallelised Bayesian Optimisation via Thompson Sampling



Kirthevasan Kandasamy

Carnegie Mellon University

Google Research, Mountain View, CA

Sep 27, 2017

Slides: www.cs.cmu.edu/~kkandasa/talks/google-ts-slides.pdf

Slides are up on my website: www.cs.cmu.edu/∼kkandasa

www.cs.cmu.edu/~kkandasa/index.html

home research software misc.

kirthevasan kandasamy

PhD Student, Carnegie Mellon University

[CV] [Google Scholar] [GitHub] [Contact]

I am a fourth year Machine Learning PhD student (now ABD) in the School of Computer Science at Carnegie Mellon University. I am co-advised by Jeff Schneider and Barnabas Poczos. I am a member of the Auton Lab and the StatML Group. Prior to CMU, I completed my B.Sc in Electronics & Telecommunications Engineering at the University of Moratuwa, Sri Lanka.

My research interests lie in the intersection of statistical and algorithmic Machine Learning. My current research spans bandit problems, Bayesian optimisation, Gaussian processes, nonparametric statistics and graphical models. As of late, I have also hopped on the deep learning bandwagon. For more details, see my publications.

I am generously supported by a Facebook PhD fellowship (2017) and a CMU Presidential fellowship (2015).

Recent updates

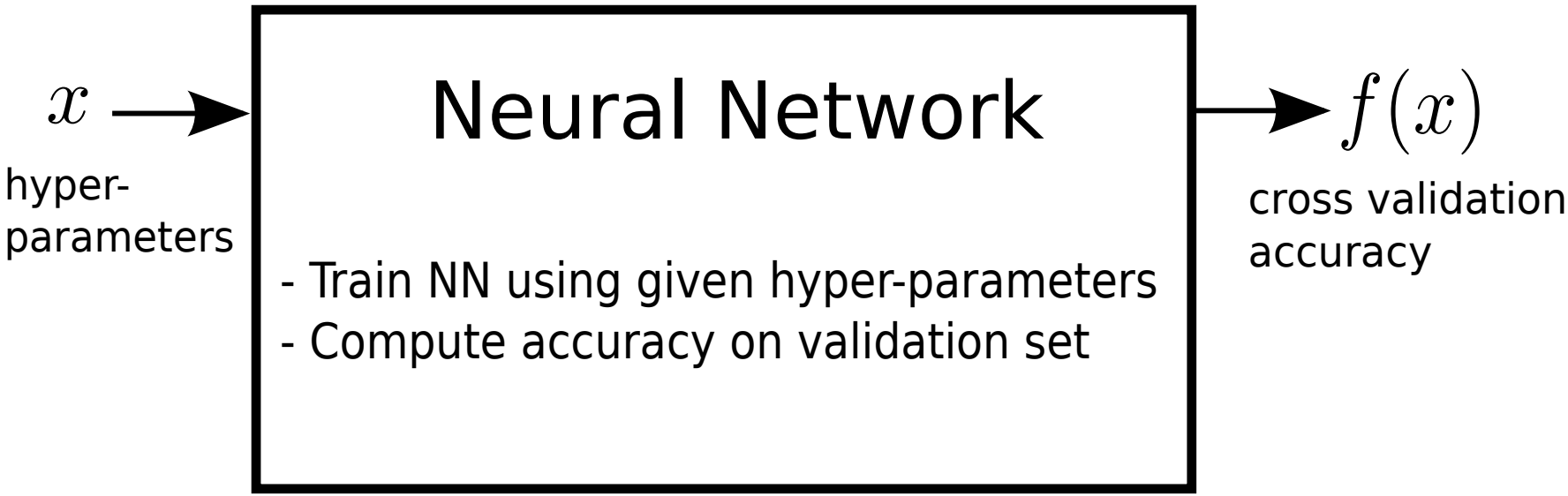
Sep 26: Talk at Facebook on Multi-fidelity Bayesian Optimisation [slides]
Sep 27: Talk at Google on Parallelised Thompson Sampling [slides]

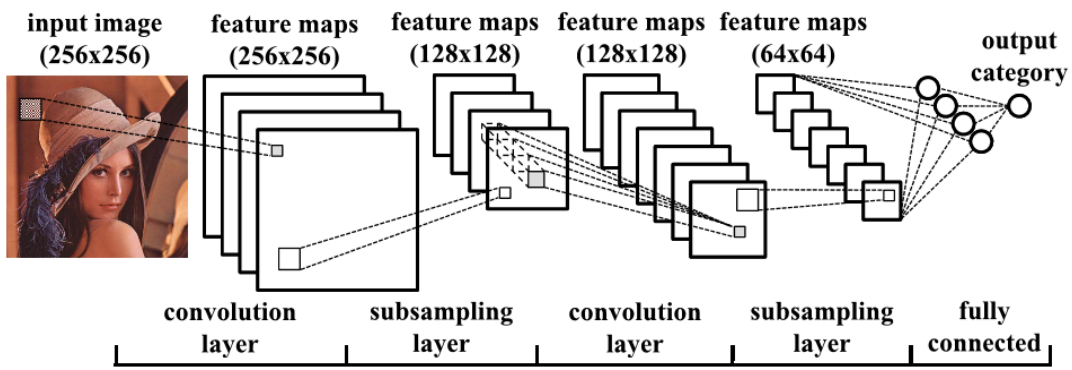
Slides

Contact

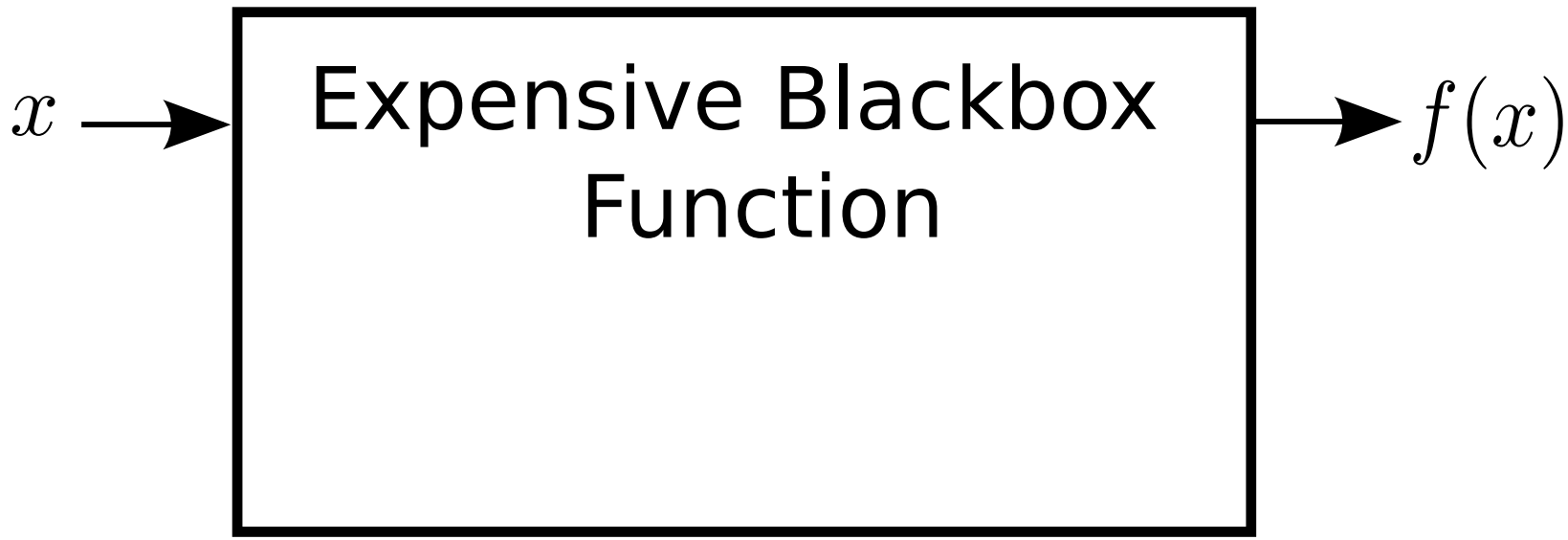
GHC 8213
Machine Learning Department
School of Computer Science
Carnegie Mellon University

Black-box Optimisation

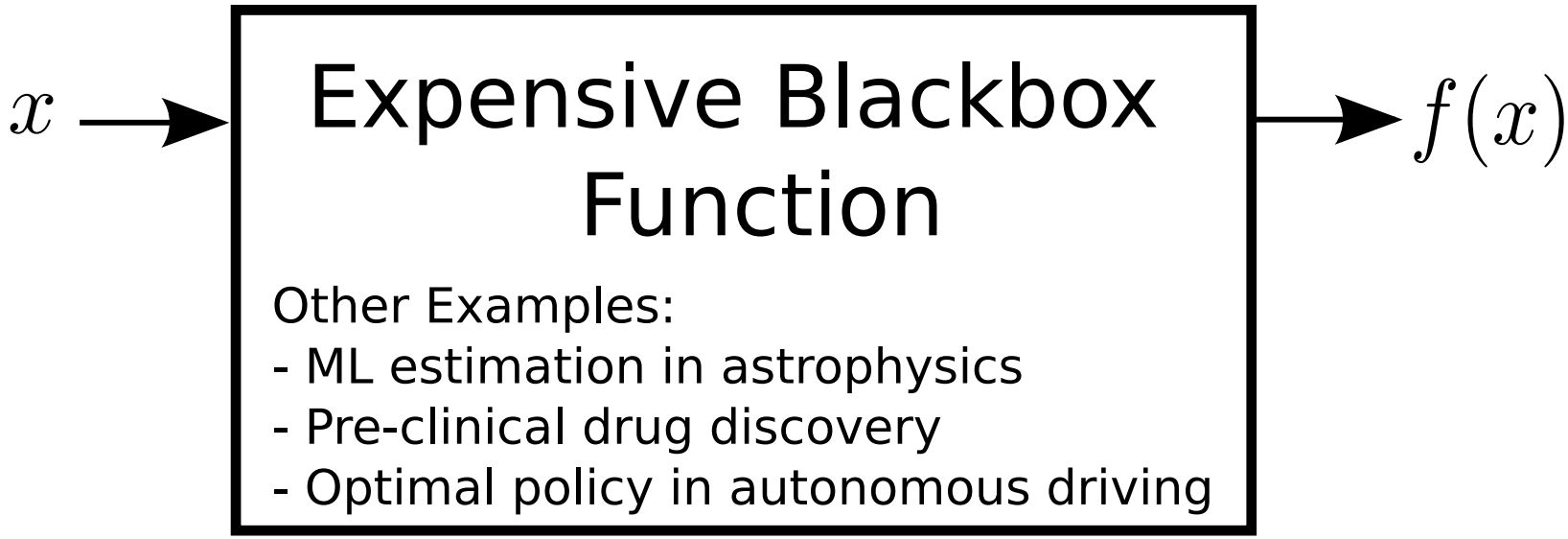




Black-box Optimisation

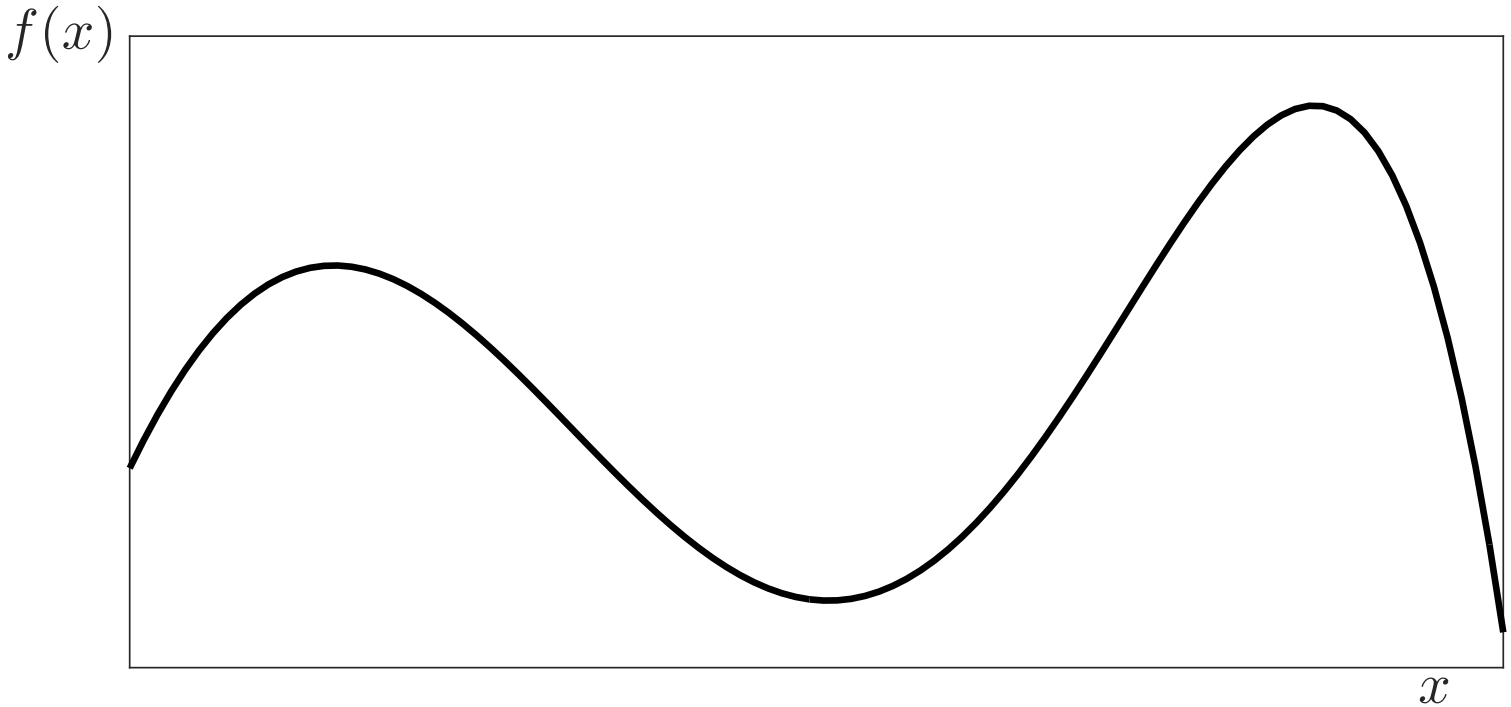


Black-box Optimisation



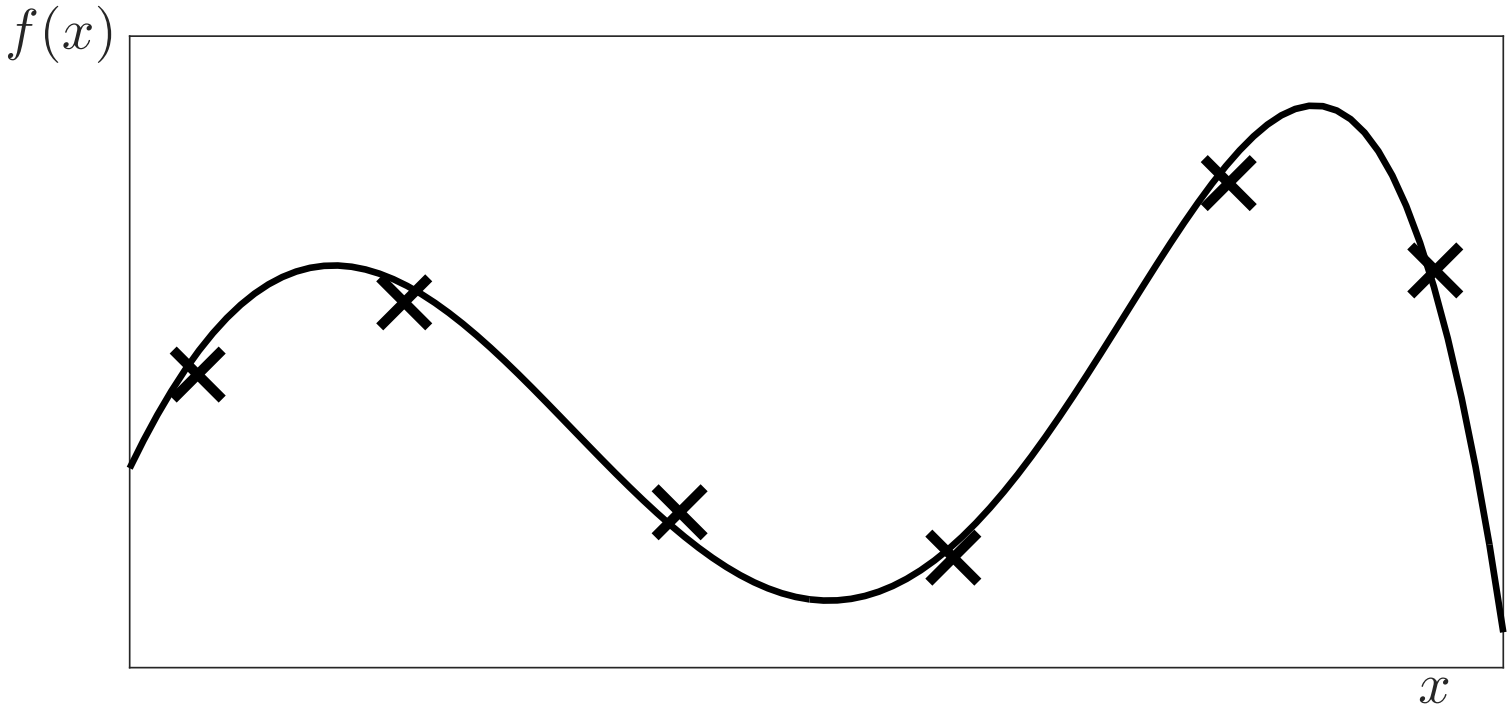
Black-box Optimisation

$f : \mathcal{X} \to \mathbb{R}$ is an expensive, black-box, noisy function, accessible only via noisy evaluations.



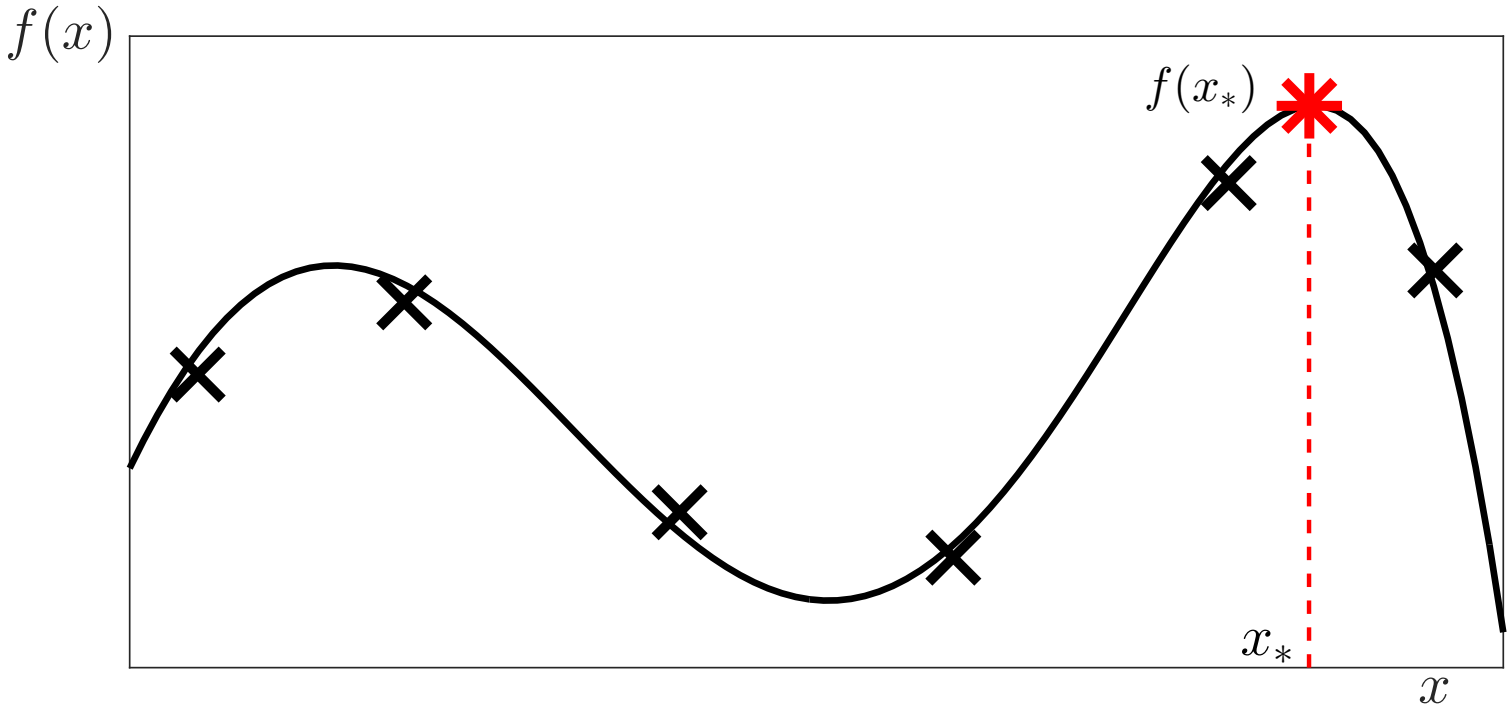
Black-box Optimisation

$f : \mathcal{X} \to \mathbb{R}$ is an expensive, black-box, noisy function, accessible only via noisy evaluations.



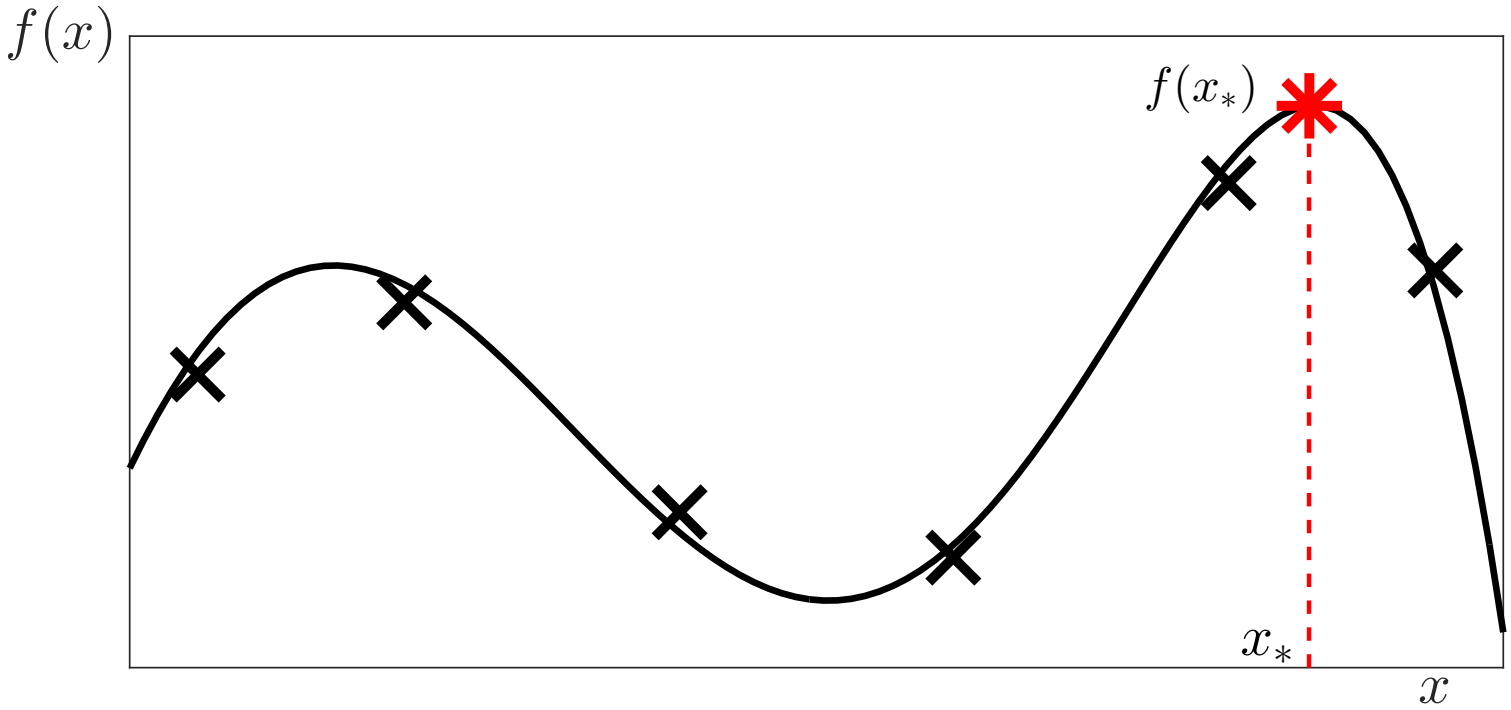
Black-box Optimisation

$f : \mathcal{X} \to \mathbb{R}$ is an expensive, black-box, noisy function, accessible only via noisy evaluations.
Let $x_\star = \mathrm{argmax}_x f(x)$.



Black-box Optimisation

$f : \mathcal{X} \to \mathbb{R}$ is an expensive, black-box, noisy function, accessible only via noisy evaluations.
Let $x_\star = \mathrm{argmax}_x f(x)$.

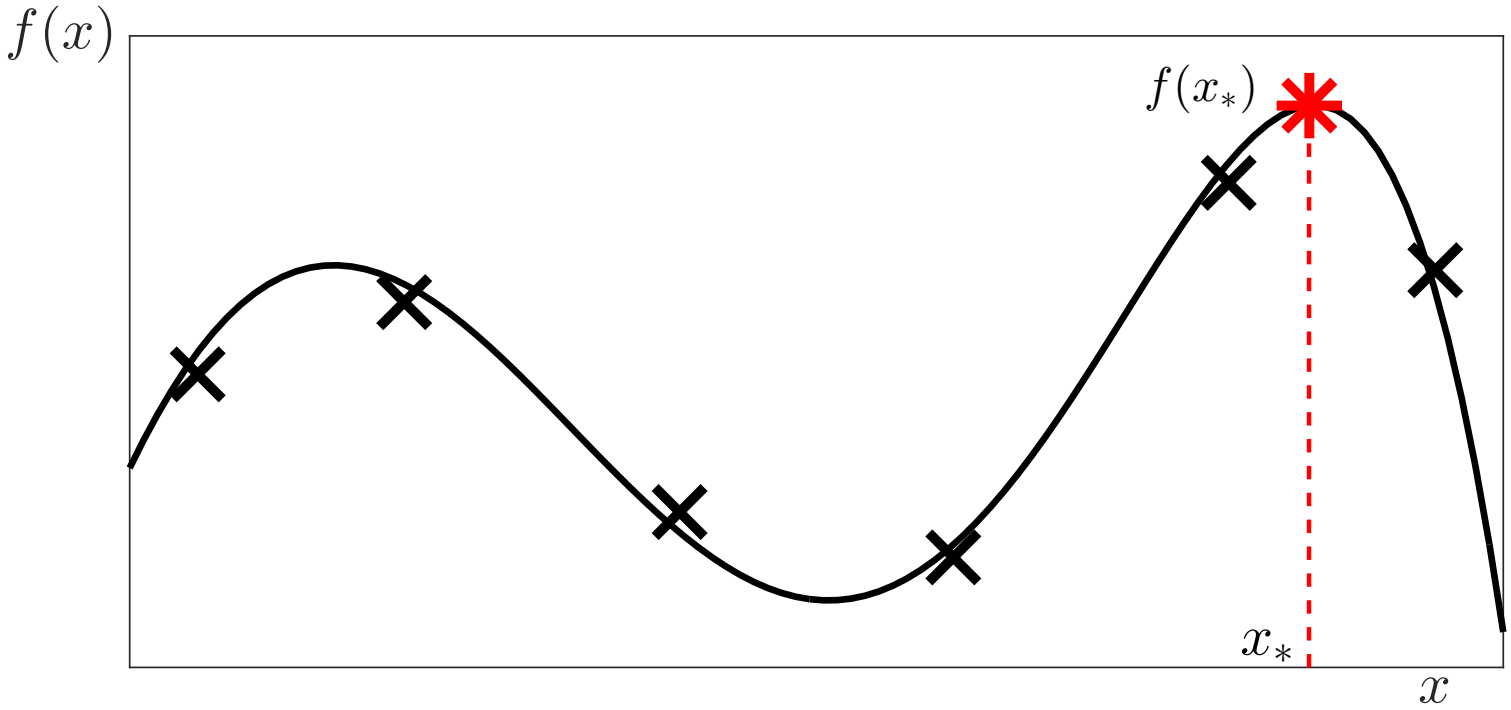


Simple Regret after n evaluations

$$\mathrm{SR}(n) = f(x_\star) - \max_{t=1,\ldots,n} f(x_t).$$

Black-box Optimisation

$f : \mathcal{X} \to \mathbb{R}$ is an expensive, black-box, noisy function, accessible only via noisy evaluations.
Let $x_\star = \mathrm{argmax}_x f(x)$.

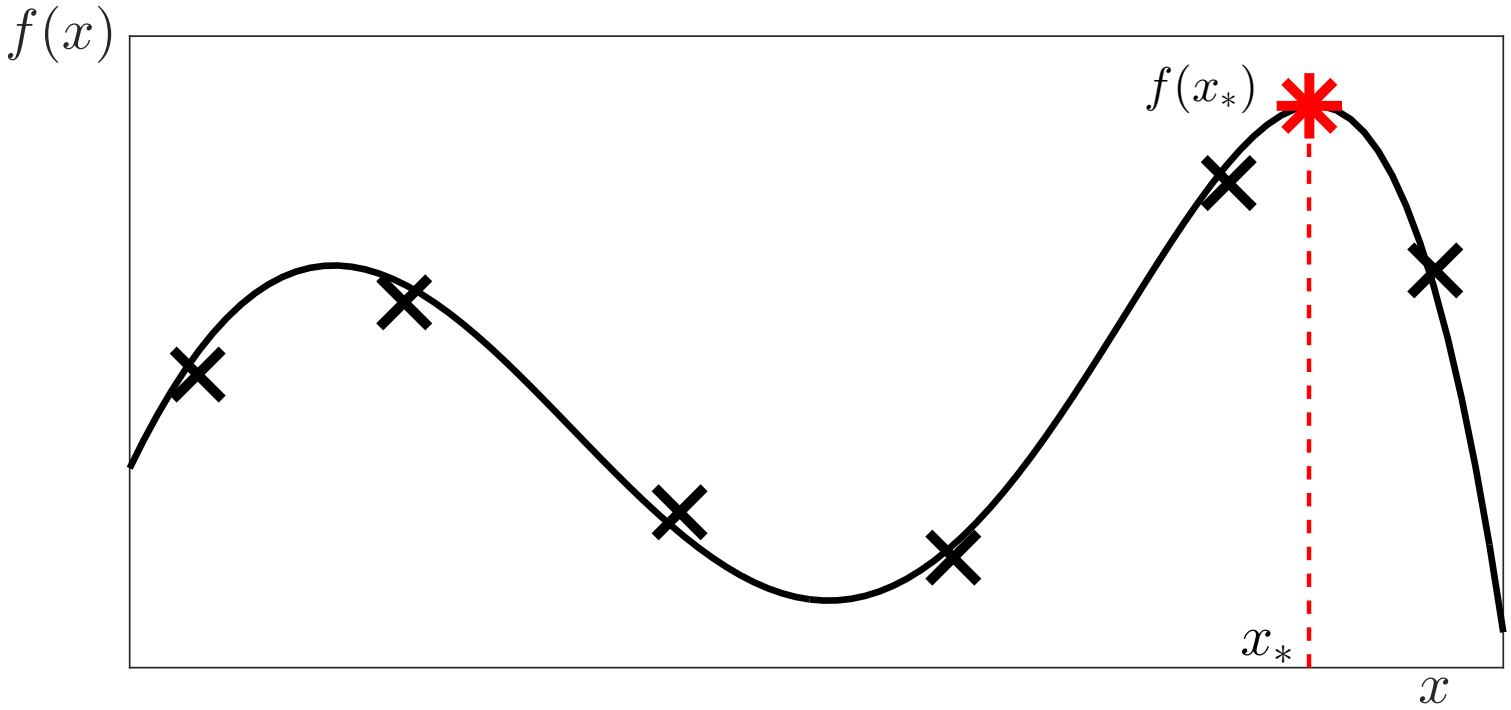


Cumulative Regret after n evaluations

$$\mathrm{CR}(n) = \sum_{t=1}^{n} \Big(f(x_\star) \; - f(x_t) \Big)$$

Black-box Optimisation

$f : \mathcal{X} \to \mathbb{R}$ is an expensive, black-box, noisy function, accessible only via noisy evaluations.
Let $x_\star = \mathrm{argmax}_x f(x)$.



Simple Regret after n evaluations

$$\mathrm{SR}(n) = f(x_\star) - \max_{t=1,\ldots,n} f(x_t).$$

A walk-through Bayesian Optimisation (BO) with Gaussian Processes

- A review of Gaussian Processes (GPs)
- Thompson Sampling (TS): an algorithm for BO
- Other methods and models for BO

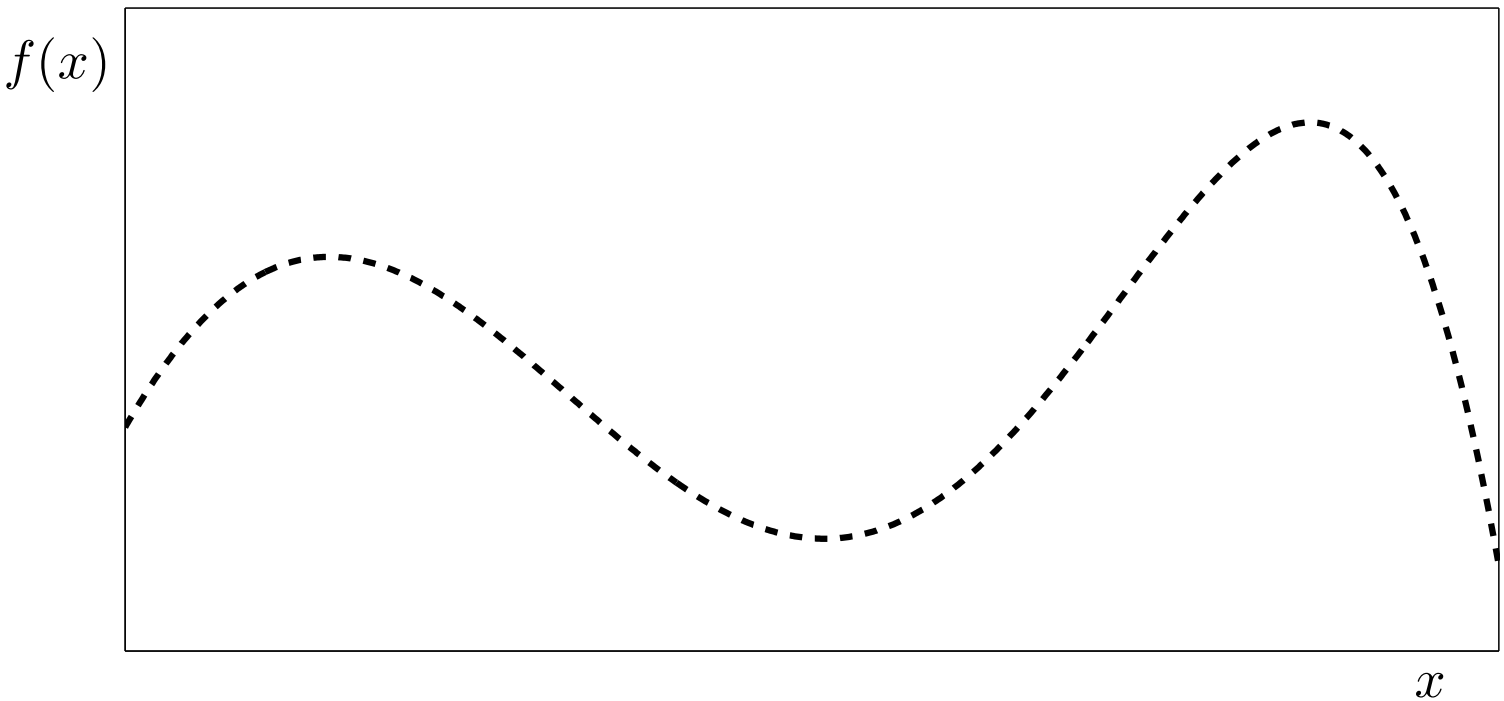
Gaussian Processes ($\mathcal{GP}$)

$\mathcal{GP}(\mu, \kappa)$: A distribution over functions from $\mathcal{X}$ to $\mathbb{R}$.

Gaussian Processes ($\mathcal{GP}$)

$\mathcal{GP}(\mu, \kappa)$: A distribution over functions from $\mathcal{X}$ to $\mathbb{R}$.

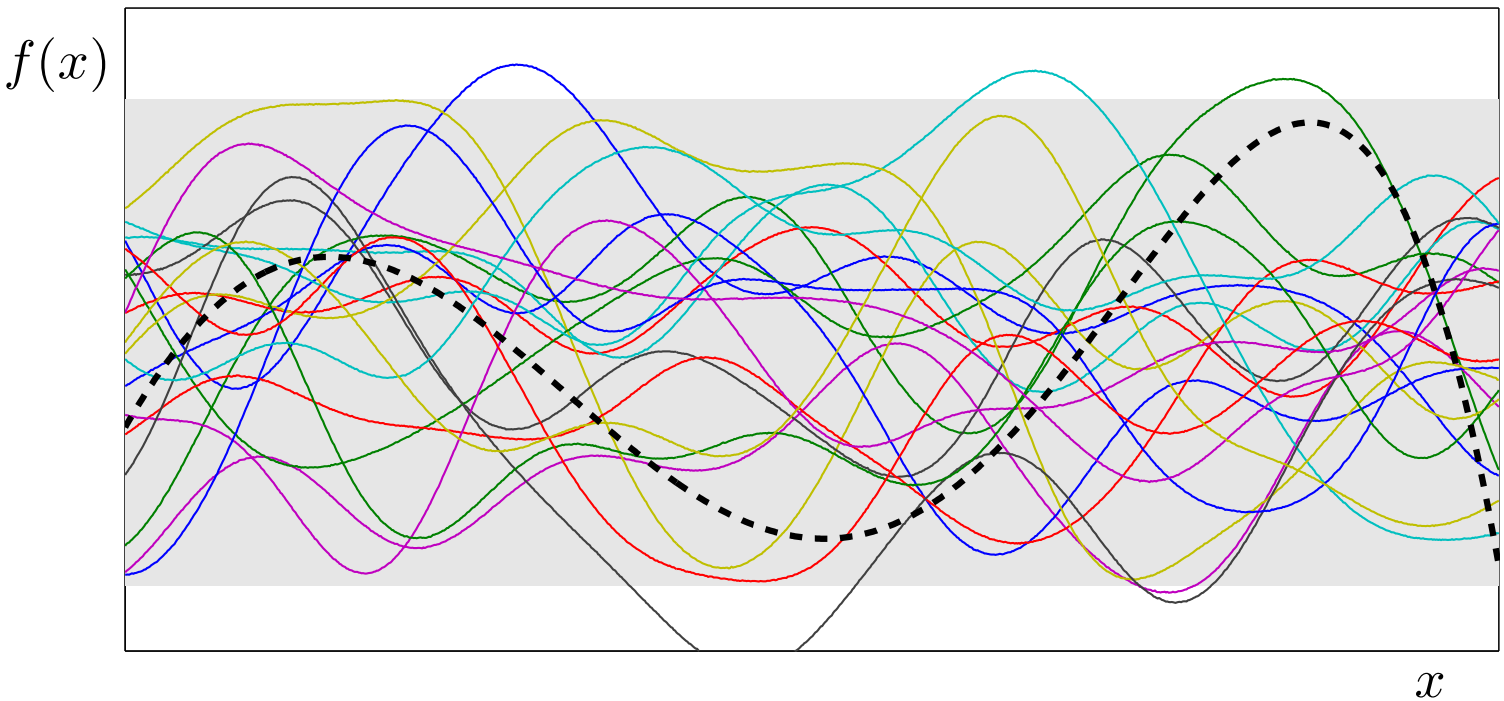
Functions with no observations



Gaussian Processes ($\mathcal{GP}$)

$\mathcal{GP}(\mu, \kappa)$: A distribution over functions from $\mathcal{X}$ to $\mathbb{R}$.

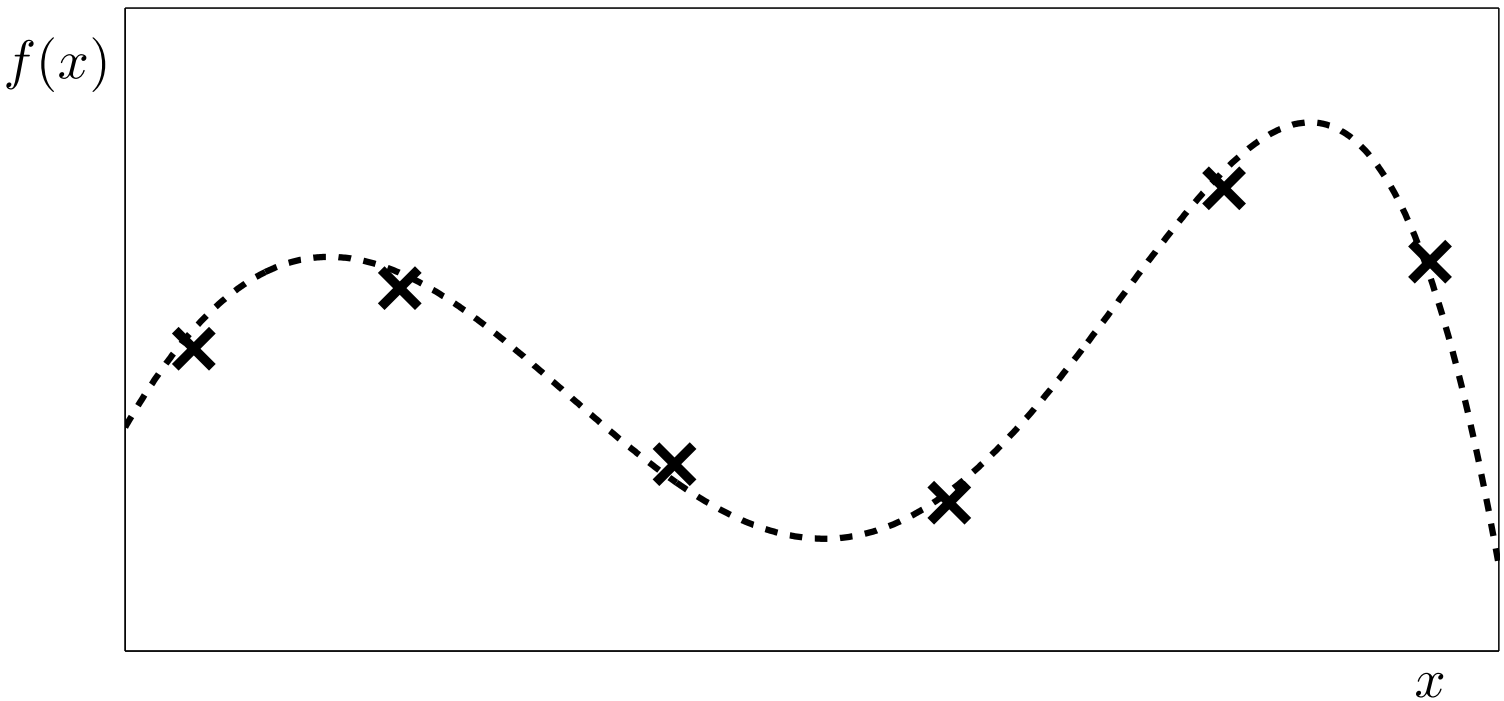
Prior $\mathcal{GP}$



Gaussian Processes ($\mathcal{GP}$)

$\mathcal{GP}(\mu, \kappa)$: A distribution over functions from $\mathcal{X}$ to $\mathbb{R}$.

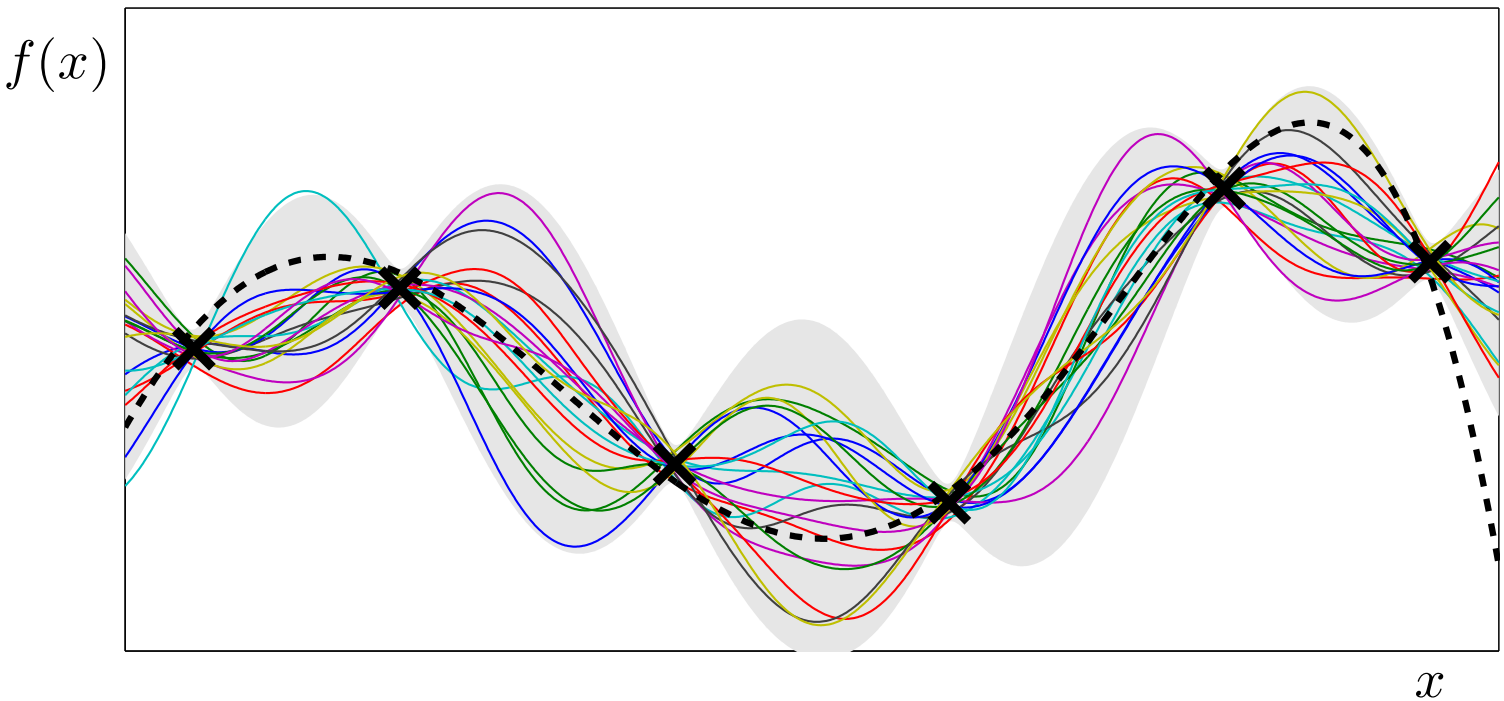
Observations



Gaussian Processes ($\mathcal{GP}$)

$\mathcal{GP}(\mu, \kappa)$: A distribution over functions from $\mathcal{X}$ to $\mathbb{R}$.

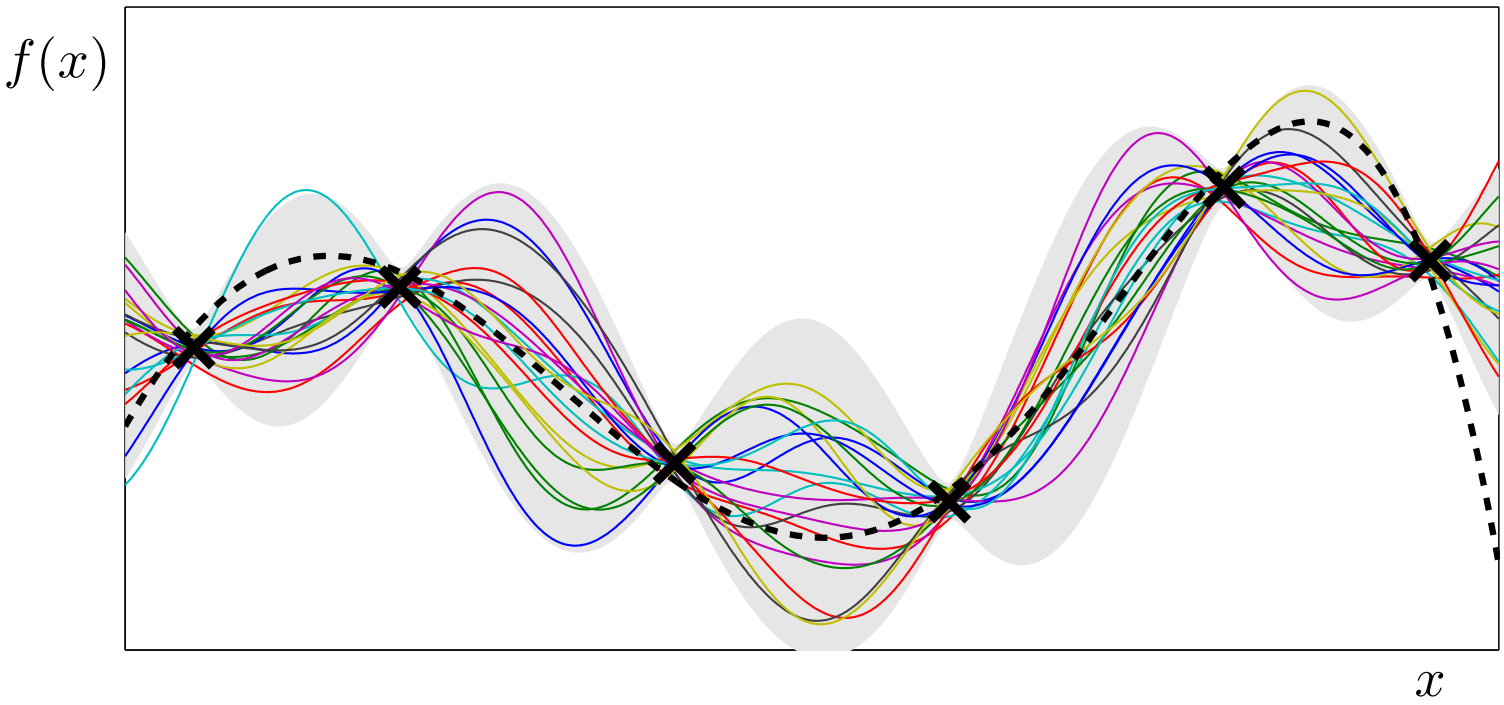
Posterior $\mathcal{GP}$ given observations



Gaussian Processes ($\mathcal{GP}$)

$\mathcal{GP}(\mu, \kappa)$: A distribution over functions from $\mathcal{X}$ to $\mathbb{R}$.

Posterior $\mathcal{GP}$ given observations

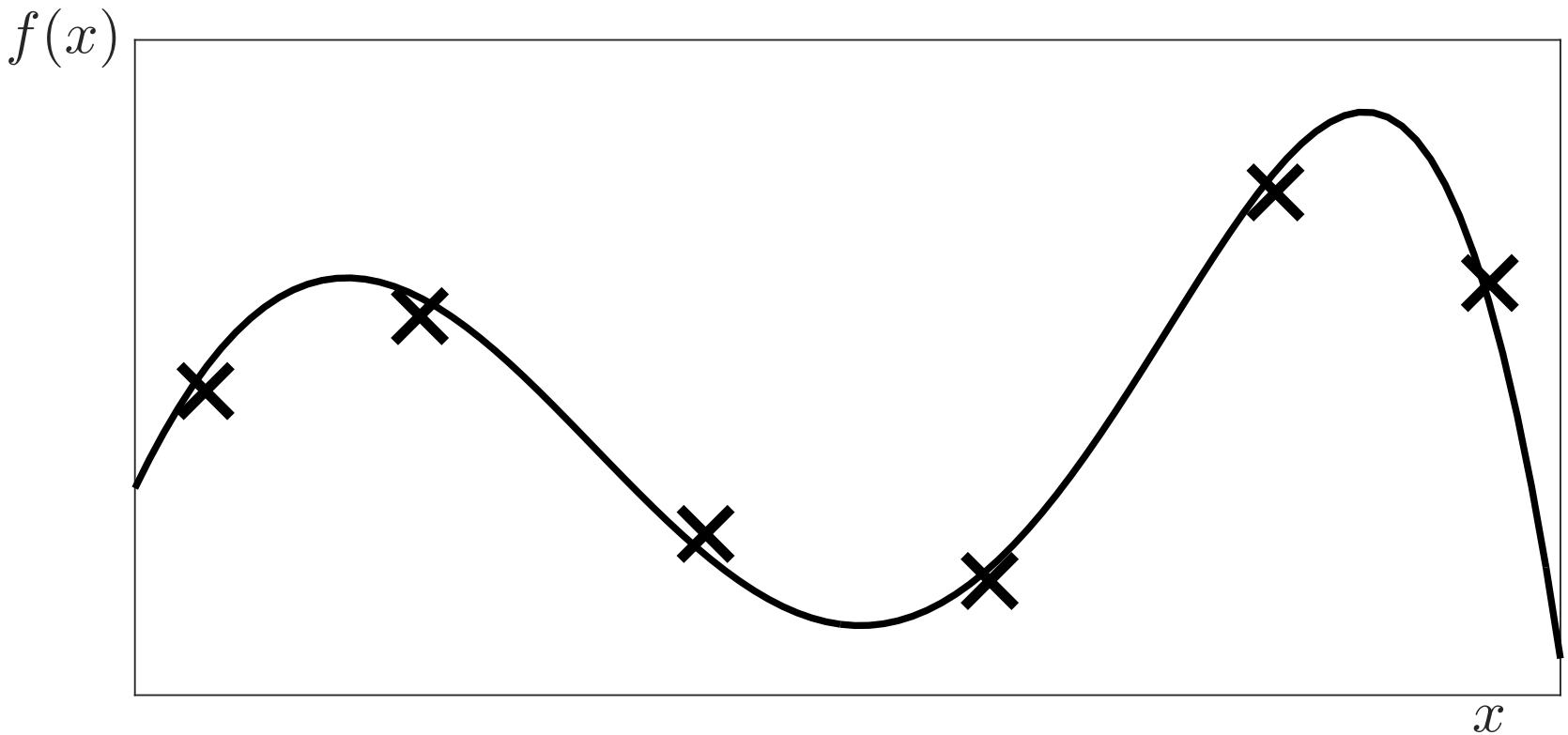


Completely characterised by mean function $\mu : \mathcal{X} \to \mathbb{R}$, and covariance kernel $\kappa : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$.
After t observations, $\quad f(x) \sim \mathcal{N}(\, \mu_t(x), \sigma_t^2(x)\,)$.

Gaussian Process (Bayesian) Optimisation

Model $f \sim \mathcal{GP}(\mathbf{0}, \kappa)$.

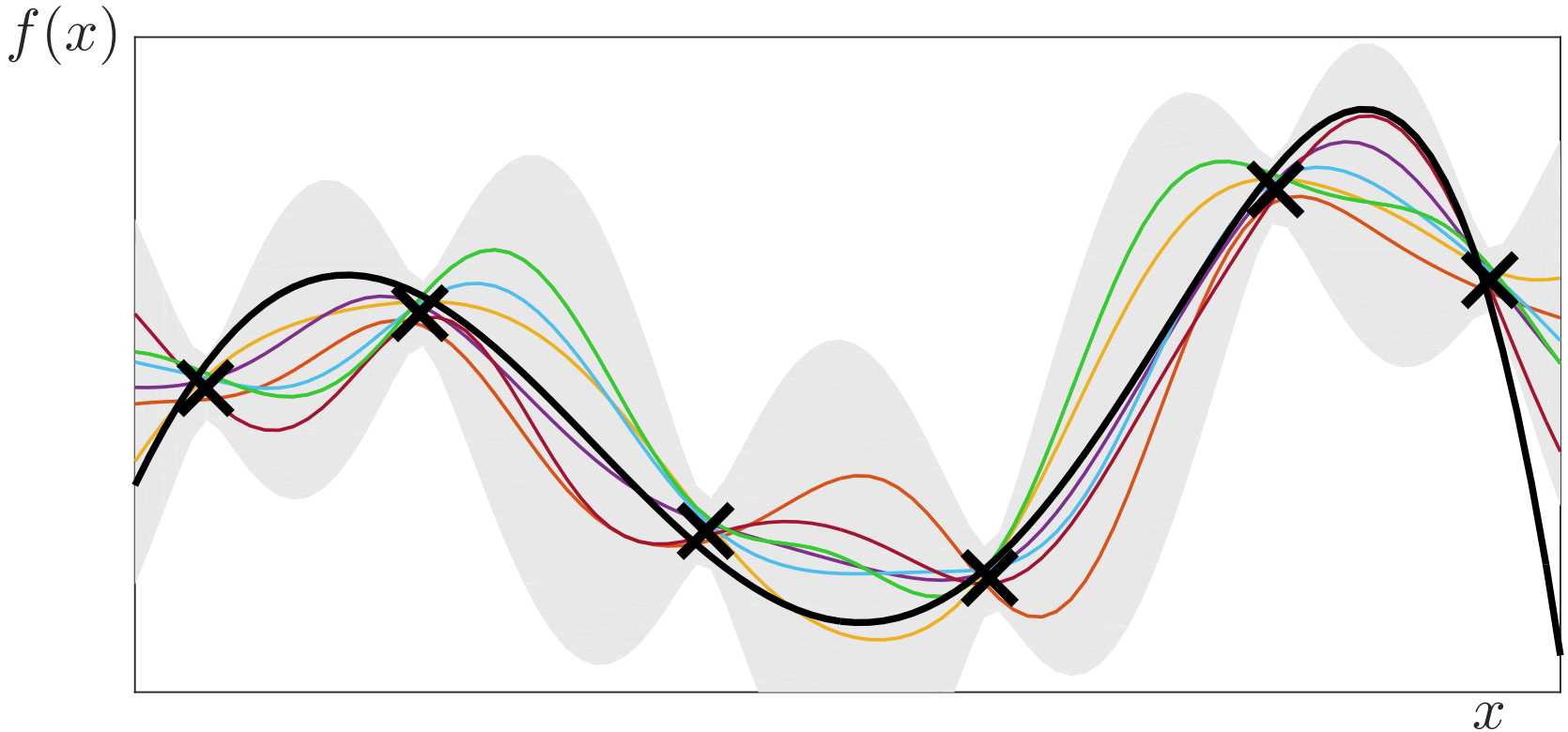
Thompson Sampling (TS) (Thompson, 1933).



Gaussian Process (Bayesian) Optimisation

Model $f \sim \mathcal{GP}(\mathbf{0}, \kappa)$.

Thompson Sampling (TS) (Thompson, 1933).

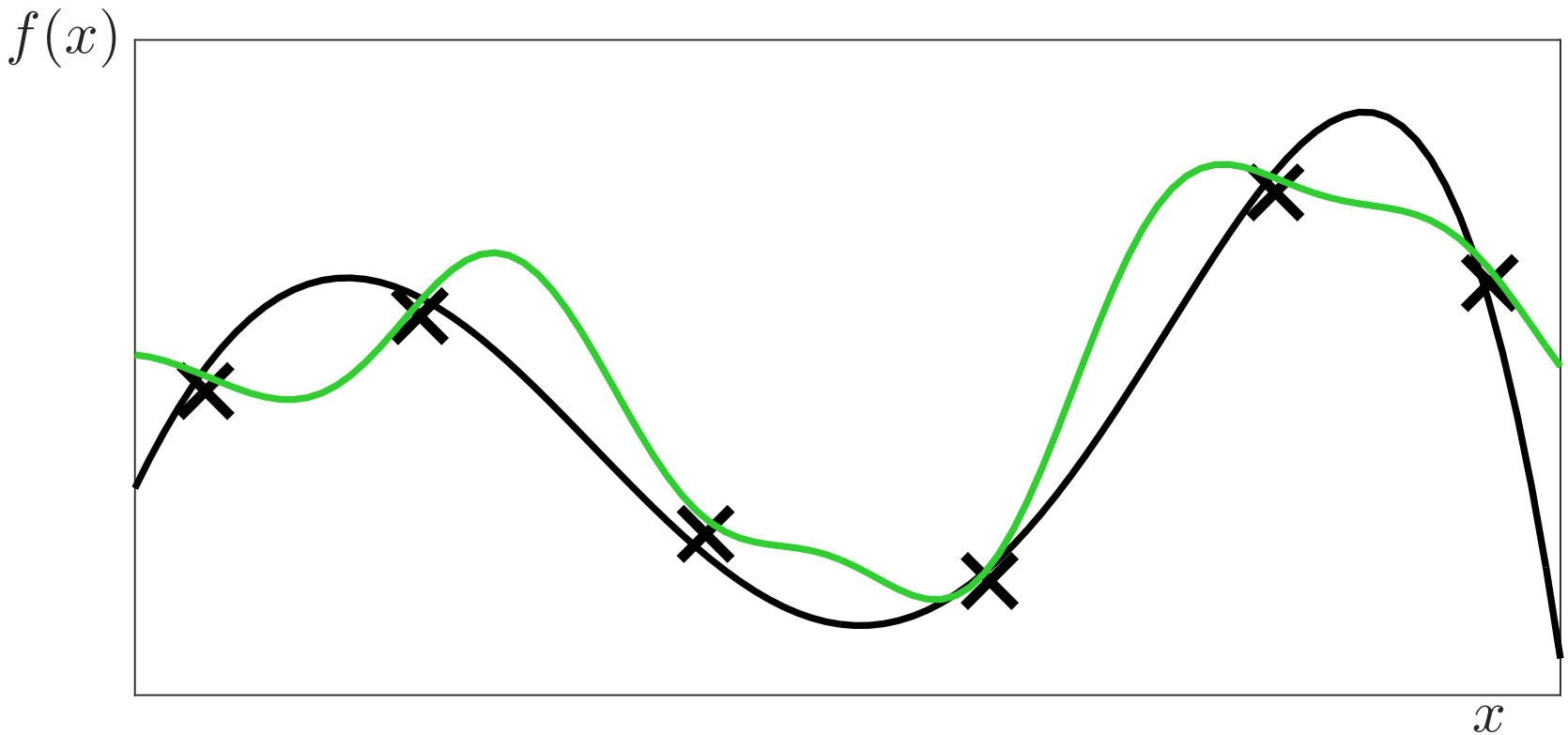


1) Construct posterior $\mathcal{GP}$.

Gaussian Process (Bayesian) Optimisation

Model $f \sim \mathcal{GP}(\mathbf{0}, \kappa)$.

Thompson Sampling (TS) (Thompson, 1933).

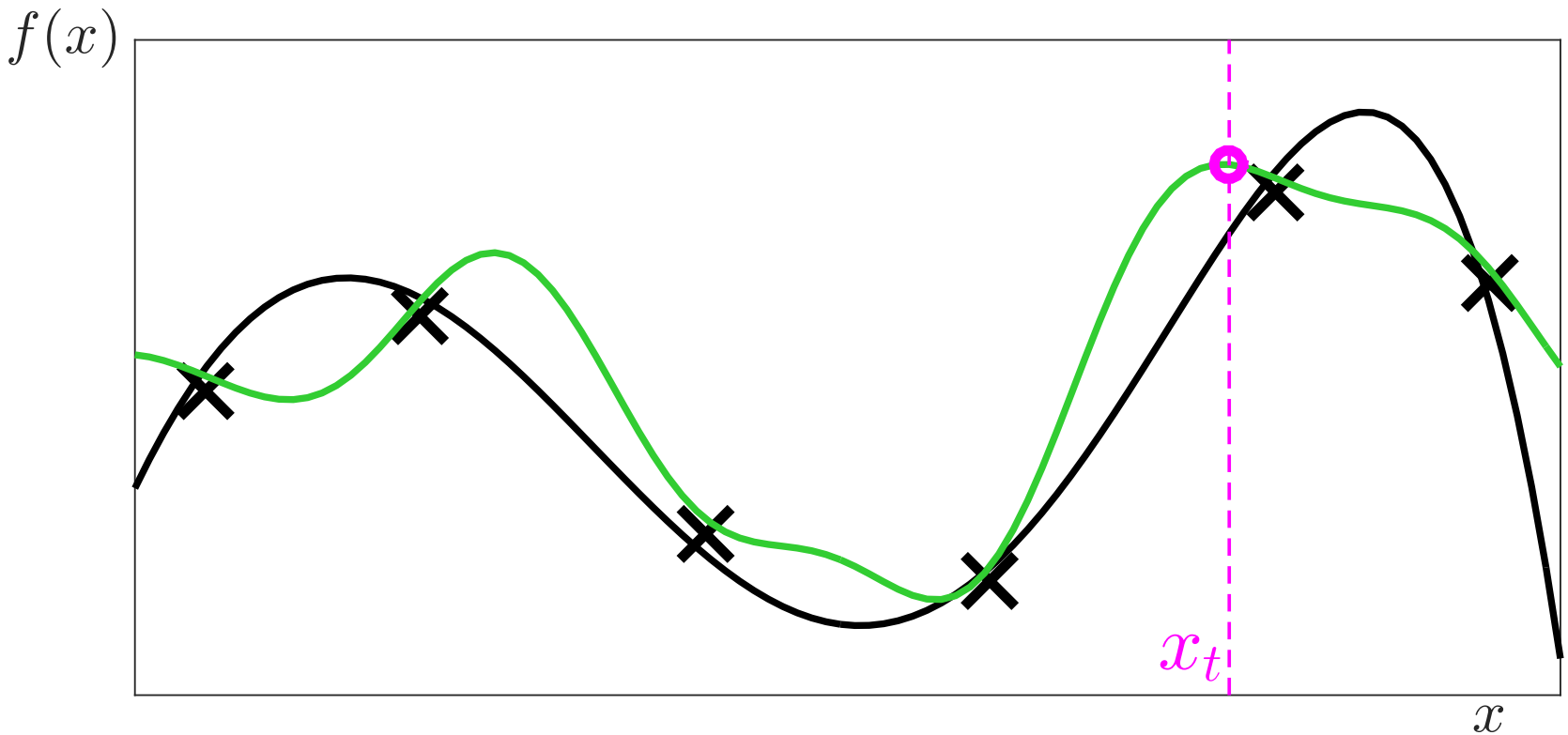


1) Construct posterior $\mathcal{GP}$. 2) Draw sample g from posterior.

Gaussian Process (Bayesian) Optimisation

Model $f \sim \mathcal{GP}(\mathbf{0}, \kappa)$.

Thompson Sampling (TS) (Thompson, 1933).

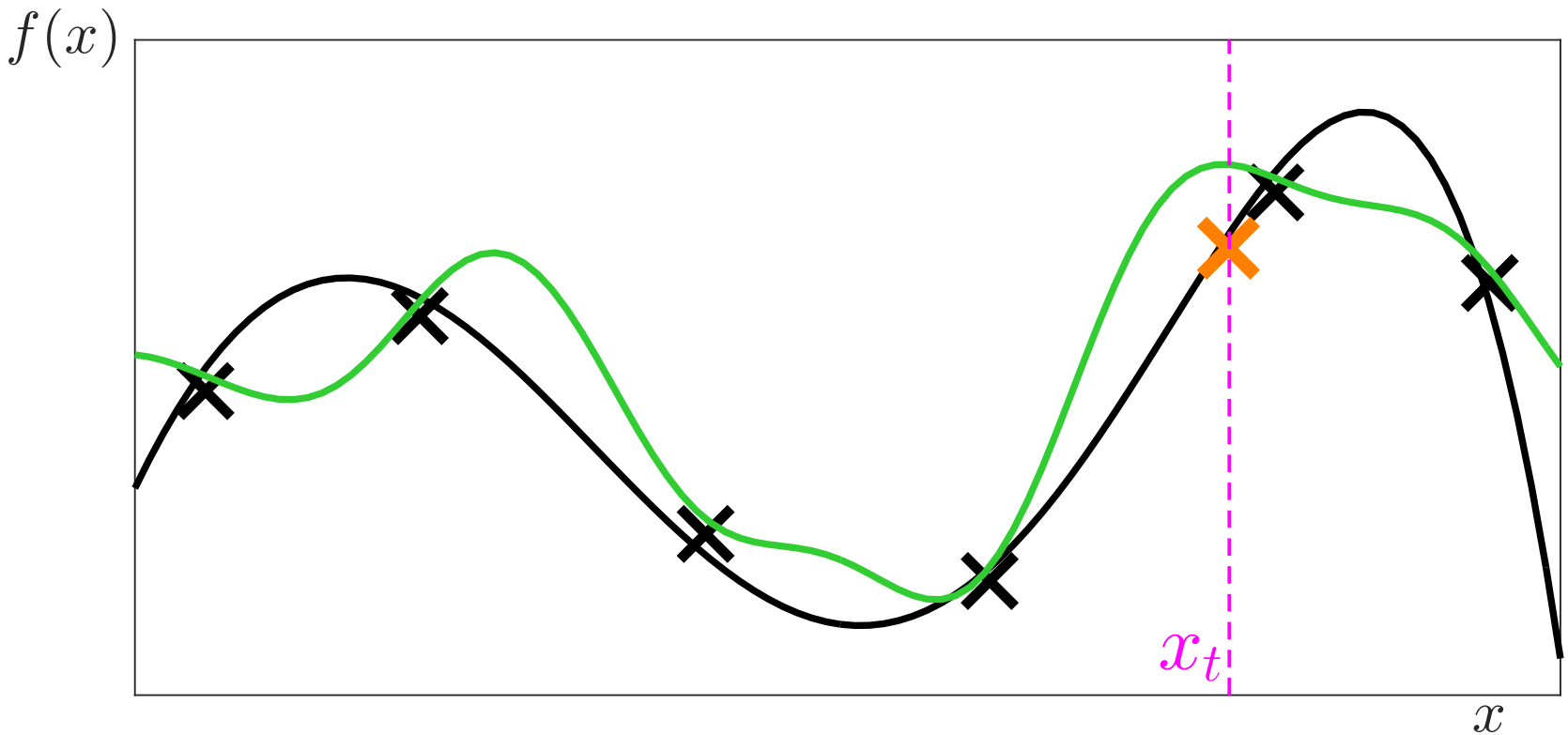


1) Construct posterior $\mathcal{GP}$. 2) Draw sample g from posterior.
3) Choose $x_t = \mathrm{argmax}_x\, g(x)$.

Gaussian Process (Bayesian) Optimisation

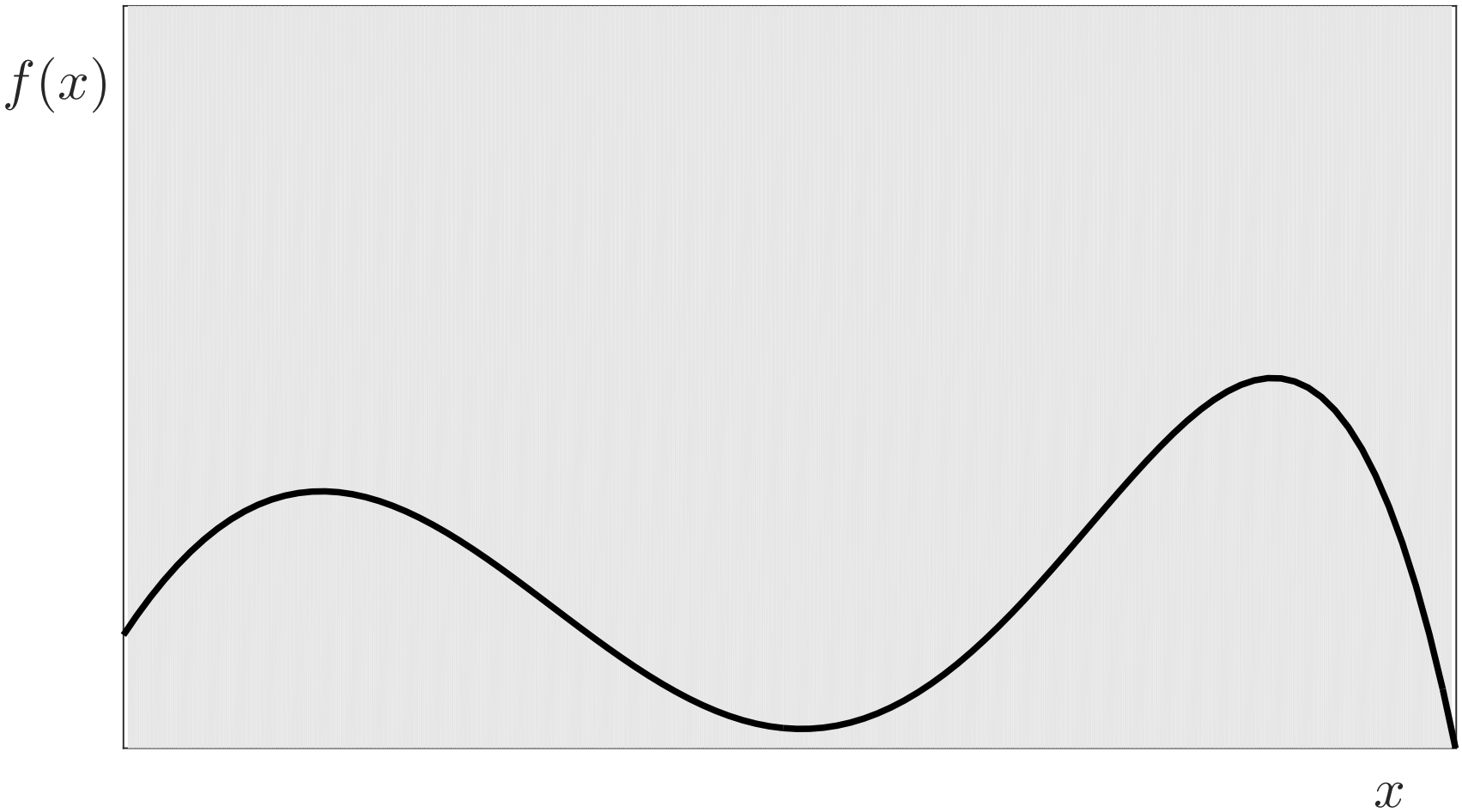
Model $f \sim \mathcal{GP}(\mathbf{0}, \kappa)$.

Thompson Sampling (TS) (Thompson, 1933).

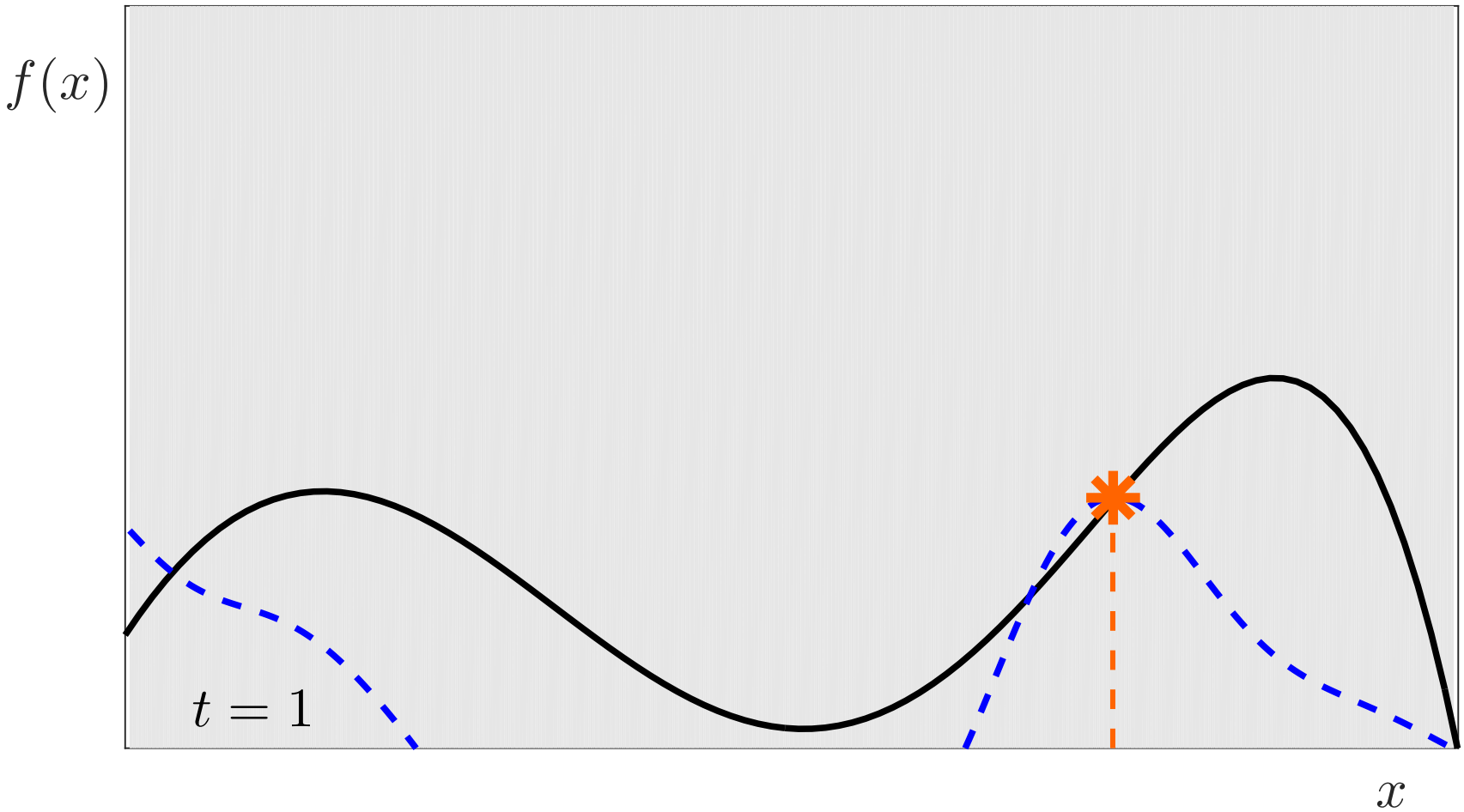


1) Construct posterior $\mathcal{GP}$. 2) Draw sample g from posterior.
3) Choose $x_t = \mathrm{argmax}_x\, g(x)$. 4) Evaluate f at x_t.

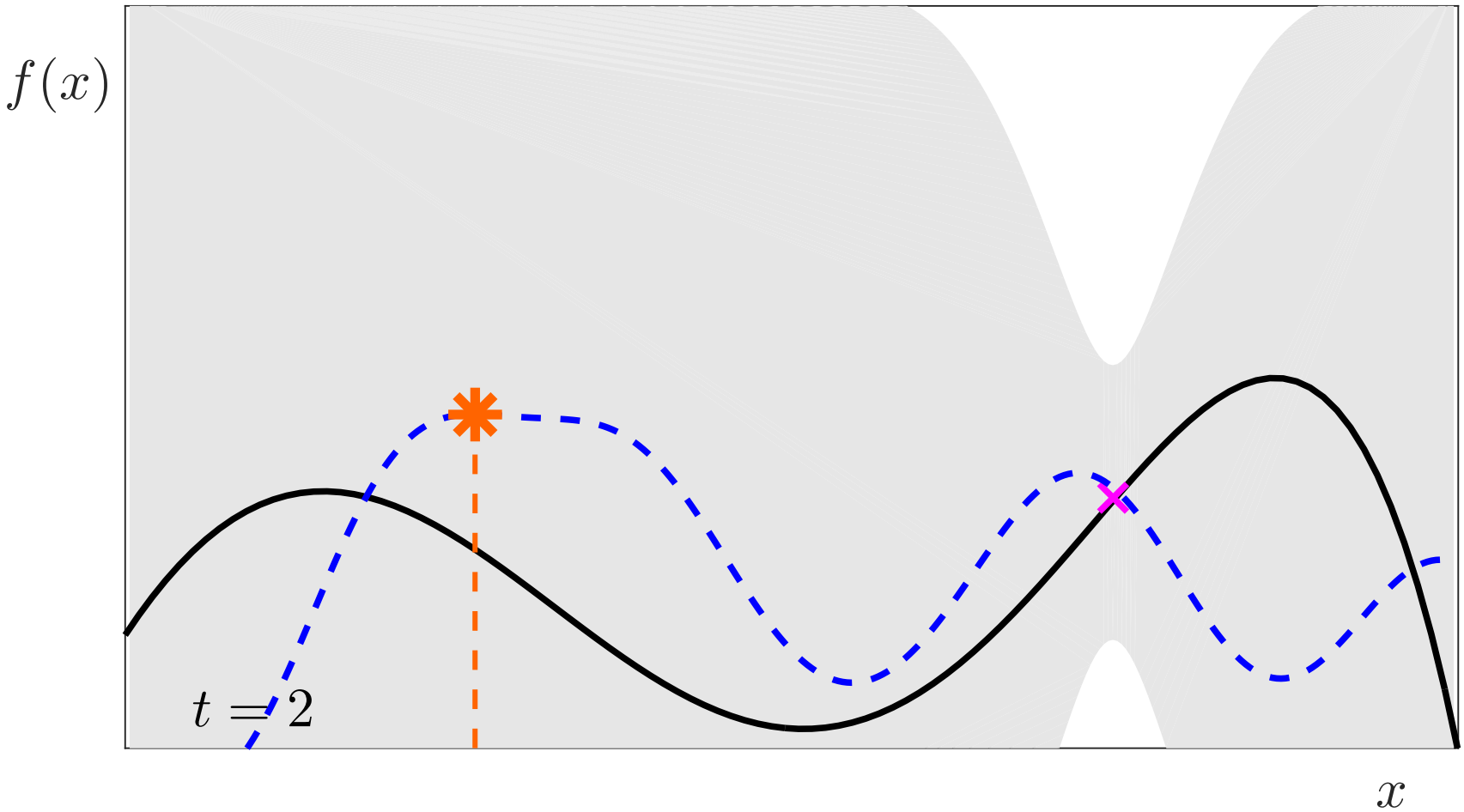
Thompson Sampling (TS) in GPs (Thompson, 1933)



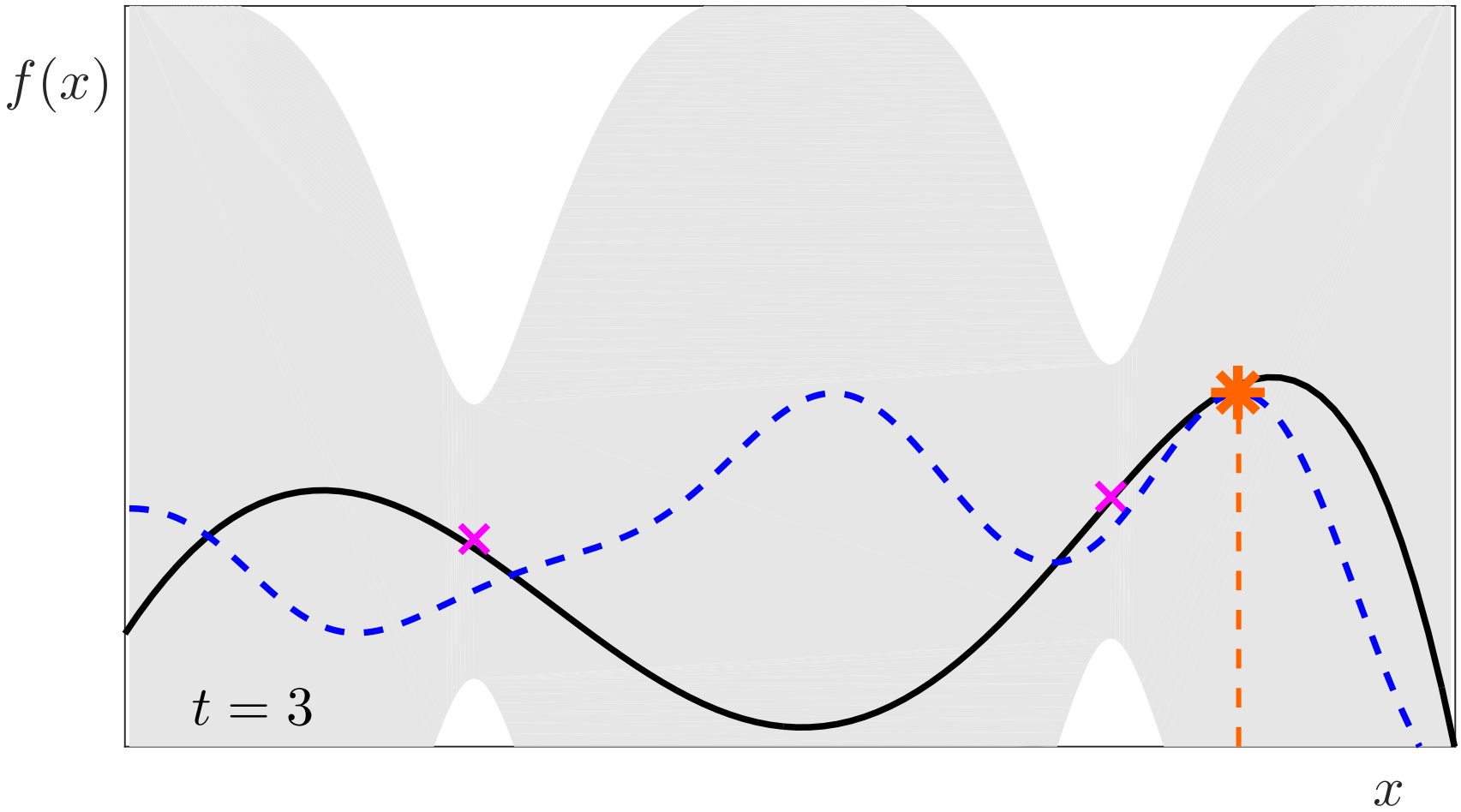
Thompson Sampling (TS) in GPs (Thompson, 1933)



Thompson Sampling (TS) in GPs (Thompson, 1933)

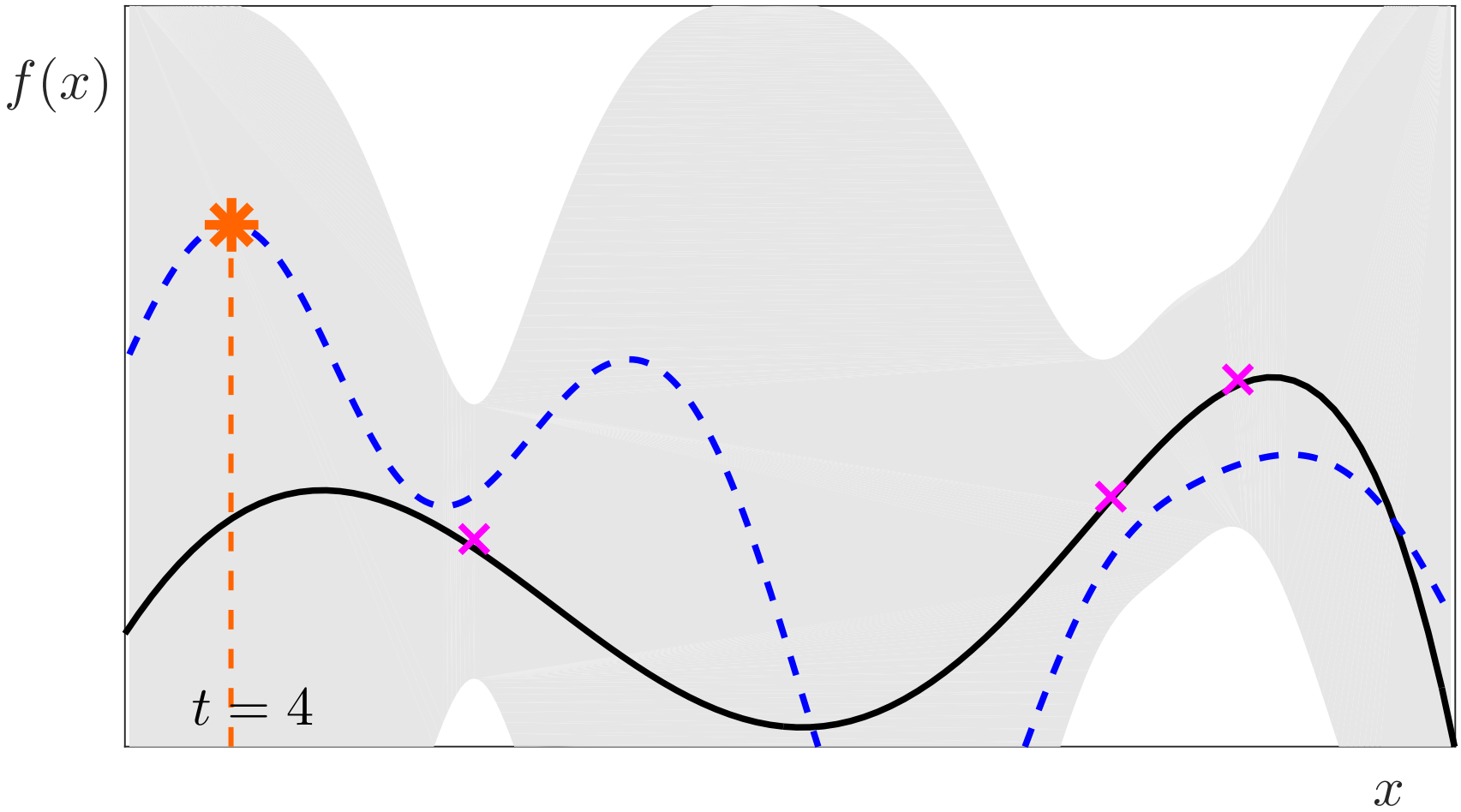


Thompson Sampling (TS) in GPs (Thompson, 1933)

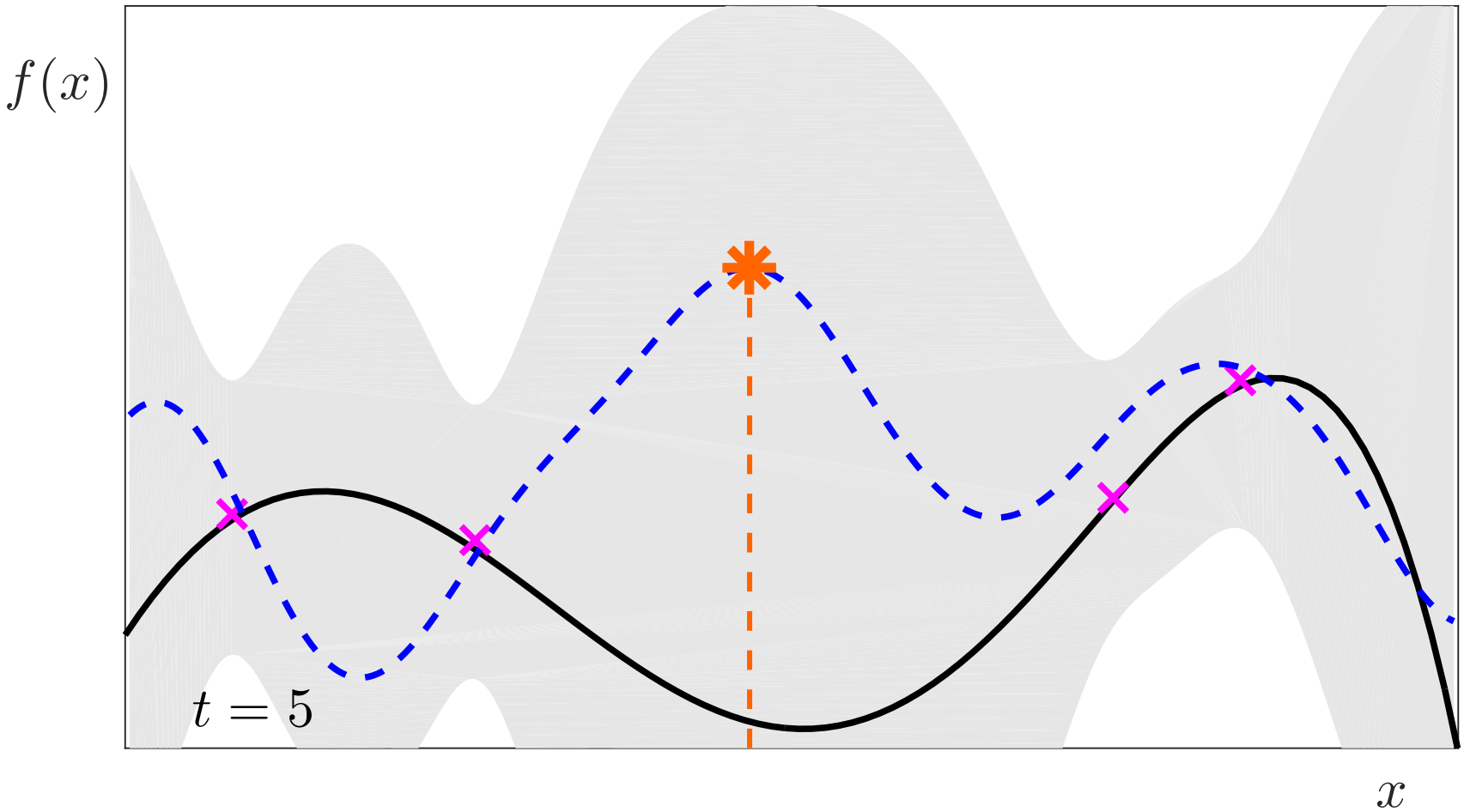


Thompson Sampling (TS) in GPs

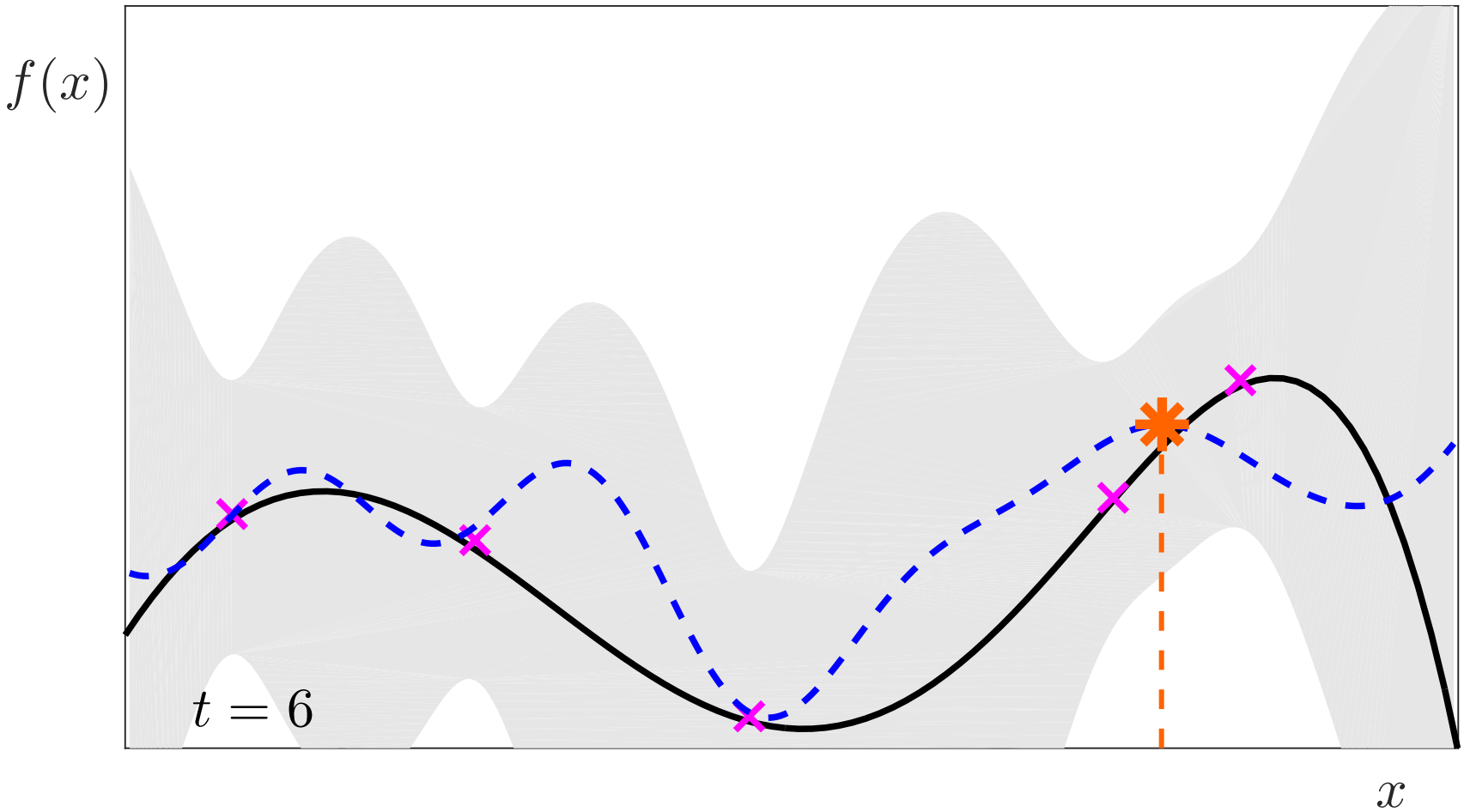
(Thompson, 1933)



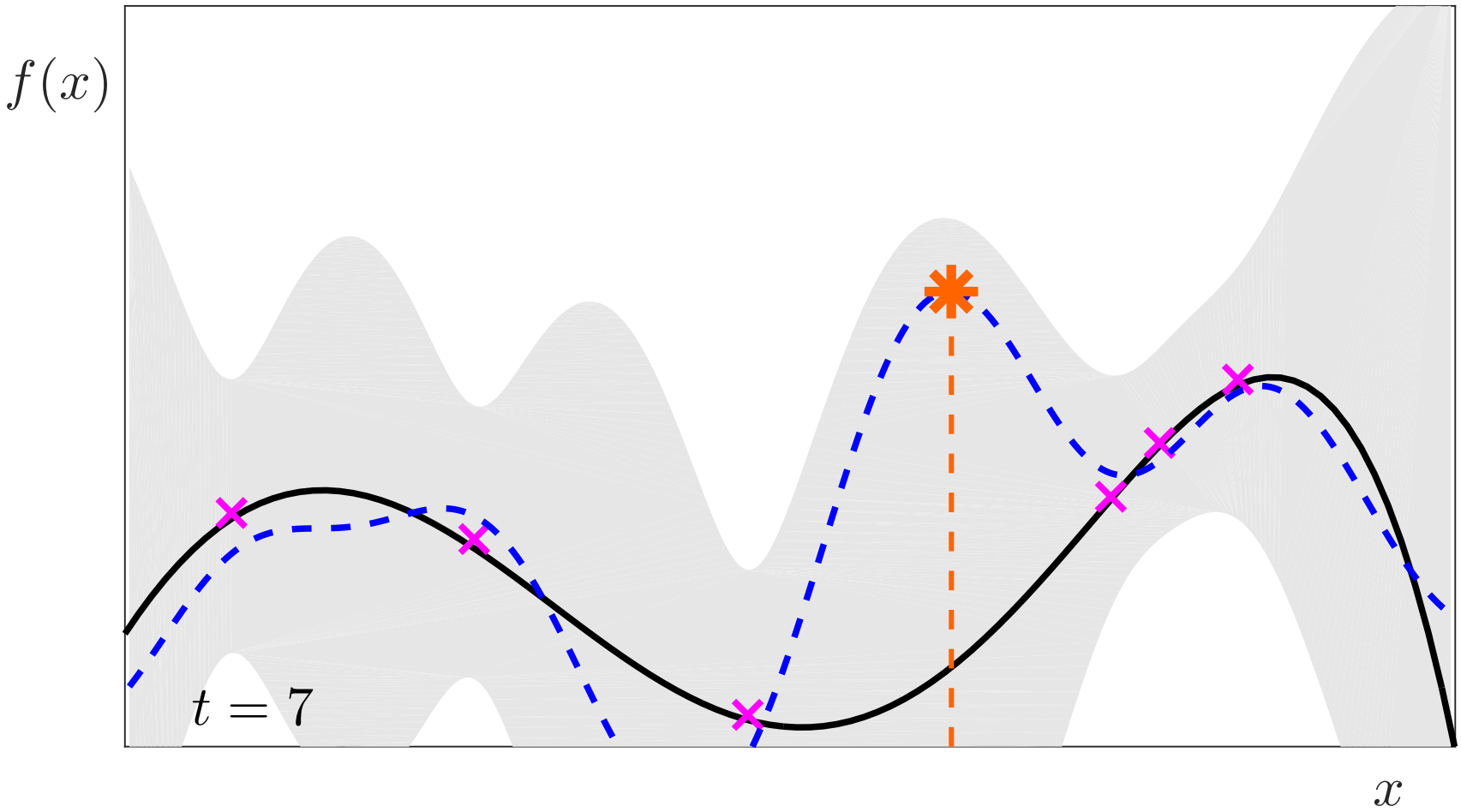
Thompson Sampling (TS) in GPs (Thompson, 1933)



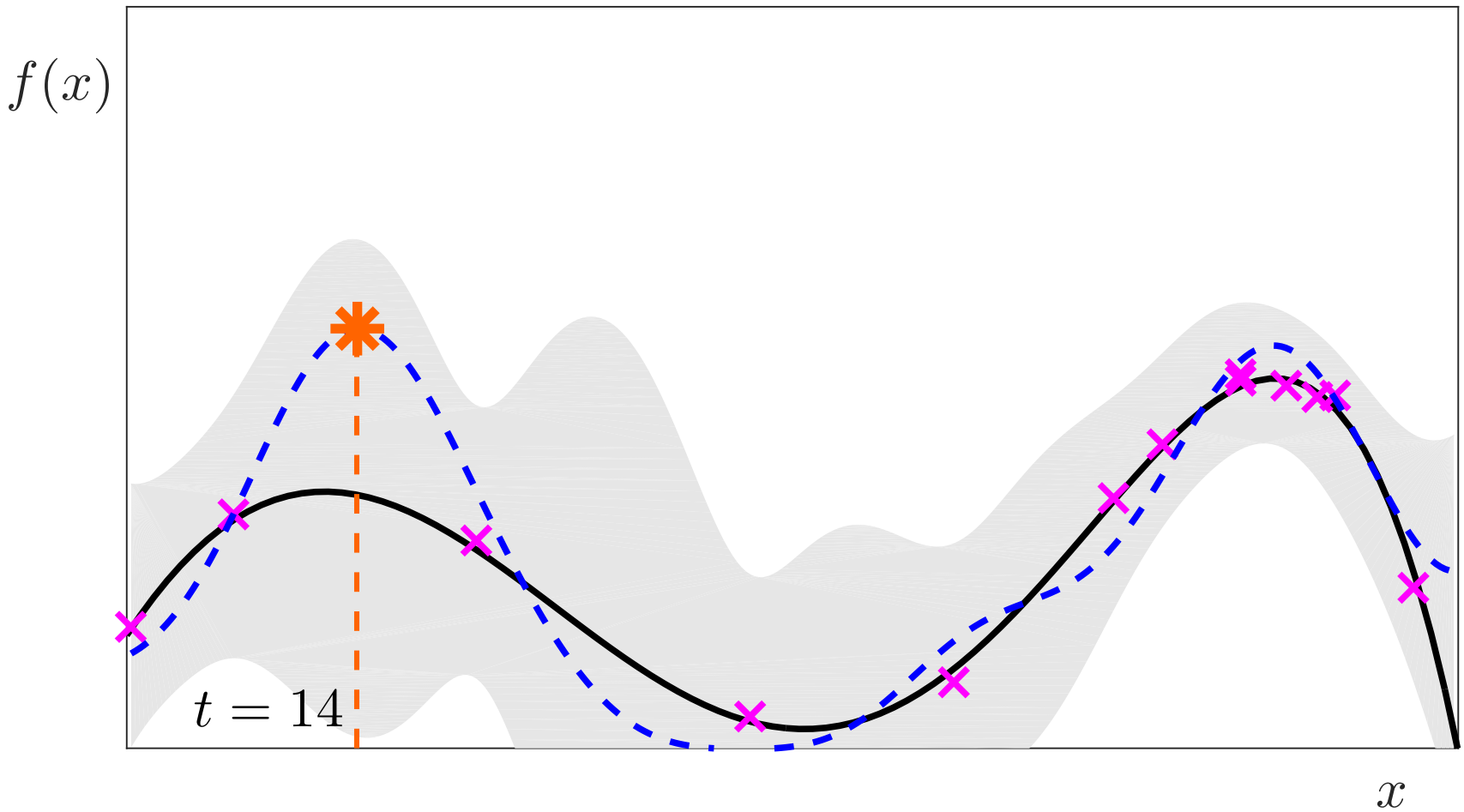
Thompson Sampling (TS) in GPs (Thompson, 1933)



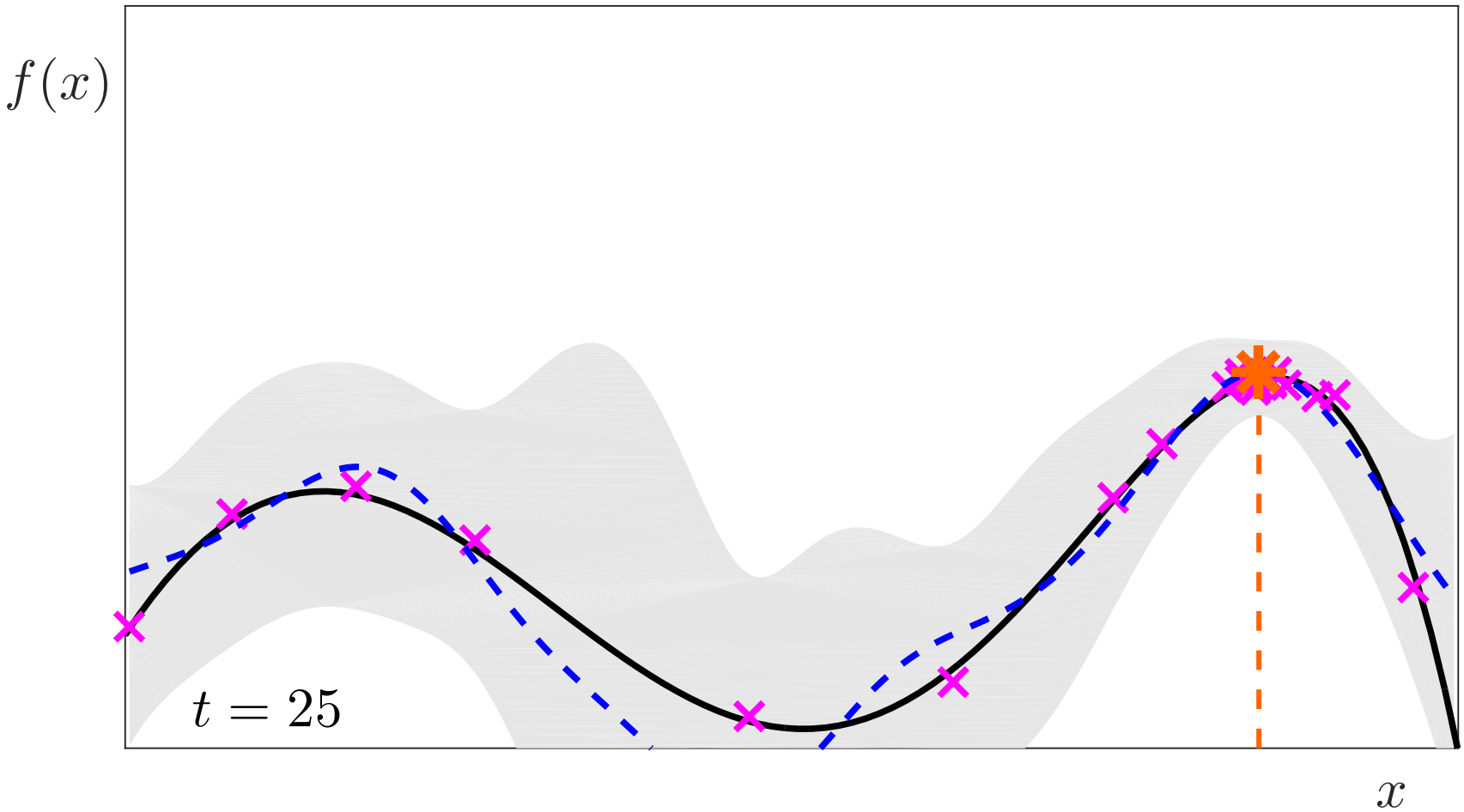
Thompson Sampling (TS) in GPs (Thompson, 1933)



Thompson Sampling (TS) in GPs (Thompson, 1933)



Thompson Sampling (TS) in GPs (Thompson, 1933)



Some Theoretical Results for TS

Simple Regret: $\mathrm{SR}(n) = f(x_\star) - \max_{t=1,\ldots,n} f(x_t)$

Some Theoretical Results for TS

Simple Regret: $\mathrm{SR}(n) = f(x_\star) - \max_{t=1,\ldots,n} f(x_t)$

Theorem: For Thompson sampling, (Russo & van Roy 2014, Srinivas et al. 2010)

$$\mathbb{E}[\mathrm{SR}(n)] \lesssim \sqrt{\frac{\Psi_n \log(n)}{n}}.$$

$\lesssim$ ignores constants

$\Psi_n \leftarrow$ Maximum Information Gain.

Some Theoretical Results for TS

Simple Regret: $\mathrm{SR}(n) = f(x_\star) - \max_{t=1,\ldots,n} f(x_t)$

Theorem: For Thompson sampling, (Russo & van Roy 2014, Srinivas et al. 2010)

$$\mathbb{E}[\mathrm{SR}(n)] \lesssim \sqrt{\frac{\Psi_n \log(n)}{n}}.$$

$\lesssim$ ignores constants

$\Psi_n \leftarrow$ Maximum Information Gain.

When $\mathcal{X} \subset \mathbb{R}^d$, SE (Gaussian) kernel: $\Psi_n \asymp d^d \log(n)^d$.

Matérn kernel: $\Psi_n \asymp n^{1-\frac{c}{d^2}}$.

Some Theoretical Results for TS

Simple Regret: $\mathrm{SR}(n) = f(x_\star) - \max_{t=1,\ldots,n} f(x_t)$

Theorem: For Thompson sampling, (Russo & van Roy 2014, Srinivas et al. 2010)

$$\mathbb{E}[\mathrm{SR}(n)] \lesssim \sqrt{\frac{\Psi_n \log(n)}{n}}.$$

$\lesssim$ ignores constants

$\Psi_n \leftarrow$ Maximum Information Gain.

When $\mathcal{X} \subset \mathbb{R}^d$, SE (Gaussian) kernel: $\Psi_n \asymp d^d \log(n)^d$.

Matérn kernel: $\Psi_n \asymp n^{1-\frac{c}{d^2}}$.

Several other results: (Agrawal et al 2012, Kaufmann et al 2012, Russo & van Roy 2016, Chowdhury & Gopalan 2017 and more . . .)

Other methods for BO

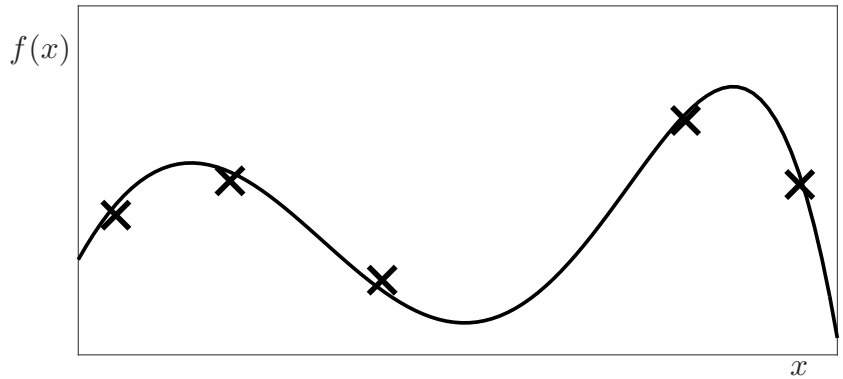
Other criteria for selecting x_t:

- Upper Confidence Bounds (Srinivas et al. 2010)

Other methods for BO

Other criteria for selecting x_t:

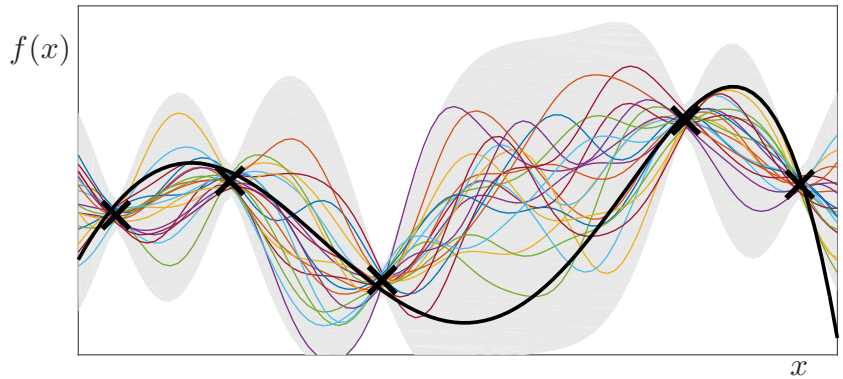
- Upper Confidence Bounds (Srinivas et al. 2010)



Other methods for BO

Other criteria for selecting x_t:

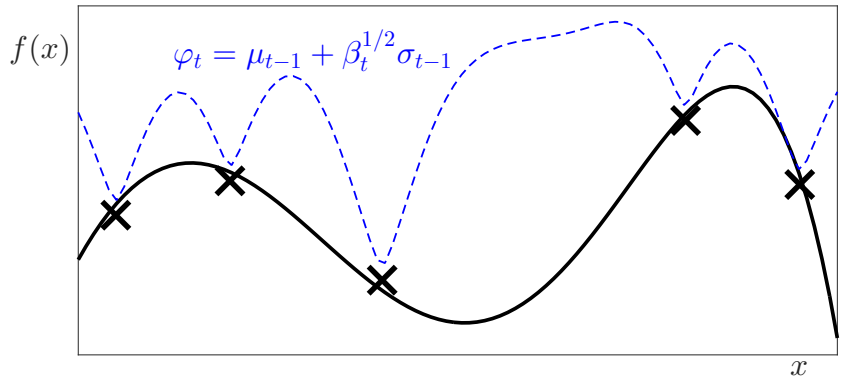
- Upper Confidence Bounds (Srinivas et al. 2010)



Other methods for BO

Other criteria for selecting x_t:

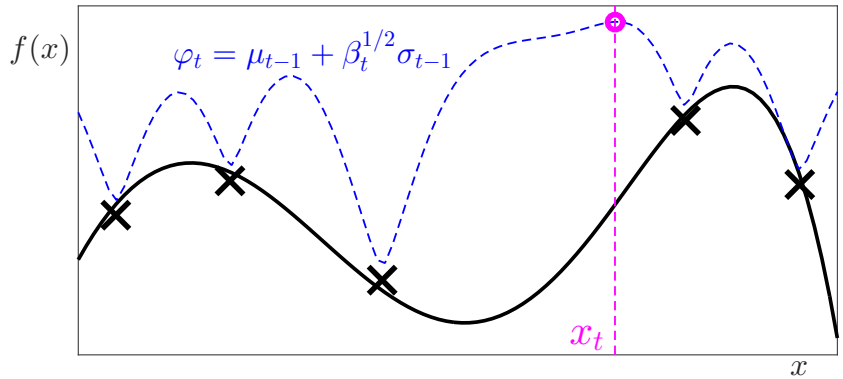
- Upper Confidence Bounds (Srinivas et al. 2010)



Other methods for BO

Other criteria for selecting x_t:

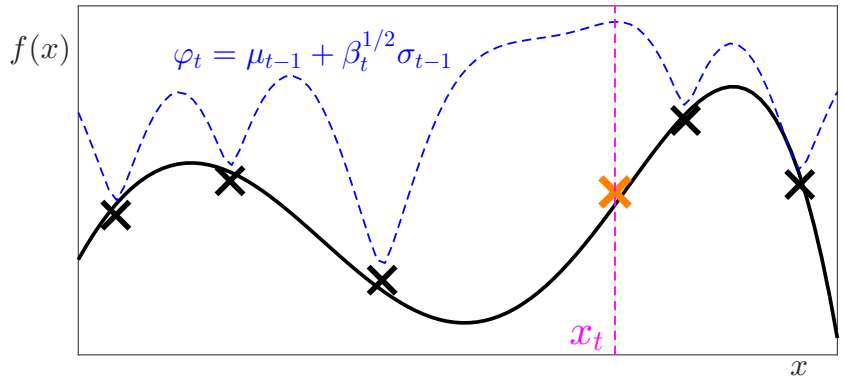
- Upper Confidence Bounds (Srinivas et al. 2010)



Other methods for BO

Other criteria for selecting x_t:

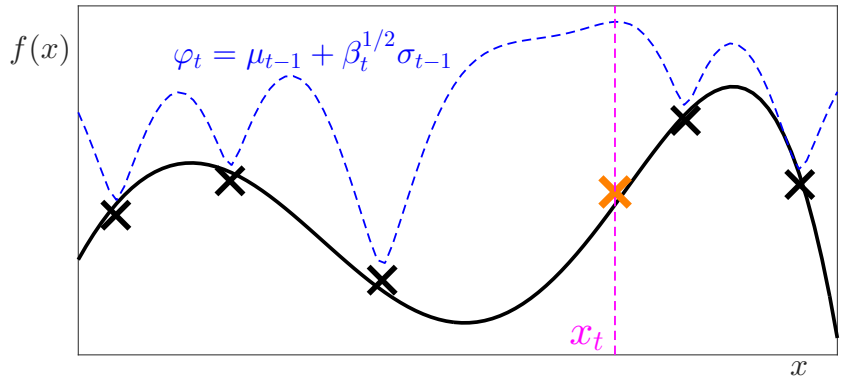
- Upper Confidence Bounds (Srinivas et al. 2010)



Other methods for BO

Other criteria for selecting x_t:

- Upper Confidence Bounds (Srinivas et al. 2010)



- Expected improvement (Jones et al. 1998)
- Probability of improvement (Kushner et al. 1964)
- Entropy search (Hernández-Lobato et al. 2014)

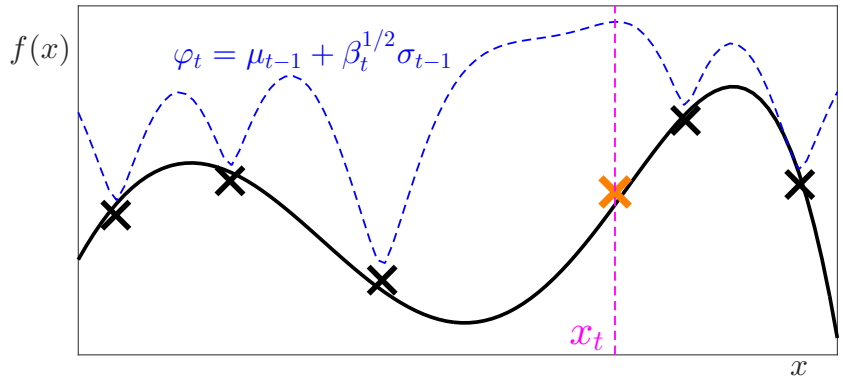
All deterministic methods, choose next point for evaluation by maximising a *deterministic* acquisition function,

i.e. $x_t = \operatorname{argmax}_{x \in \mathcal{X}} \varphi_t(x)$.

Other methods for BO

Other criteria for selecting x_t:

- Upper Confidence Bounds (Srinivas et al. 2010)



- Expected improvement (Jones et al. 1998)
- Probability of improvement (Kushner et al. 1964)
- Entropy search (Hernández-Lobato et al. 2014)

All deterministic methods, choose next point for evaluation by maximising a *deterministic* acquisition function,

i.e. $x_t = \operatorname{argmax}_{x \in \mathcal{X}} \varphi_t(x)$.

Other models for f: Neural networks (Snoek et al. 2015), Random Forests (Hutter 2009).

Big picture: scaling up black-box optimisation

Big picture: scaling up black-box optimisation

- Optimising in high dimensional spaces
  e.g.: Tuning models with several hyper-parameters
  Additive models for f lead to statistically and computationally tractable algorithms. (Kandasamy et al. ICML 2015)

Big picture: scaling up black-box optimisation

- Optimising in high dimensional spaces
  e.g.: Tuning models with several hyper-parameters
  Additive models for f lead to statistically and computationally tractable algorithms. (Kandasamy et al. ICML 2015)

- Multi-fidelity optimisation: what if we have cheap approximations to f?
  E.g. Train an ML model with $N_\bullet$ data and $T_\bullet$ iterations.
  But use $N < N_\bullet$ data and $T < T_\bullet$ iterations to approximate cross validation performance at $(N_\bullet, T_\bullet)$.
  (Kandasamy et al. NIPS 2016a&b, Kandasamy et al. ICML 2017)

Big picture: scaling up black-box optimisation

- Optimising in high dimensional spaces
  e.g.: Tuning models with several hyper-parameters
  Additive models for f lead to statistically and computationally tractable algorithms. (Kandasamy et al. ICML 2015)

- Multi-fidelity optimisation: what if we have cheap approximations to f?
  E.g. Train an ML model with $N_\bullet$ data and $T_\bullet$ iterations.
  But use $N < N_\bullet$ data and $T < T_\bullet$ iterations to approximate cross validation performance at $(N_\bullet, T_\bullet)$.
  (Kandasamy et al. NIPS 2016a&b, Kandasamy et al. ICML 2017)

Extends beyond GPs.

This work: Parallel Evaluations (Kandasamy et al. Arxiv 2017)

Parallelisation with M workers: can evaluate f at M different points at the same time.
E.g. Train M models with different hyper-parameter values in parallel at the same time.

Inability to parallelise is a real bottleneck in practice!

This work: Parallel Evaluations (Kandasamy et al. Arxiv 2017)

Parallelisation with M workers: can evaluate f at M different points at the same time.
E.g. Train M models with different hyper-parameter values in parallel at the same time.

Inability to parallelise is a real bottleneck in practice!

Some desiderata:

- Statistically, achieve $\times M$ improvement.
- Methodologically, be scalable for a very large number of workers,
  - Method remains computationally tractable as M increases.
  - Method is conceptually simple, for robustness in practice.

Outline

(Kandasamy et al. Arxiv 2017)

1. Set up & definitions
2. Prior work & challenges
3. **Algorithms** synTS, asyTS: direct application of TS to synchronous and asynchronous parallel settings
4. Experiments
5. Theoretical Results

6. Open questions/challenges

Outline

(Kandasamy et al. Arxiv 2017)

1. Set up & definitions
2. Prior work & challenges
3. **Algorithms** synTS, asyTS: direct application of TS to synchronous and asynchronous parallel settings
4. Experiments
5. Theoretical Results
   - synTS and asyTS perform essentially the same as seqTS in terms of the number of evaluations.
   - When we factor time as a resource, asyTS outperforms synTS and seqTS.

   ... with some caveats.
6. Open questions/challenges

Outline

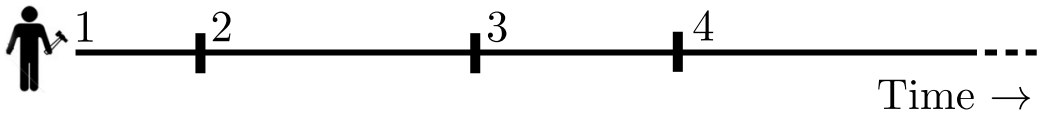
(Kandasamy et al. Arxiv 2017)

1. Set up & definitions
2. Prior work & challenges
3. **Algorithms** synTS, asyTS: direct application of TS to synchronous and asynchronous parallel settings
4. Experiments
5. Theoretical Results
   - synTS and asyTS perform essentially the same as seqTS in terms of the number of evaluations.
   - When we factor time as a resource, asyTS outperforms synTS and seqTS.

   . . . with some caveats
6. Open questions/challenges

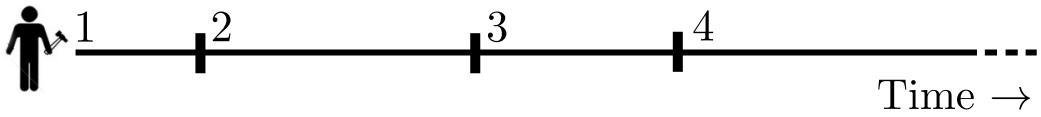
Parallel Evaluations: set up

Sequential evaluations with one worker

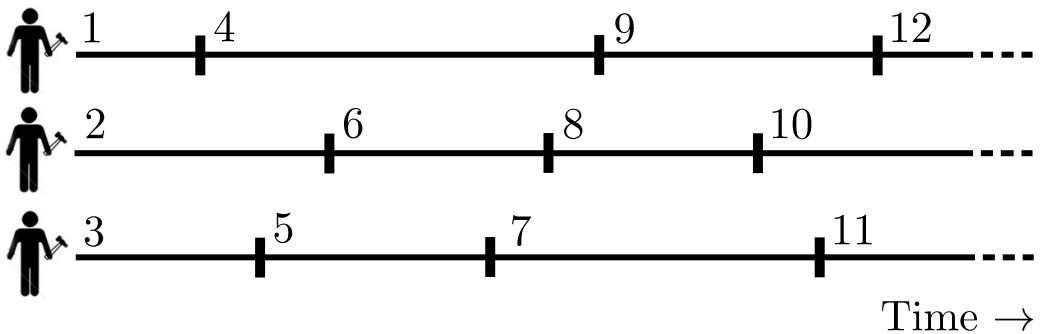


Parallel Evaluations: set up

Sequential evaluations with one worker

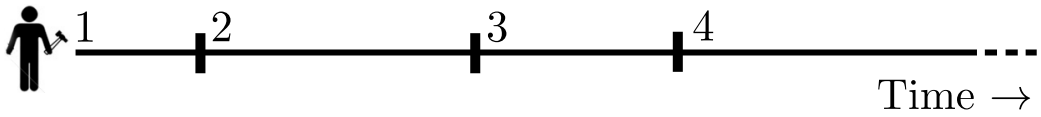


Parallel evaluations with M workers (Asynchronous)

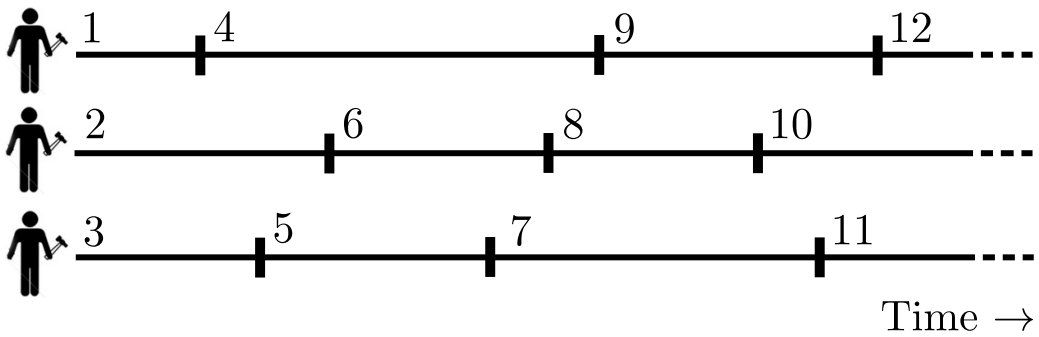


Parallel Evaluations: set up

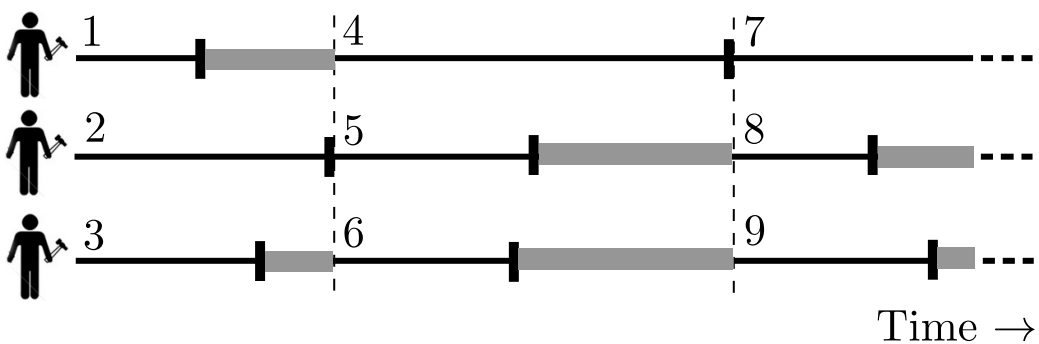
Sequential evaluations with one worker



Parallel evaluations with M workers (Asynchronous)

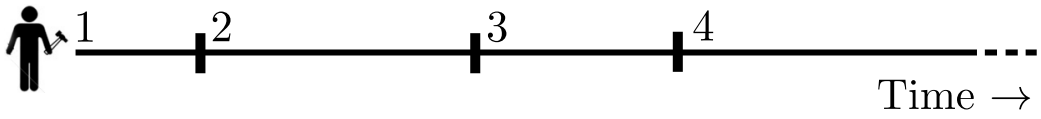


Parallel evaluations with M workers (Synchronous)



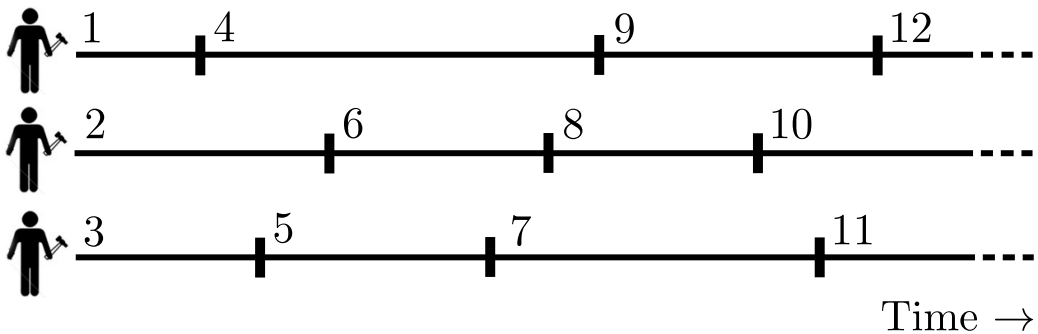
Parallel Evaluations: set up

Sequential evaluations with one worker



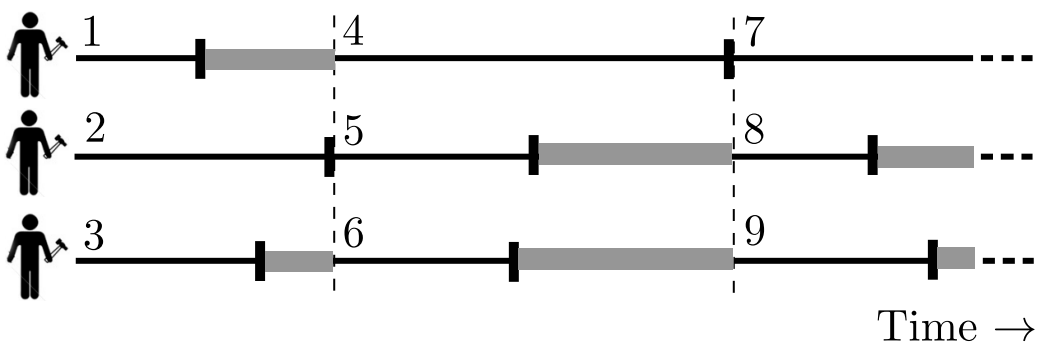
j^{th} job has feedback from all previous $j-1$ jobs.

Parallel evaluations with M workers (Asynchronous)



j^{th} job missing feedback from exactly $M-1$ jobs.

Parallel evaluations with M workers (Synchronous)



j^{th} job missing feedback from $\leq M-1$ jobs.

Simple Regret in Parallel Settings (Kandasamy et al. Arxiv 2017)

Simple regret after n evaluations,

$$\mathrm{SR}(n) = f(x_\star) - \max_{t=1,\ldots,n} f(x_t).$$

$n \leftarrow$ number of completed evaluations by all M workers.

Simple Regret in Parallel Settings (Kandasamy et al. Arxiv 2017)

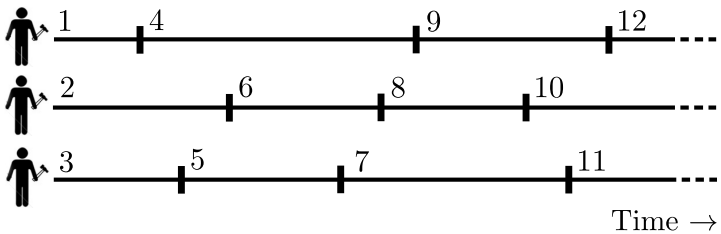
Simple regret after n evaluations,

$$\mathrm{SR}(n) = f(x_\star) - \max_{t=1,\ldots,n} f(x_t).$$

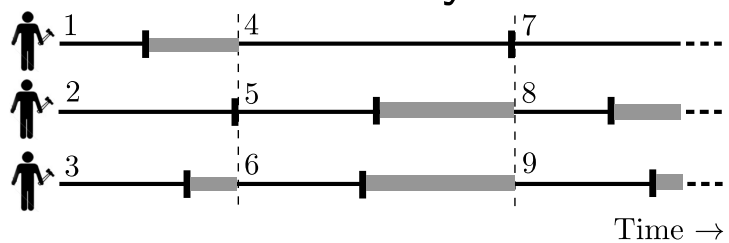
$n \leftarrow$ number of completed evaluations by all M workers.

Simple regret with time as a resource,

Asynchronous



Synchronous



$$\mathrm{SR}'(T) = f(x_\star) - \max_{t=1,\ldots,N} f(x_t).$$

$N \leftarrow$ (possibly random) number of completed evaluations by all M workers within time T.

Outline

(Kandasamy et al. Arxiv 2017)

1. Set up & definitions
2. Prior work & challenges
3. **Algorithms** synTS, asyTS: direct application of TS to synchronous and asynchronous parallel settings
4. Experiments
5. Theoretical Results
   - synTS and asyTS perform essentially the same as seqTS in terms of the number of evaluations.
   - When we factor time as a resource, asyTS outperforms synTS and seqTS.

   ... with some caveats
6. Open questions/challenges

Prior work in Parallel BO

(Ginsbourger et al. 2011)			
(Janusevkis et al. 2012)			
(Contal et al. 2013)			
(Desautels et al. 2014)			
(Gonzalez et al. 2015)			
(Shah & Ghahramani. 2015)			
(Wang et al. 2016)			
(Kathuria et al. 2016)			
(Wu & Frazier. 2017)			
(Wang et al. 2017)			
(Kandasamy et al. Arxiv 2017)			

Prior work in Parallel BO

	Asynchr-onicity		
(Ginsbourger et al. 2011)	✓		
(Janusevkis et al. 2012)	✓		
(Contal et al. 2013)			
(Desautels et al. 2014)			
(Gonzalez et al. 2015)			
(Shah & Ghahramani. 2015)			
(Wang et al. 2016)	✓		
(Kathuria et al. 2016)			
(Wu & Frazier. 2017)			
(Wang et al. 2017)			
(Kandasamy et al. Arxiv 2017)	✓		

Prior work in Parallel BO

	Asynchr-onicity	Theoretical guarantees	
(Ginsbourger et al. 2011)	✓		
(Janusevkis et al. 2012)	✓		
(Contal et al. 2013)		✓	
(Desautels et al. 2014)		✓	
(Gonzalez et al. 2015)			
(Shah & Ghahramani. 2015)			
(Wang et al. 2016)	✓		
(Kathuria et al. 2016)		✓	
(Wu & Frazier. 2017)			
(Wang et al. 2017)			
(Kandasamy et al. Arxiv 2017)	✓	✓	

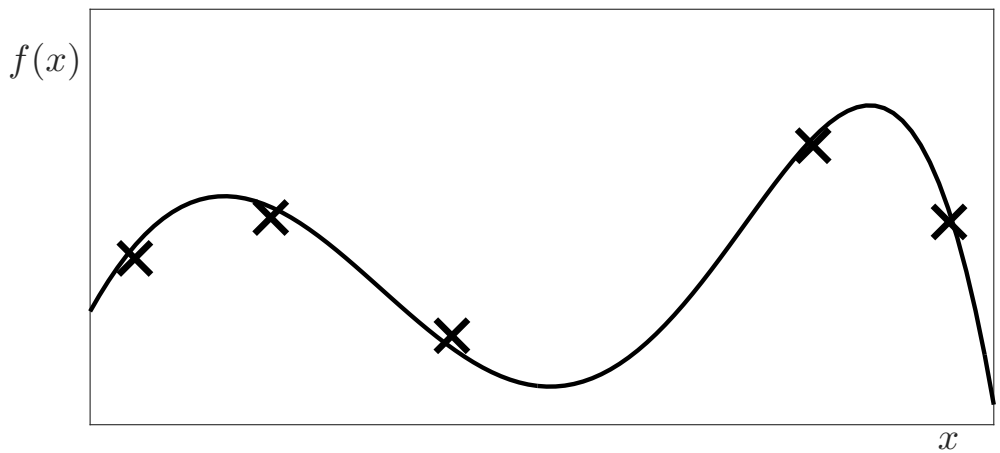
Prior work in Parallel BO

	Asynchr-onicity	Theoretical guarantees	Conceptual simplicity *
(Ginsbourger et al. 2011)	✓		
(Janusevkis et al. 2012)	✓		
(Contal et al. 2013)		✓	
(Desautels et al. 2014)		✓	
(Gonzalez et al. 2015)			
(Shah & Ghahramani. 2015)			
(Wang et al. 2016)	✓		
(Kathuria et al. 2016)		✓	
(Wu & Frazier. 2017)			
(Wang et al. 2017)			
(Kandasamy et al. Arxiv 2017)	✓	✓	✓

* straightforward extension of sequential algorithm works.

Why are deterministic algorithms not "simple"?
Need to encourage diversity in parallel evaluations

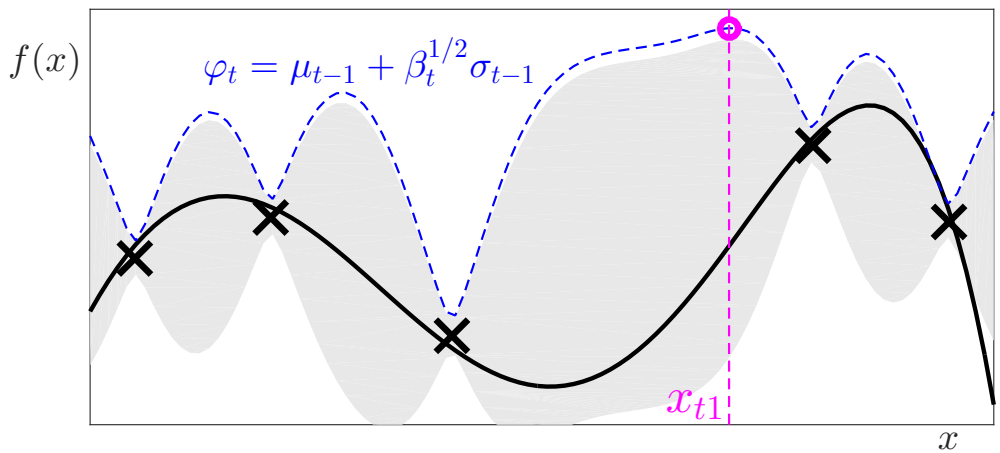
Direct application of GP-UCB in the synchronous setting ...



Why are deterministic algorithms not "simple"?
Need to encourage diversity in parallel evaluations

Direct application of GP-UCB in the synchronous setting ...

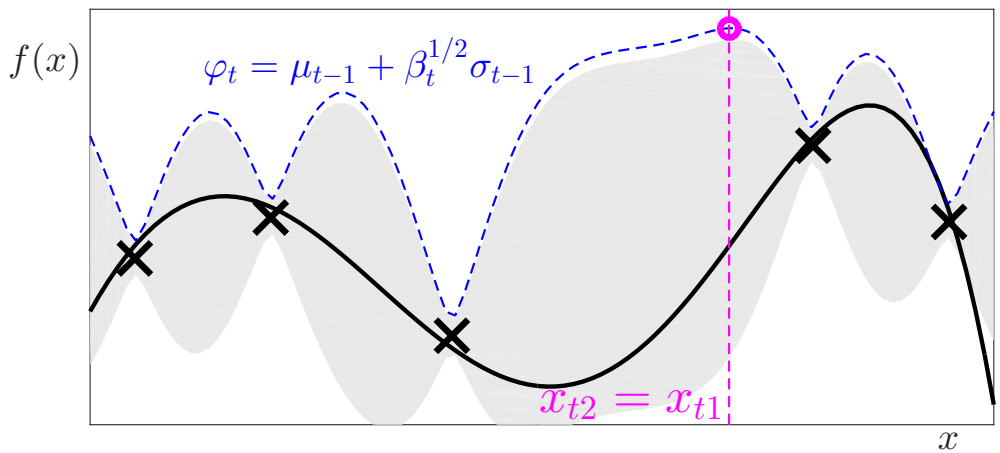
- First worker: maximise acquisition, $x_{t1} = \operatorname{argmax} \varphi_t(x)$.



Why are deterministic algorithms not "simple"?
Need to encourage diversity in parallel evaluations

Direct application of GP-UCB in the synchronous setting ...

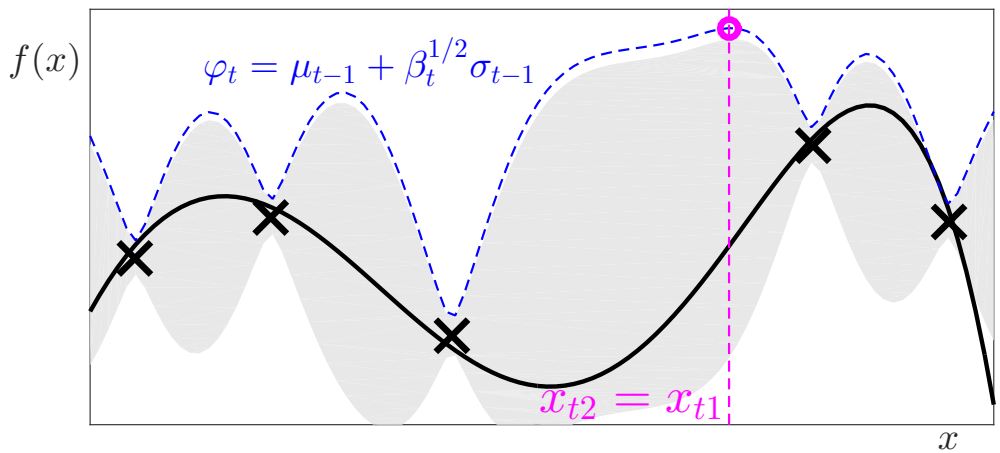
- First worker: maximise acquisition, $x_{t1} = \operatorname{argmax} \varphi_t(x)$.
- Second worker: acquisition is the same! $x_{t1} = x_{t2}$



Why are deterministic algorithms not "simple"?
Need to encourage diversity in parallel evaluations

Direct application of GP-UCB in the synchronous setting ...

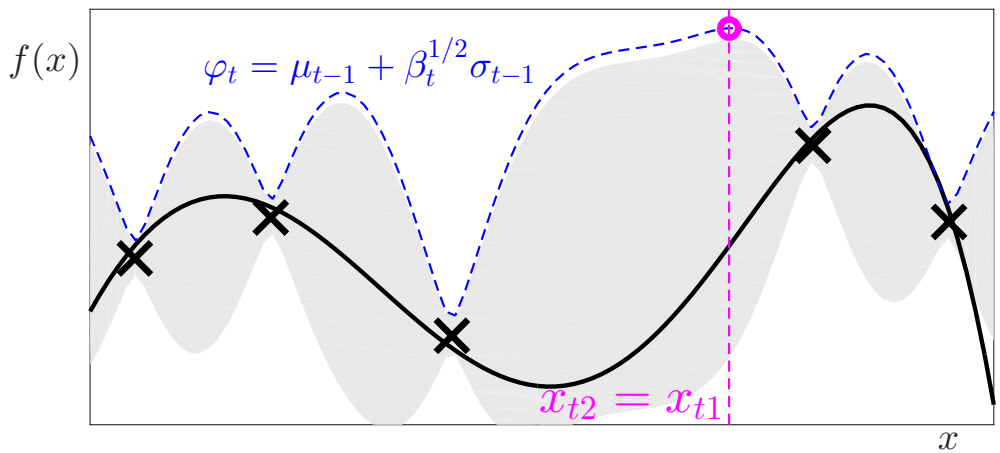
- First worker: maximise acquisition, $x_{t1} = \operatorname{argmax} \varphi_t(x)$.
- Second worker: acquisition is the same! $x_{t1} = x_{t2}$
- $x_{t1} = x_{t2} = \cdots = x_{tM}$.



Why are deterministic algorithms not "simple"?
Need to encourage diversity in parallel evaluations

Direct application of GP-UCB in the synchronous setting ...

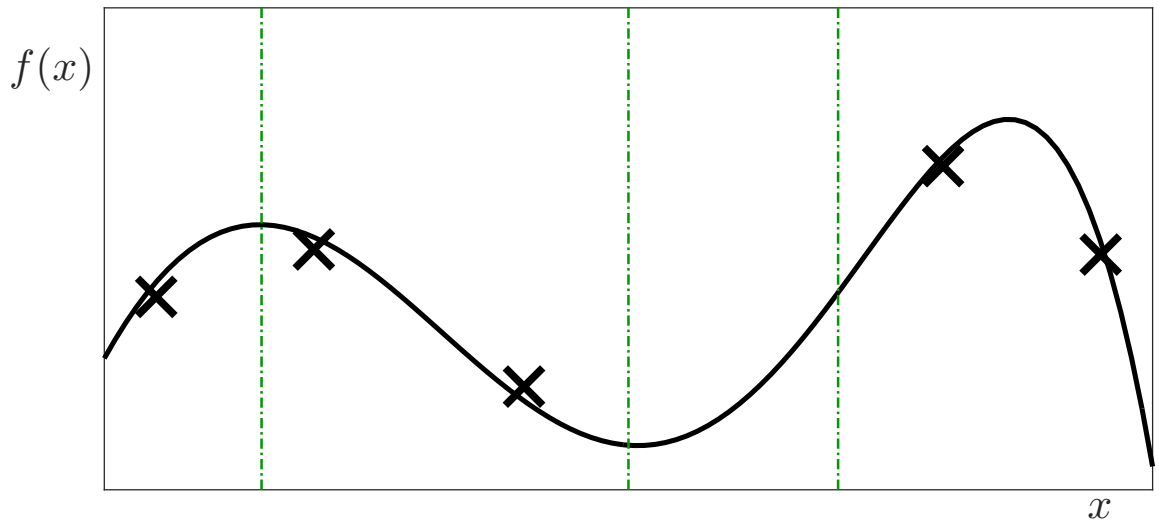
- First worker: maximise acquisition, $x_{t1} = \operatorname{argmax} \varphi_t(x)$.
- Second worker: acquisition is the same! $x_{t1} = x_{t2}$
- $x_{t1} = x_{t2} = \cdots = x_{tM}$.



Direct application of sequential algorithm does not work.
Need to "encourage diversity".

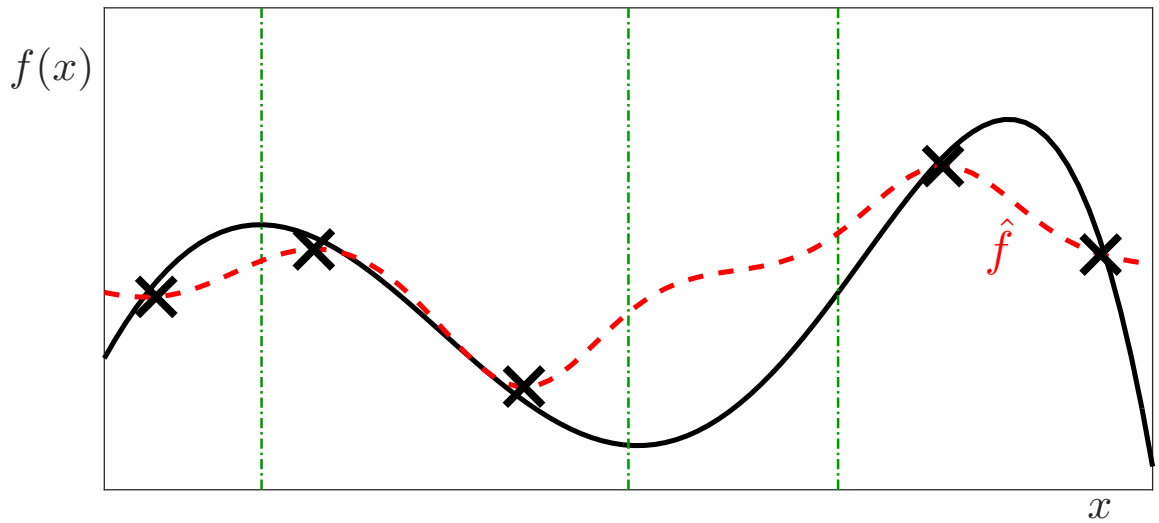
Why are deterministic algorithms not "simple"?
Need to encourage diversity in parallel evaluations

- Add hallucinated observations.



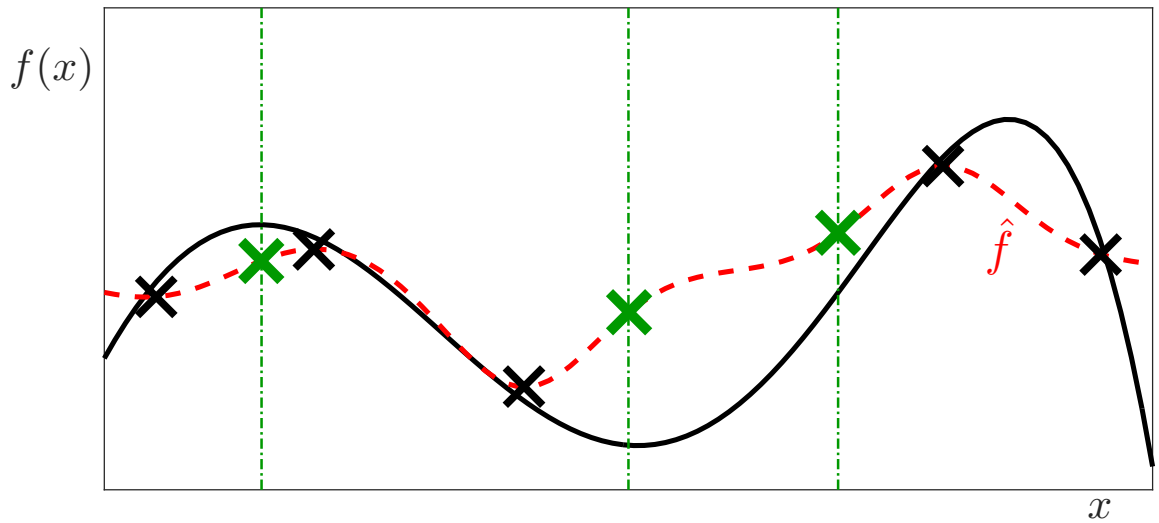
Why are deterministic algorithms not "simple"?
Need to encourage diversity in parallel evaluations

- Add hallucinated observations.



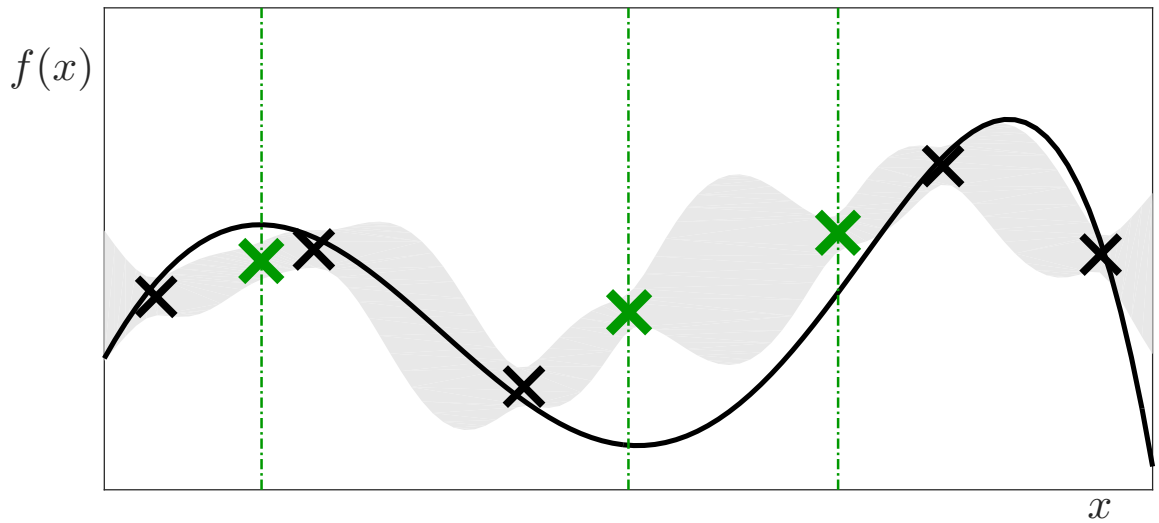
Why are deterministic algorithms not "simple"?
Need to encourage diversity in parallel evaluations

- Add hallucinated observations.



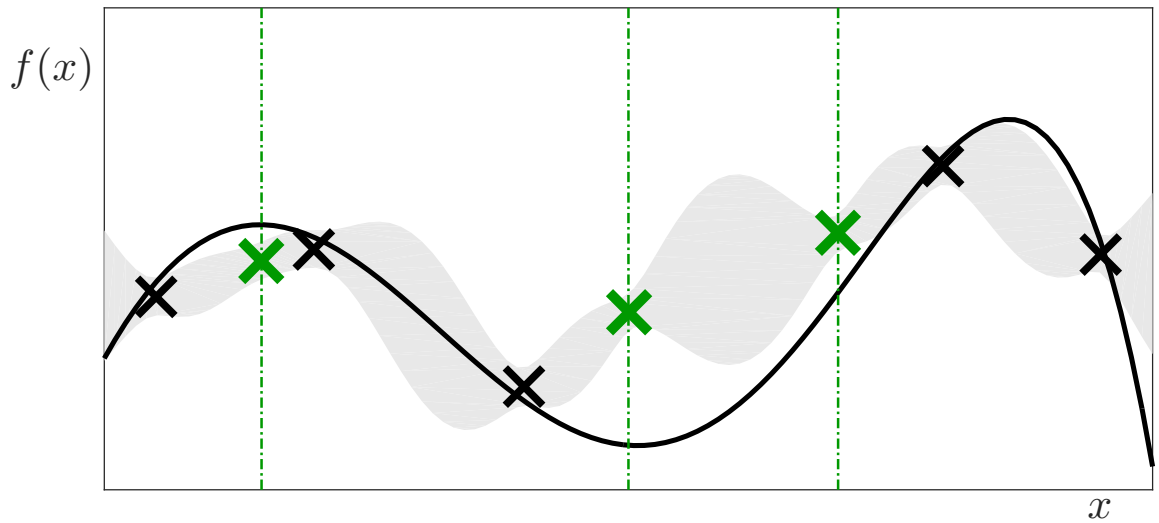
Why are deterministic algorithms not "simple"?
Need to encourage diversity in parallel evaluations

- Add hallucinated observations.



Why are deterministic algorithms not "simple"?
Need to encourage diversity in parallel evaluations

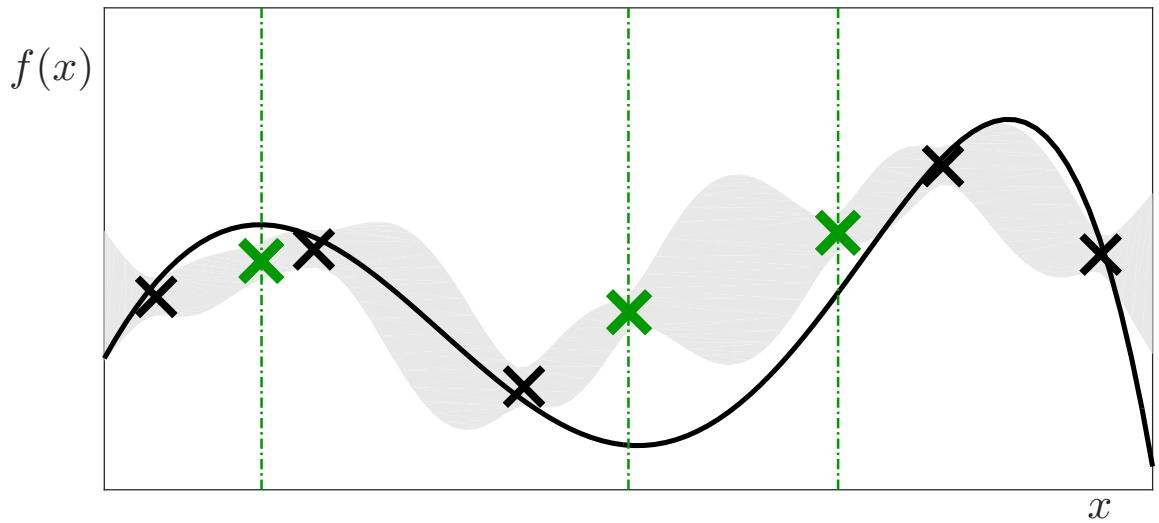
- Add hallucinated observations.



- Optimise an acquisition over $\mathcal{X}^M$.

Why are deterministic algorithms not "simple"?
Need to encourage diversity in parallel evaluations

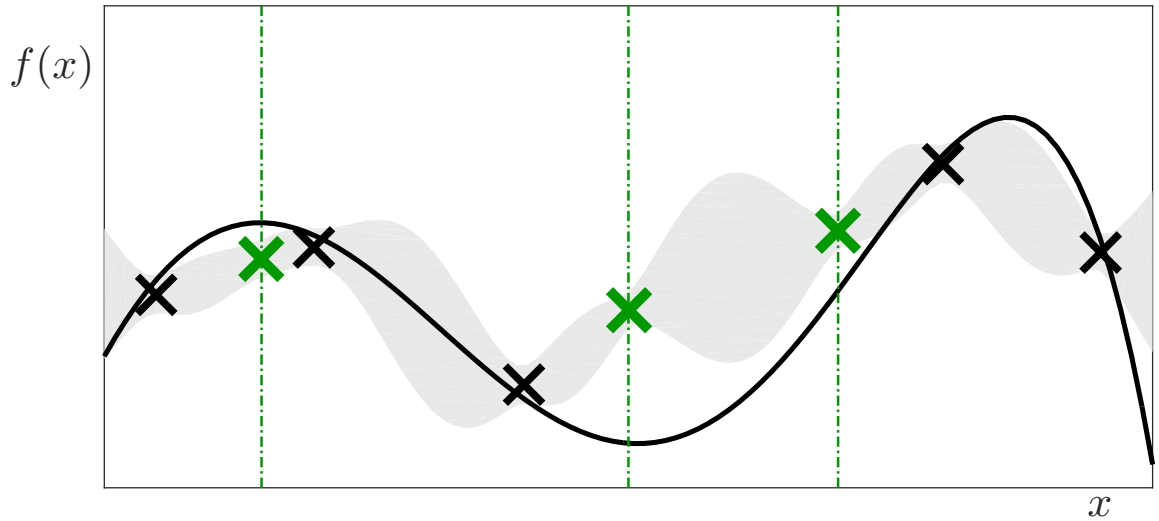
- Add hallucinated observations.



- Optimise an acquisition over $\mathcal{X}^M$.
- Resort to heuristics, typically requires additional hyper-parameters and/or computational routines.

Why are deterministic algorithms not "simple"?
Need to encourage diversity in parallel evaluations

- Add hallucinated observations.



- Optimise an acquisition over $\mathcal{X}^M$.
- Resort to heuristics, typically requires additional hyper-parameters and/or computational routines.

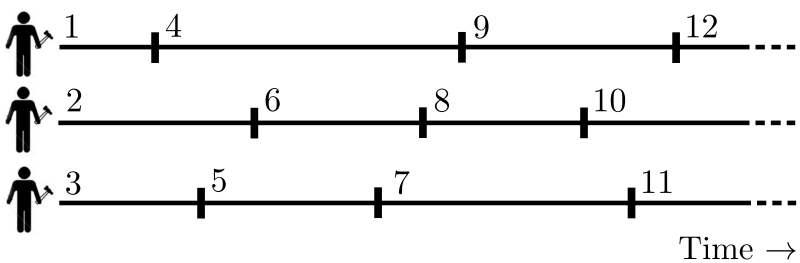
Take-home message: Straightforward application of sequential algorithm works for TS. Inherent randomness takes care of exploration vs. exploitation trade-off when managing M workers.

Parallel Thompson Sampling (Kandasamy et al. Arxiv 2017)

Asynchronous: asyTS

At any given time,

1. $(x', y') \leftarrow$ Wait for a worker to finish.
2. Compute posterior $\mathcal{GP}$.
3. Draw a sample $g \sim \mathcal{GP}$.
4. Re-deploy worker at $\operatorname{argmax} g$.

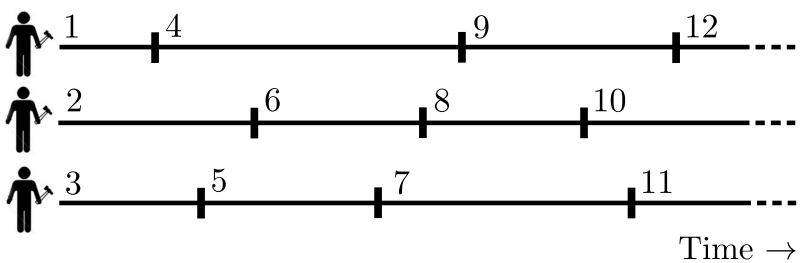


Parallel Thompson Sampling (Kandasamy et al. Arxiv 2017)

Asynchronous: asyTS

At any given time,

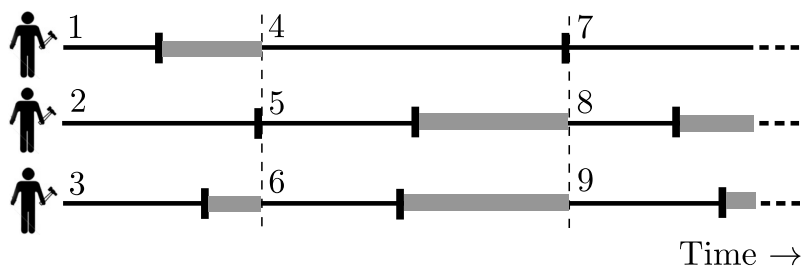
1. $(x', y') \leftarrow$ Wait for a worker to finish.
2. Compute posterior $\mathcal{GP}$.
3. Draw a sample $g \sim \mathcal{GP}$.
4. Re-deploy worker at $\operatorname{argmax} g$.



Synchronous: synTS

At any given time,

1. $\{(x'_m, y'_m)\}_{m=1}^{M} \leftarrow$ Wait for all workers to finish.
2. Compute posterior $\mathcal{GP}$.
3. Draw M samples $g_m \sim \mathcal{GP}$, $\forall m$.
4. Re-deploy worker m at $\operatorname{argmax} g_m$, $\forall m$.



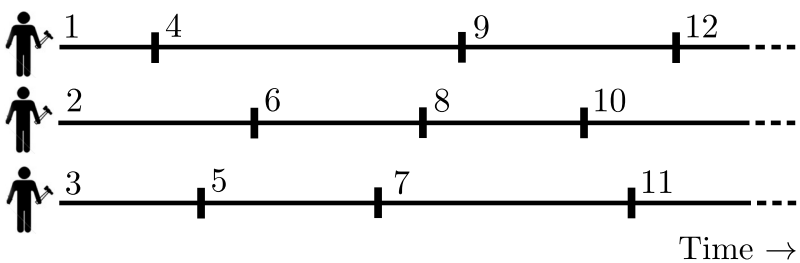
Parallel Thompson Sampling

(Kandasamy et al. Arxiv 2017)

Asynchronous: asyTS

At any given time,

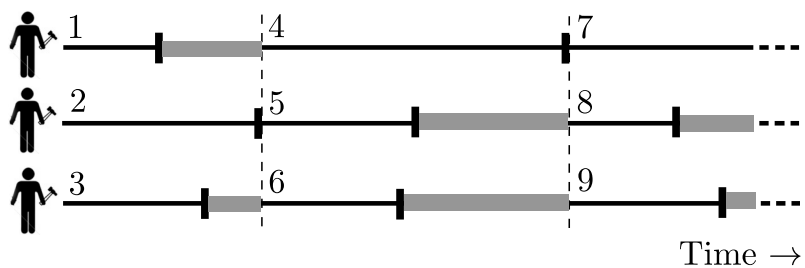
1. $(x', y') \leftarrow$ Wait for a worker to finish.
2. Compute posterior $\mathcal{GP}$.
3. Draw a sample $g \sim \mathcal{GP}$.
4. Re-deploy worker at $\operatorname{argmax} g$.



Synchronous: synTS

At any given time,

1. $\{(x'_m, y'_m)\}_{m=1}^{M} \leftarrow$ Wait for all workers to finish.
2. Compute posterior $\mathcal{GP}$.
3. Draw M samples $g_m \sim \mathcal{GP}$, $\forall m$.
4. Re-deploy worker m at $\operatorname{argmax} g_m$, $\forall m$.



Variants in prior work: (Osband et al. 2016, Israelsen et al. 2016, Hernandez-Lobato et al. 2017)

Outline

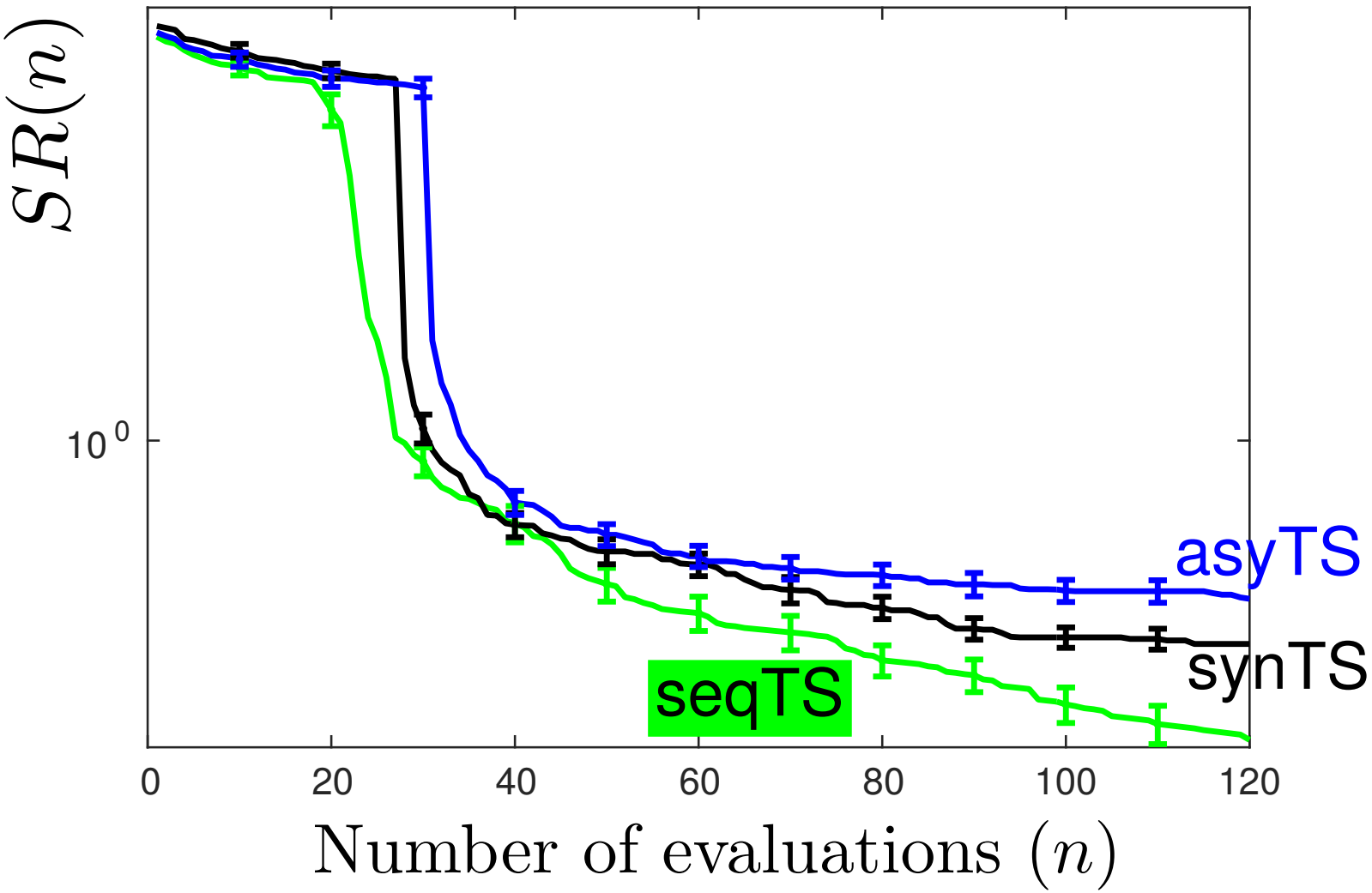
(Kandasamy et al. Arxiv 2017)

1. Set up & definitions
2. Prior work & challenges
3. **Algorithms** synTS, asyTS: direct application of TS to synchronous and asynchronous parallel settings
4. Experiments
5. Theoretical Results
   - synTS and asyTS perform essentially the same as seqTS in terms of the number of evaluations.
   - When we factor time as a resource, asyTS outperforms synTS and seqTS.

   . . . with some caveats
6. Open questions/challenges

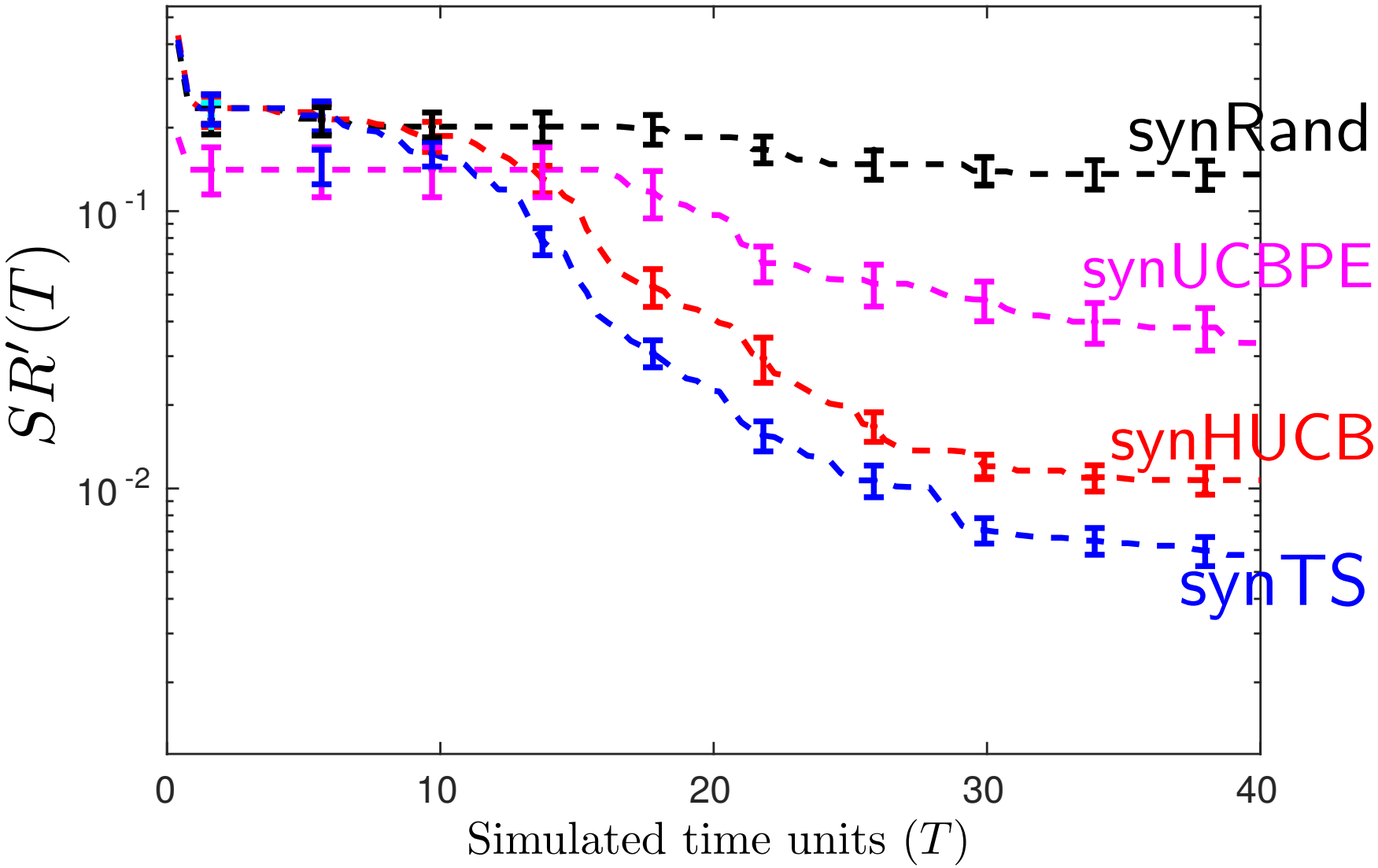
Experiment: Park1-4D $M = 10$

Comparison in terms of number of evaluations



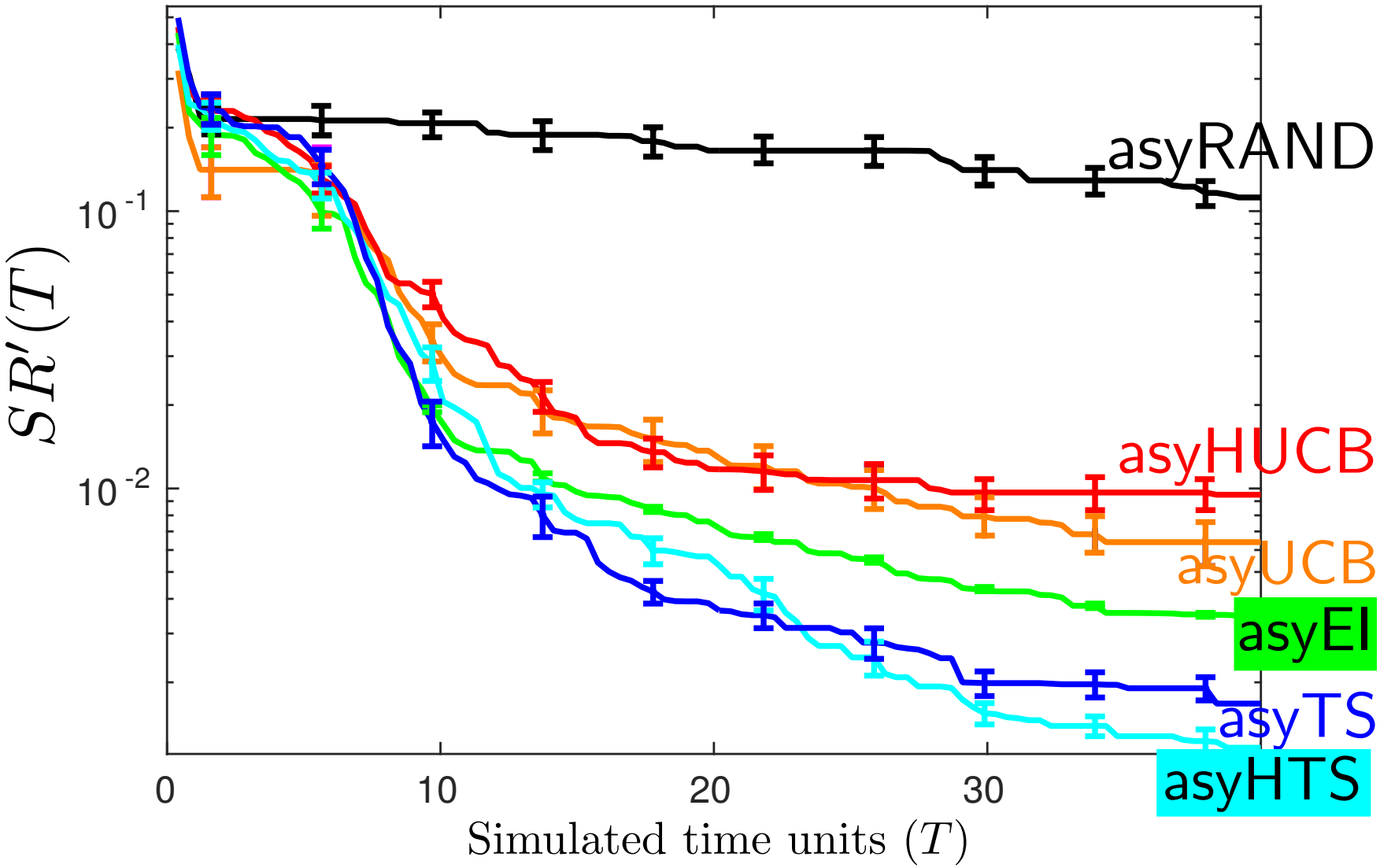
Experiment: Branin-2D $M = 4$

Evaluation time sampled from a uniform distribution



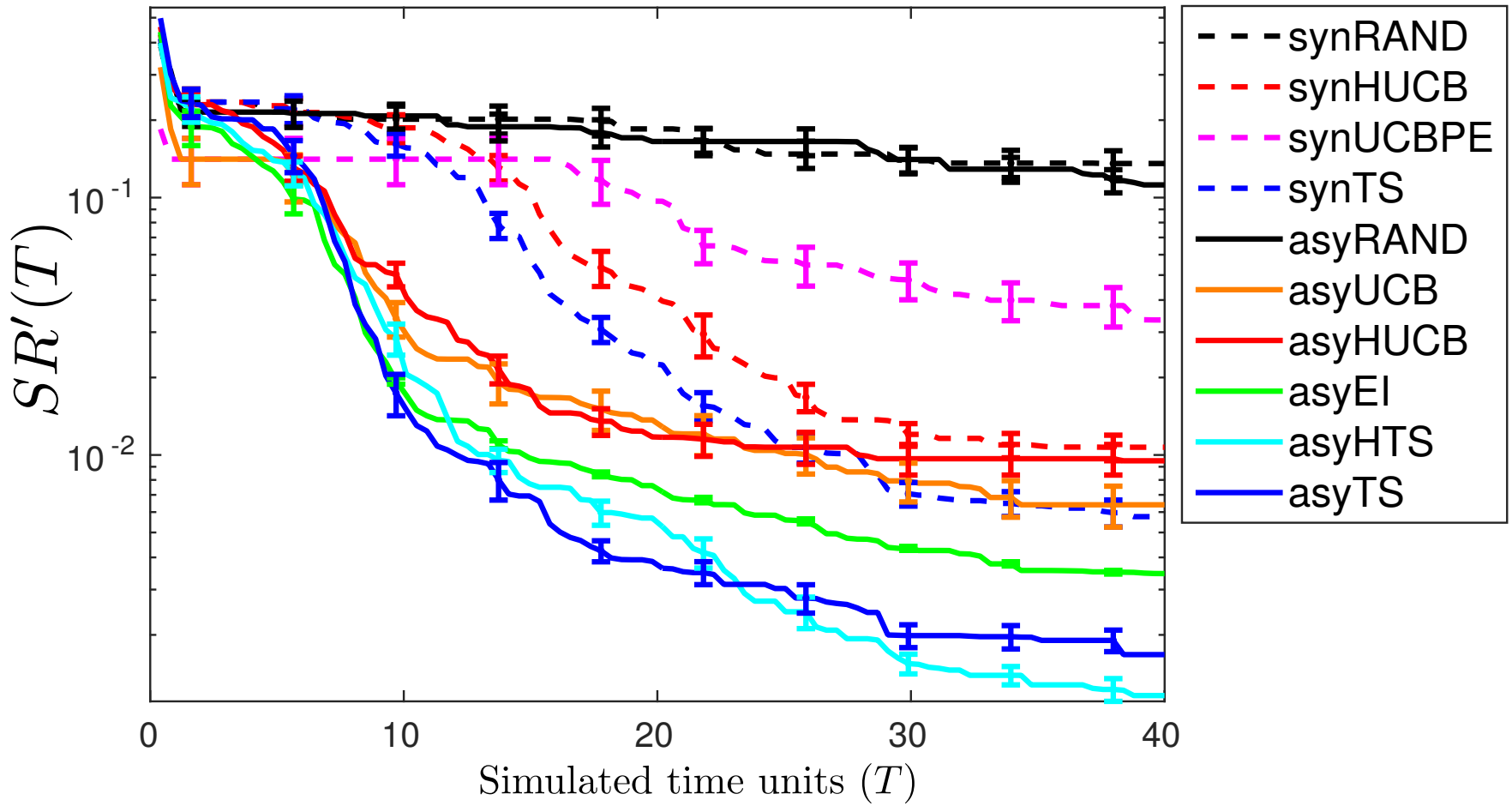
Experiment: Branin-2D $M = 4$

Evaluation time sampled from a uniform distribution



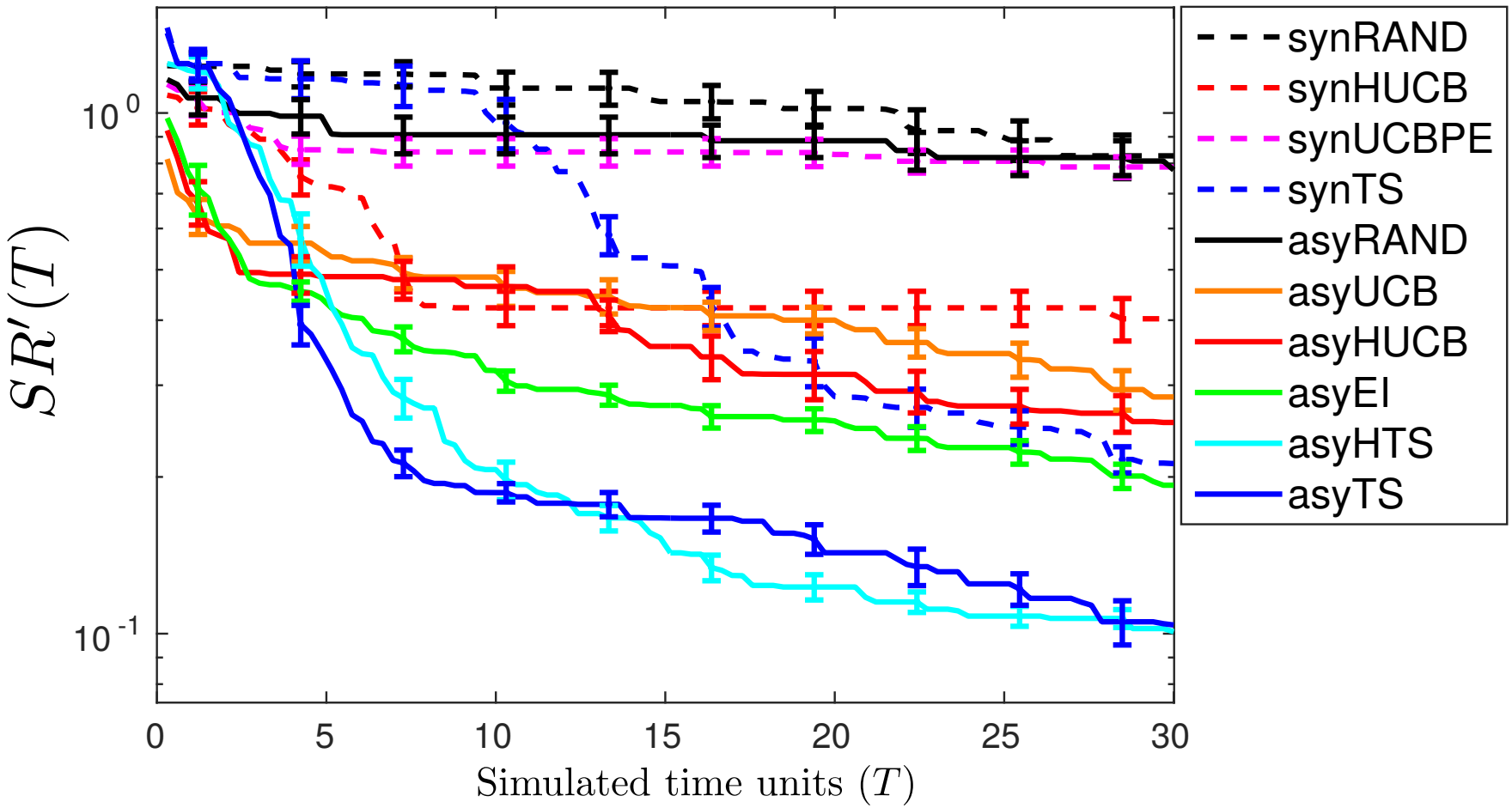
Experiment: Branin-2D $M = 4$

Evaluation time sampled from a uniform distribution



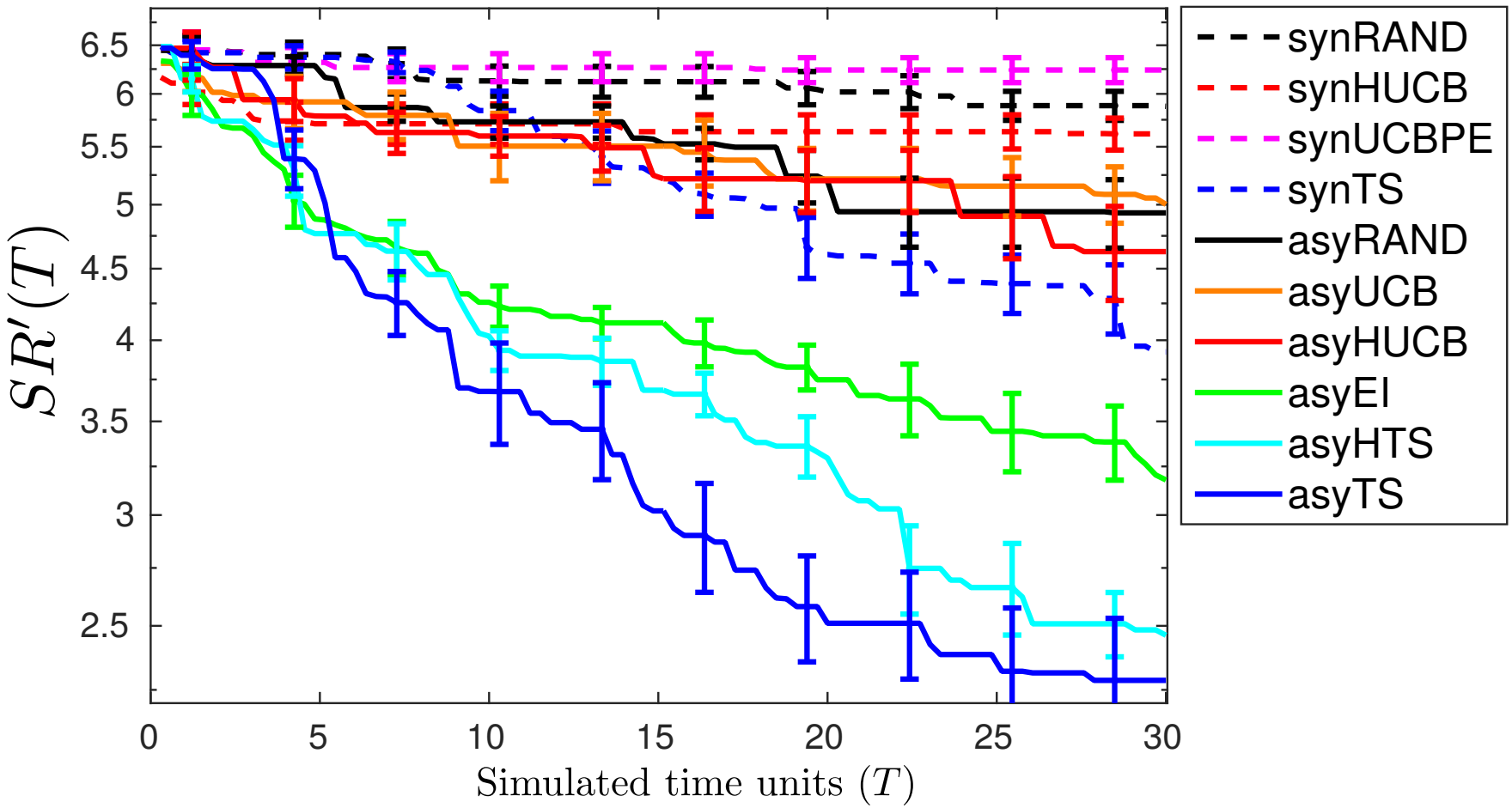
Experiment: Hartmann-6D $M = 12$

Evaluation time sampled from a half-normal distribution



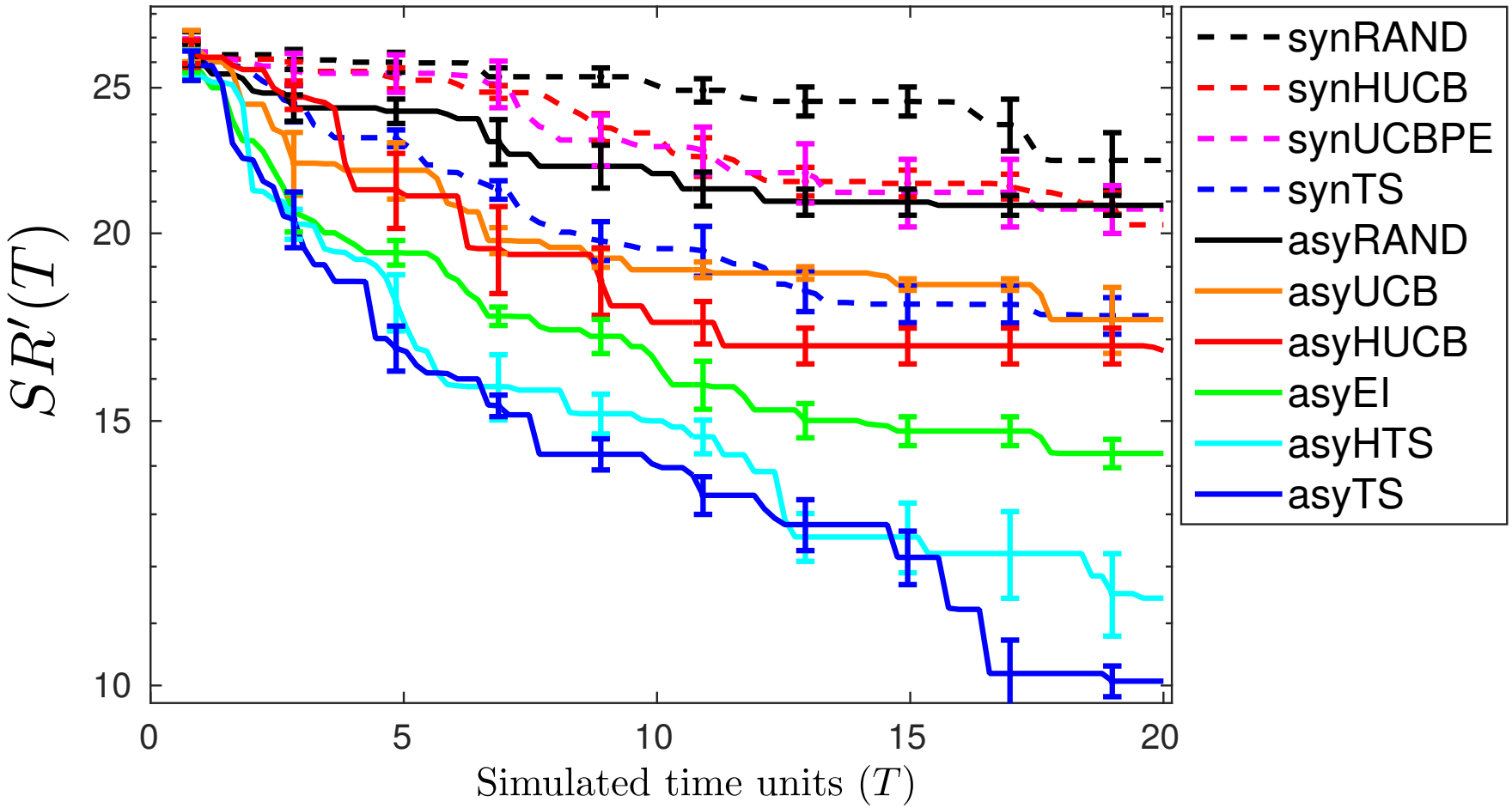
Experiment: Hartmann-18D $M = 25$

Evaluation time sampled from an exponential distribution



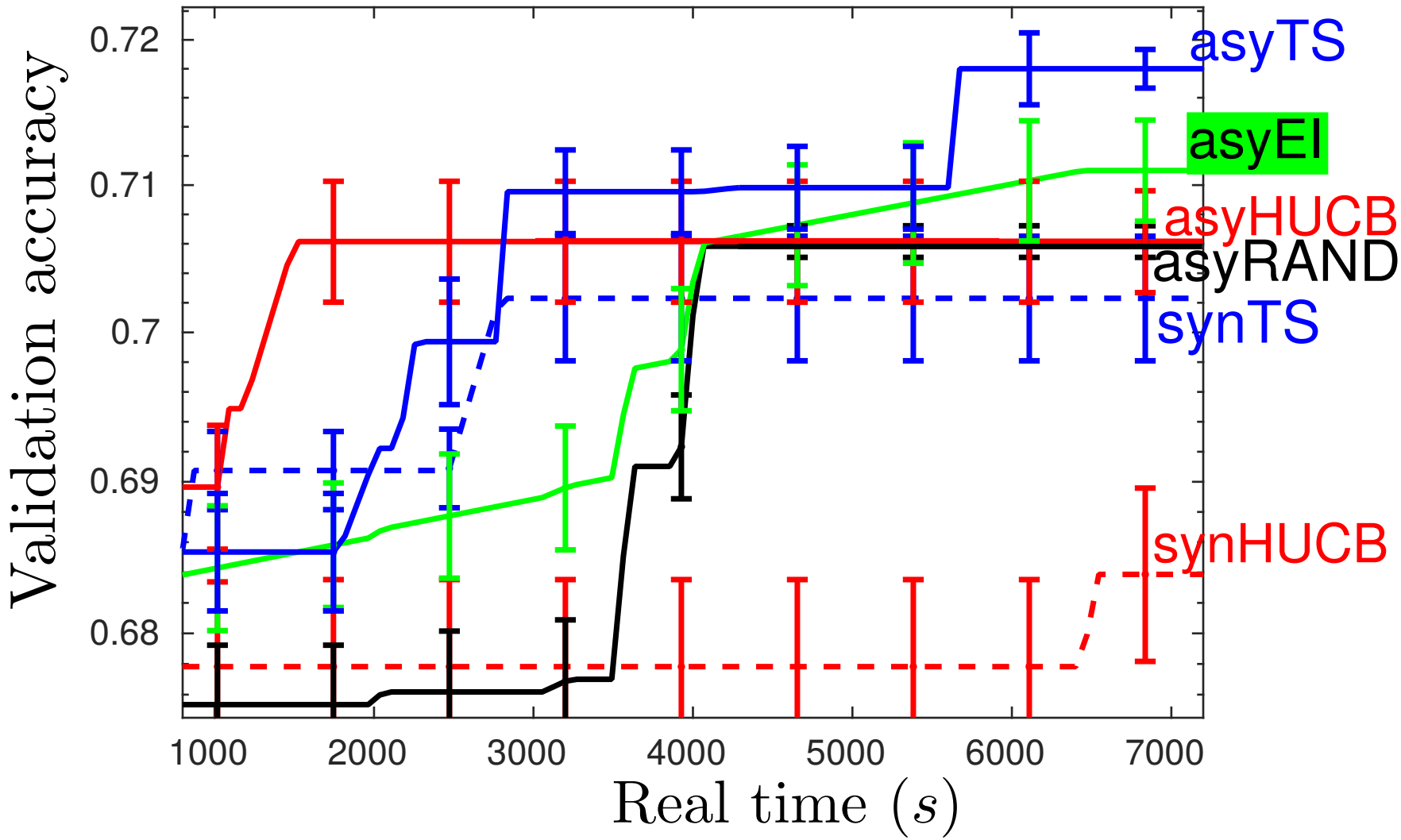
Experiment: Currin-Exponential-14D $M = 35$

Evaluation time sampled from a Pareto-3 distribution



Experiment: Model Selection in Cifar10 $M = 4$

Tune # filters in in range $(32, 256)$ for each layer in a 6 layer CNN.
Time taken for an evaluation: 4 - 16 minutes.



Outline

(Kandasamy et al. Arxiv 2017)

1. Set up & definitions
2. Prior work & challenges
3. **Algorithms** synTS, asyTS: direct application of TS to synchronous and asynchronous parallel settings
4. Experiments
5. Theoretical Results
   - synTS and asyTS perform essentially the same as seqTS in terms of the number of evaluations.
   - When we factor time as a resource, asyTS outperforms synTS and seqTS.

   ... with some caveats.
6. Open questions/challenges

Bounds for SR(n), synTS

seqTS (Russo & van Roy 2014)

$$\mathbb{E}[\mathrm{SR}(n)] \lesssim \sqrt{\frac{\Psi_n \log(n)}{n}}$$

$\Psi_n \leftarrow$ Maximum information gain.

Bounds for SR(n), synTS

seqTS (Russo & van Roy 2014)

$$\mathbb{E}[\mathrm{SR}(n)] \lesssim \sqrt{\frac{\Psi_n \log(n)}{n}}$$

$\Psi_n \leftarrow$ Maximum information gain.

Theorem: synTS (Kandasamy et al. Arxiv 2017)

$$\mathbb{E}[\mathrm{SR}(n)] \lesssim \frac{M\sqrt{\log(M)}}{n} + \sqrt{\frac{\Psi_{n+M} \log(n+M)}{n}}$$

Leading constant is also the same.

Bounds for SR(n), asyTS

seqTS (Russo & van Roy 2014)

$$\mathbb{E}[\mathrm{SR}(n)] \lesssim \sqrt{\frac{\Psi_n \log(n)}{n}}$$

Bounds for SR(n), asyTS

seqTS (Russo & van Roy 2014)

$$\mathbb{E}[\mathrm{SR}(n)] \lesssim \sqrt{\frac{\Psi_n \log(n)}{n}}$$

Theorem: asyTS (Kandasamy et al. Arxiv 2017)

$$\mathbb{E}[\mathrm{SR}(n)] \lesssim \sqrt{\frac{\xi_M \Psi_n \log(n)}{n}}$$

$\xi_M = \sup_{\mathcal{D}_n, n \geq 1} \max_{A \subset \mathcal{X}, |A| \leq M} e^{I(f; A | \mathcal{D}_n)}$.

Bounds for SR(n), asyTS

seqTS (Russo & van Roy 2014)

$$\mathbb{E}[\mathrm{SR}(n)] \lesssim \sqrt{\frac{\Psi_n \log(n)}{n}}$$

Theorem: asyTS (Kandasamy et al. Arxiv 2017)

$$\mathbb{E}[\mathrm{SR}(n)] \lesssim \sqrt{\frac{\xi_M \Psi_n \log(n)}{n}}$$

$\xi_M = \sup_{\mathcal{D}_n, n \geq 1} \max_{A \subset \mathcal{X}, |A| \leq M} e^{I(f; A | \mathcal{D}_n)}$.

Theorem: There exists an asynchronously parallelisable initialisation scheme requiring $\mathcal{O}(M\mathrm{polylog}(M))$ evaluations to f such that $\xi_M \leq C$. (Krause et al. 2008, Desautels et al. 2012)

Bounds for SR(n), asyTS

seqTS (Russo & van Roy 2014)

$$\mathbb{E}[\mathrm{SR}(n)] \lesssim \sqrt{\frac{\Psi_n \log(n)}{n}}$$

Theorem: asyTS, arbitrary $\mathcal{X}$ (Kandasamy et al. Arxiv 2017)

$$\mathbb{E}[\mathrm{SR}(n)] \lesssim \frac{M\mathrm{polylog}(M)}{n} + \sqrt{\frac{C\Psi_n \log(n)}{n}}$$

$\xi_M = \sup_{\mathcal{D}_n, n \geq 1} \max_{A \subset \mathcal{X}, |A| \leq M} e^{I(f;A|\mathcal{D}_n)}$.

Theorem: There exists an asynchronously parallelisable initialisation scheme requiring $\mathcal{O}(M\mathrm{polylog}(M))$ evaluations to f such that $\xi_M \leq C$. (Krause et al. 2008, Desautels et al. 2012)

Bounds for SR(n), asyTS

seqTS (Russo & van Roy 2014)

$$\mathbb{E}[\mathrm{SR}(n)] \lesssim \sqrt{\frac{\Psi_n \log(n)}{n}}$$

Theorem: asyTS, arbitrary $\mathcal{X}$ (Kandasamy et al. Arxiv 2017)

$$\mathbb{E}[\mathrm{SR}(n)] \lesssim \frac{M\mathrm{polylog}(M)}{n} + \sqrt{\frac{C\Psi_n \log(n)}{n}}$$

$\xi_M = \sup_{\mathcal{D}_n, n \geq 1} \max_{A \subset \mathcal{X}, |A| \leq M} e^{I(f;A|\mathcal{D}_n)}$.

Theorem: There exists an asynchronously parallelisable initialisation scheme requiring $\mathcal{O}(M\mathrm{polylog}(M))$ evaluations to f such that $\xi_M \leq C$. (Krause et al. 2008, Desautels et al. 2012)

* We do not believe this is necessary.

Bounds for asyTS without the initialisation scheme

Theorem: synTS, arbitrary $\mathcal{X}$ (Kandasamy et al. Arxiv 2017)

$$\mathbb{E}[\mathrm{SR}(n)] \lesssim \frac{M\sqrt{\log(M)}}{n} + \sqrt{\frac{\Psi_{n+M}\log(n+M)}{n}}$$

Bounds for asyTS without the initialisation scheme

Theorem: synTS, arbitrary $\mathcal{X}$ (Kandasamy et al. Arxiv 2017)

$$\mathbb{E}[\mathrm{SR}(n)] \lesssim \frac{M\sqrt{\log(M)}}{n} + \sqrt{\frac{\Psi_{n+M}\log(n+M)}{n}}$$

Theorem: asyTS, $\mathcal{X} \subset \mathbb{R}^d$ (Ongoing work)

$$\mathbb{E}[\mathrm{SR}(n)] \lesssim \ldots + \frac{\sqrt{M\log(n)}}{n^{1/\mathcal{O}(d)}}$$

Theoretical Results for $\mathrm{SR}'(T)$

Model evaluation time as an independent random variable

- Uniform $\quad \mathrm{unif}(a, b)$ bounded
- Half-normal $\quad \mathcal{HN}(\tau^2)$ sub-Gaussian
- Exponential $\quad \exp(\lambda)$ sub-exponential

Theoretical Results for $\mathrm{SR}'(T)$

Model evaluation time as an independent random variable

- Uniform $\quad \mathrm{unif}(a, b)$ bounded
- Half-normal $\quad \mathcal{HN}(\tau^2)$ sub-Gaussian
- Exponential $\quad \exp(\lambda)$ sub-exponential

Theorem (Informal):
If evaluation times are the same, synTS$\approx$ asyTS.

Otherwise, bounds for asyTS are better than synTS. More the variability in evaluation times, the bigger the difference.

Theoretical Results for $\mathrm{SR}'(T)$

Model evaluation time as an independent random variable

- Uniform $\quad \mathrm{unif}(a, b)$ bounded
- Half-normal $\quad \mathcal{HN}(\tau^2)$ sub-Gaussian
- Exponential $\quad \exp(\lambda)$ sub-exponential

Theorem (Informal):
If evaluation times are the same, synTS$\approx$ asyTS.

Otherwise, bounds for asyTS are better than synTS. More the variability in evaluation times, the bigger the difference.

- Uniform: constant factor
- Half-normal: $\sqrt{\log(M)}$ factor
- Exponential: $\log(M)$ factor

Outline

(Kandasamy et al. Arxiv 2017)

1. Set up & definitions
2. Prior work & challenges
3. **Algorithms** synTS, asyTS: direct application of TS to synchronous and asynchronous parallel settings
4. Experiments
5. Theoretical Results
   - synTS and asyTS perform essentially the same as seqTS in terms of the number of evaluations.
   - When we factor time as a resource, asyTS outperforms synTS and seqTS.

   . . . with some caveats
6. Open questions/challenges

Open Challenges for Parallelised TS

1. Bounds for asynchronous TS without initialisation.

2. Other models for evaluation times.
   - e.g. evaluation time depends on $x \in \mathcal{X}$.

Open Challenges for Parallelised TS

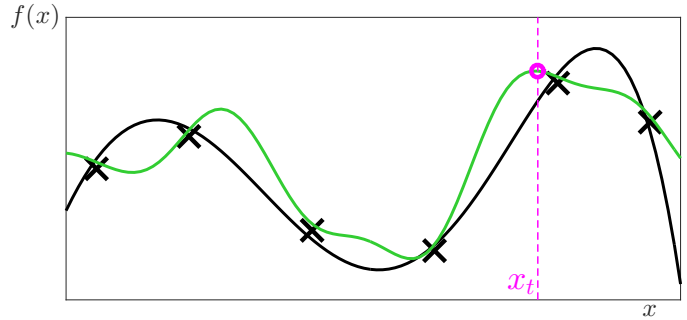
1. Bounds for asynchronous TS without initialisation.

2. Other models for evaluation times.
   - e.g. evaluation time depends on $x \in \mathcal{X}$.

3. In the asynchronous setting,
   - Should you wait for another job to finish without immediately re-deploying?
   - Do you kill an on-going job depending on the result of a completed job?

Open Challenges for Parallelised TS

4. Optimising the sample when $\mathcal{X} = [0,1]^d$,



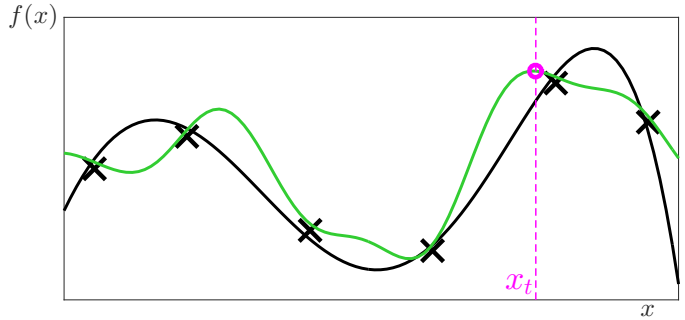
$$x_t = \underset{x \in \mathcal{X}}{\operatorname{argmax}}\, g(x), \quad \text{where } g \sim \text{Posterior } \mathcal{GP}$$

- Global optimisation of a non-convex function!

  .. a common challenge in most BO methods.

Open Challenges for Parallelised TS

4. Optimising the sample when $\mathcal{X} = [0,1]^d$,



$$x_t = \underset{x \in \mathcal{X}}{\operatorname{argmax}}\, g(x), \quad \text{where } g \sim \text{Posterior } \mathcal{GP}$$

- Global optimisation of a non-convex function!
  .. a common challenge in most BO methods.

But additionally for TS,

- As g is not deterministic, draw samples from a fixed set of points and pick the maximum.
- Or if using an adaptive method, scales $O((N+S)^3)$ where
  $N \leftarrow \#$ of evaluations to f, $\quad S \leftarrow \#$ of evaluations to g.

Summary

- synTS, asyTS: direct application of TS to synchronous and asynchronous parallel settings.

Summary

- synTS, asyTS: direct application of TS to synchronous and asynchronous parallel settings.
- Take-aways: Theory
  - Both perform essentially the same as seqTS in terms of the number of evaluations.
  - When we factor time as a resource, asyTS performs best.

Summary

- synTS, asyTS: direct application of TS to synchronous and asynchronous parallel settings.
- Take-aways: Theory
  - Both perform essentially the same as seqTS in terms of the number of evaluations.
  - When we factor time as a resource, asyTS performs best.
- Take-aways: Practice
  - Conceptually simple and scales better with the number of workers than other methods.

Akshay Krishnamurthy

Jeff Schneider

Barnabás Póczos

Code: `github.com/kirthevasank/gp-parallel-ts`

Slides: `www.cs.cmu.edu/~kkandasa/talks/google-ts-slides.pdf`

Thank you.