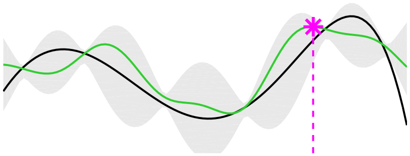
Bayesian Methods for Adaptive Experimentation



Kirthevasan Kandasamy

UC Berkeley

(Work done at Carnegie Mellon University)

Aug 9, 2019

Symposium on Autonomous Experimentation
University of Maryland, College Park, MD

Optimising Electrolyte Conductivity

$$x \longrightarrow \boxed{\textbf{Experiment}} \longrightarrow Y_x$$

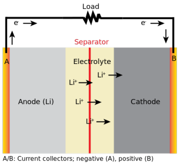
electrolyte design

solvents/salt concentrations, physical/process conditions

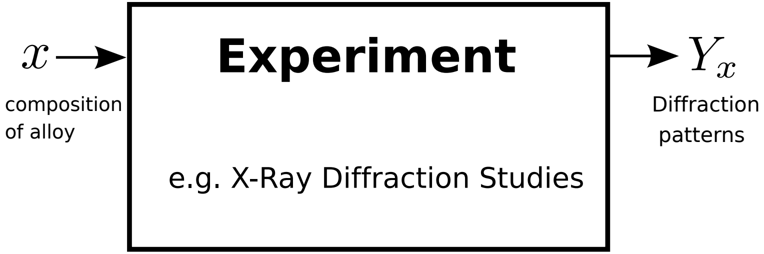
Experiment

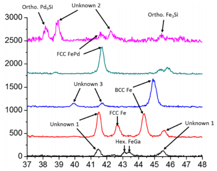
e.g. Impedance spectroscopy

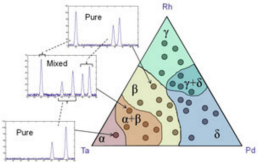
conductivity



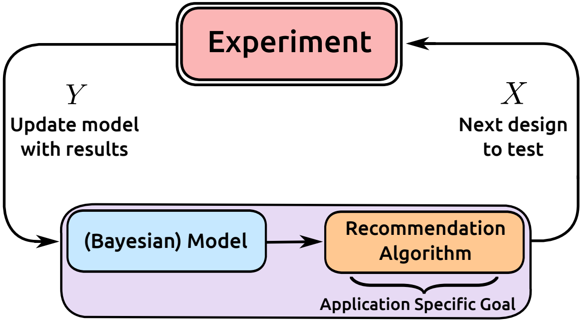
Identifying Phase Transitions in Alloys



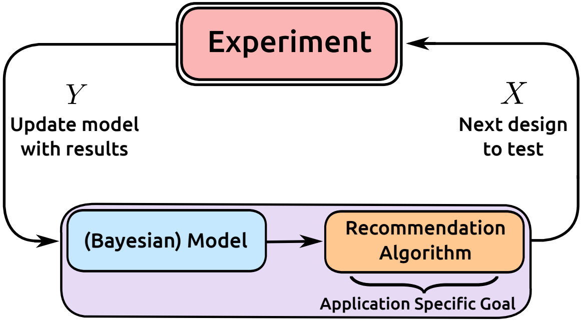




Adaptive Goal Oriented Design of Experiments



Adaptive Goal Oriented Design of Experiments



- Blackbox Optimisation
- Active Learning
- Active Quadrature (Osborne et al. 2012)
- Active Level Set Estimation (Gotovos et al. '13)
- Active Search (Ma et al. '17)
- Active Posterior Estimation (Kandasamy et al. '15)

Outline

1. Blackbox optimisation, Bayesian Models, and Bayesian Optimisation

2. New Frontiers in Bayesian Optimisation:
   Parallel evaluations, High dimensional optimisation, Multi-fidelity optimisation, Multi-objective optimisation

3. Dragonfly: An Open Source Bayesian Optimisation Implementation & Experiments

4. General Settings for Adaptive Goal Oriented Design of Experiments

Outline

1. Blackbox optimisation, Bayesian Models, and Bayesian Optimisation

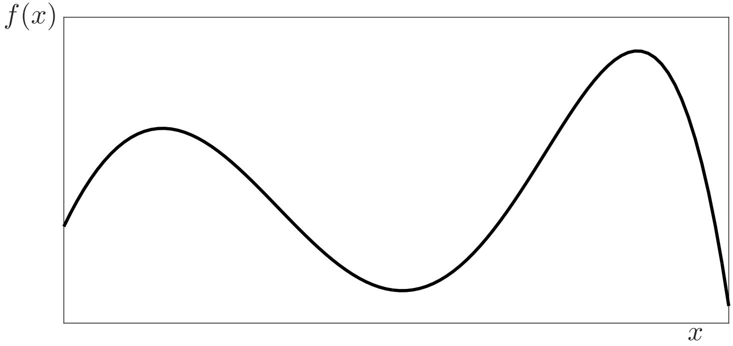
2. New Frontiers in Bayesian Optimisation:
   Parallel evaluations, High dimensional optimisation, Multi-fidelity optimisation, Multi-objective optimisation

3. Dragonfly: An Open Source Bayesian Optimisation Implementation & Experiments

4. General Settings for Adaptive Goal Oriented Design of Experiments

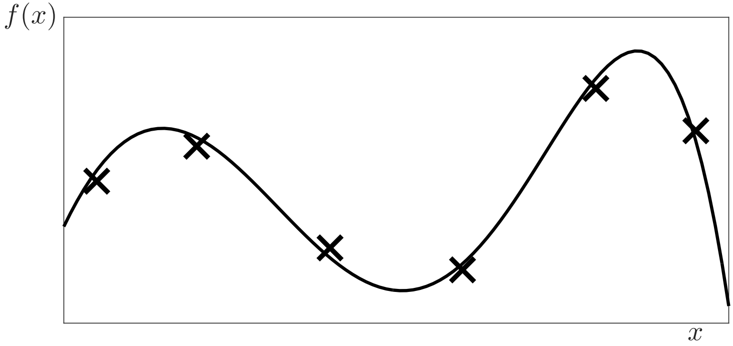
Black-box Optimisation

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



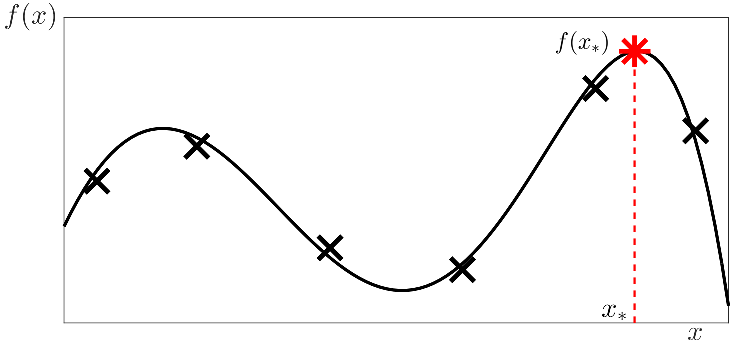
Black-box Optimisation

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



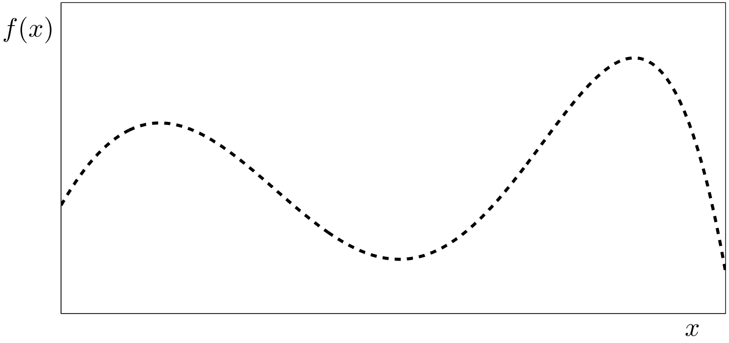
Black-box Optimisation

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



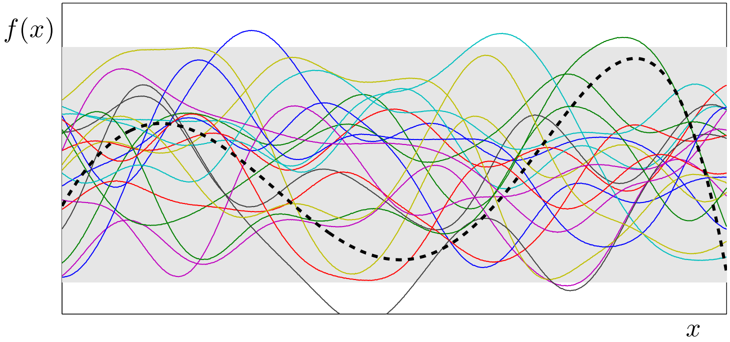
Bayesian Models for f

Functions with no observations



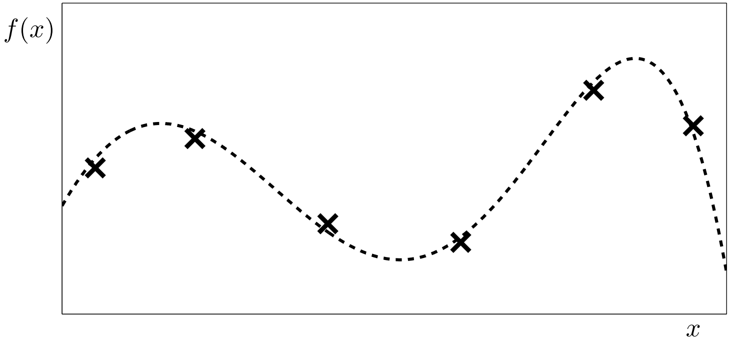
Bayesian Models for f

Prior



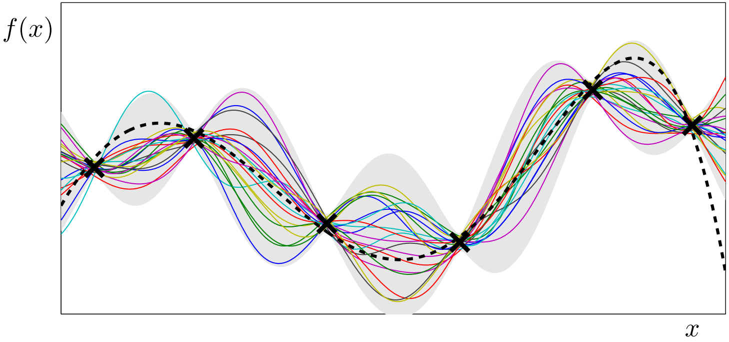
Bayesian Models for f

Observations



Bayesian Models for f

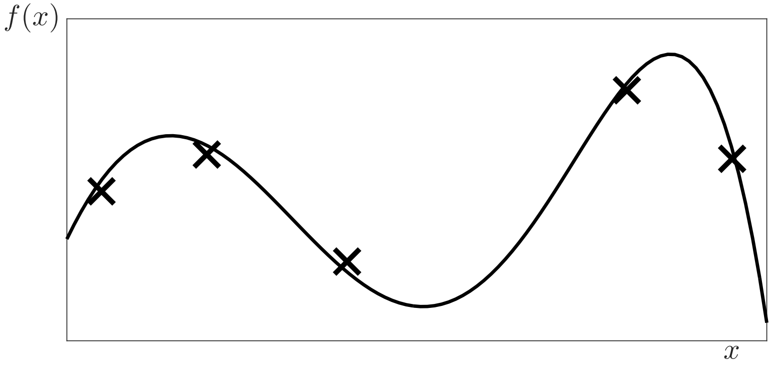
Posterior given observations



Algorithm: Posterior (Thompson) Sampling

(Thompson *1933*)

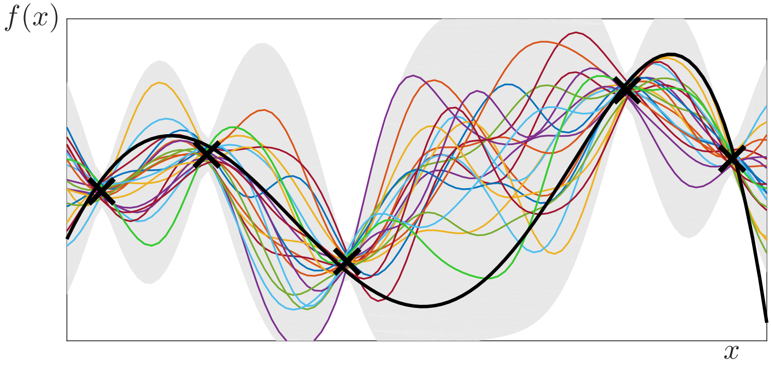
Assume f is drawn from some Bayesian model.



Algorithm: Posterior (Thompson) Sampling

(Thompson *1933*)

Assume f is drawn from some Bayesian model.

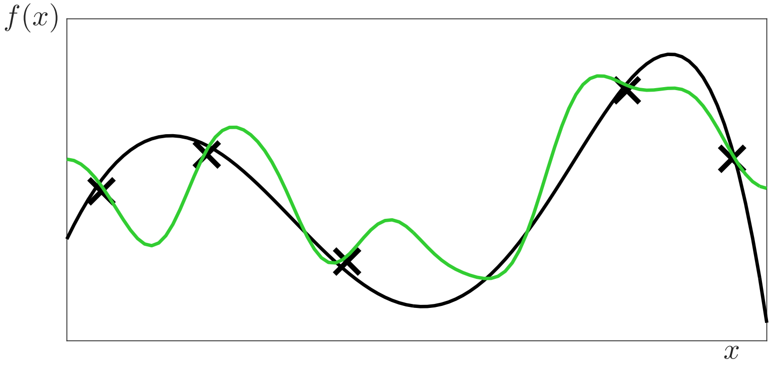


1) Construct posterior.

Algorithm: Posterior (Thompson) Sampling

(Thompson *1933*)

Assume f is drawn from some Bayesian model.



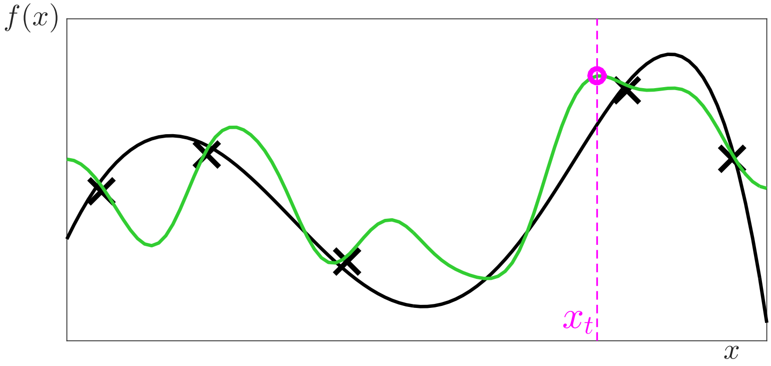
1) Construct posterior.

2) Draw sample g from posterior.

Algorithm: Posterior (Thompson) Sampling

(Thompson *1933*)

Assume f is drawn from some Bayesian model.

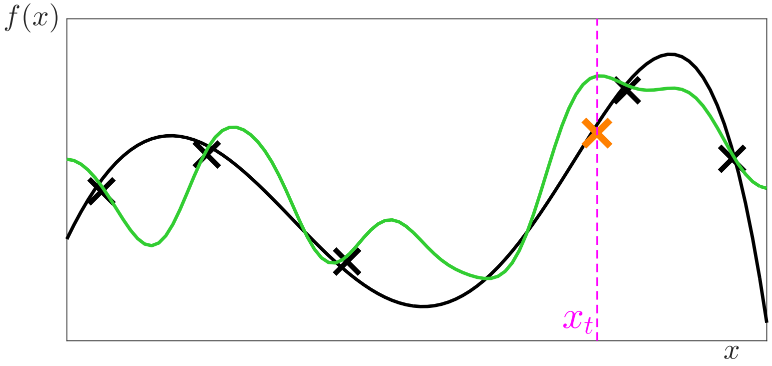


1) Construct posterior.
2) Draw sample g from posterior.
3) Choose $x_t = \mathrm{argmax}_x\, g(x)$.

Algorithm: Posterior (Thompson) Sampling

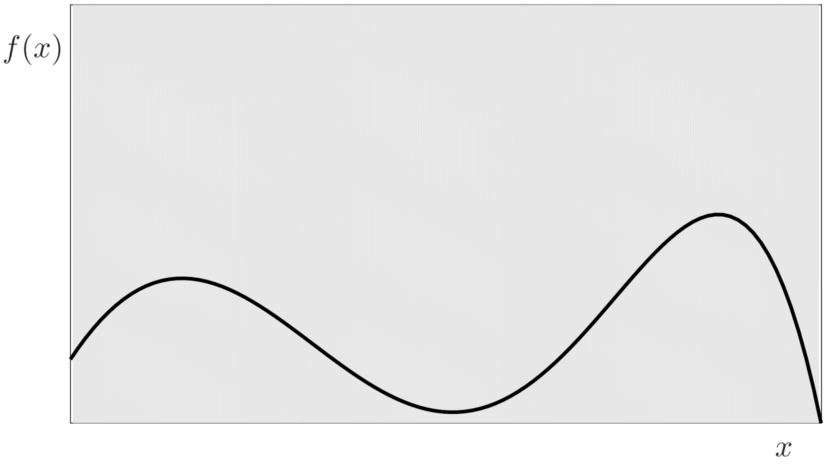
(Thompson *1933*)

Assume f is drawn from some Bayesian model.

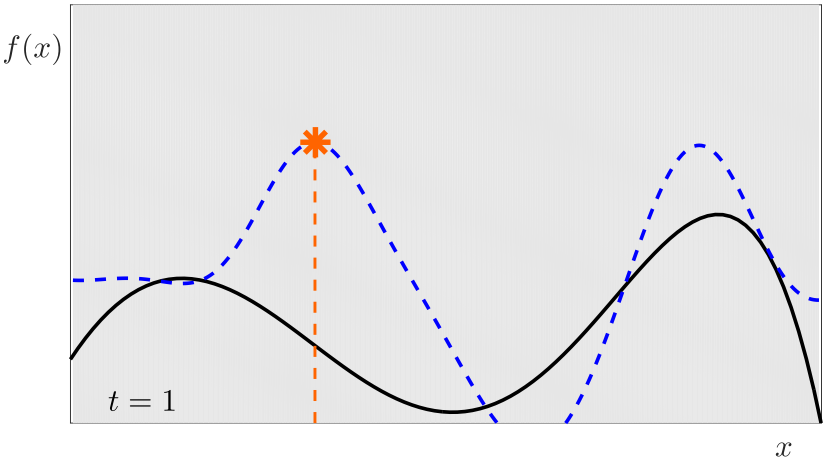


1) Construct posterior.
2) Draw sample g from posterior.
3) Choose $x_t = \mathrm{argmax}_x\, g(x)$.
4) Evaluate f at x_t.

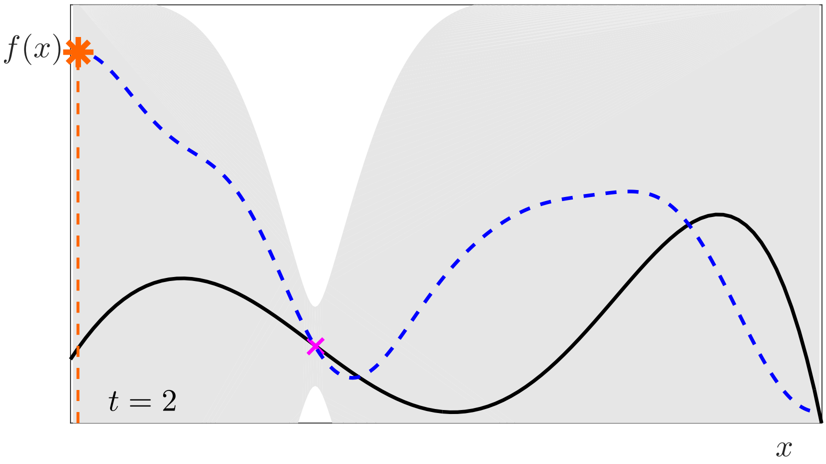
Posterior Sampling – A Simulation (Thompson *1933*)



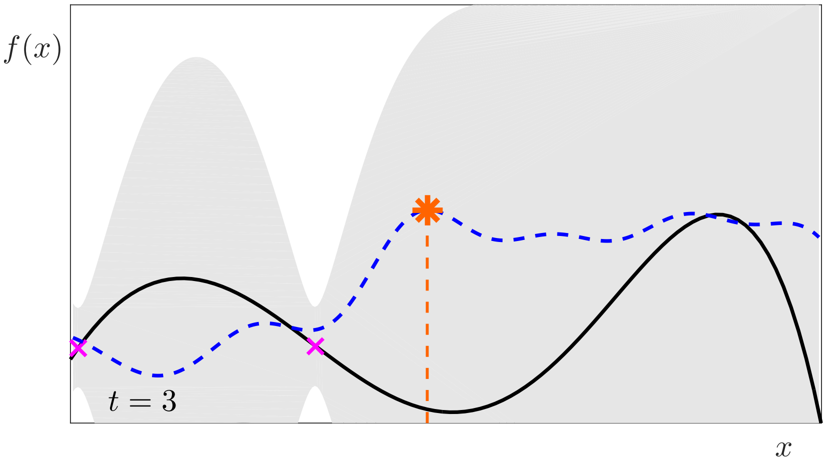
Posterior Sampling – A Simulation (Thompson *1933*)



Posterior Sampling – A Simulation (Thompson *1933*)

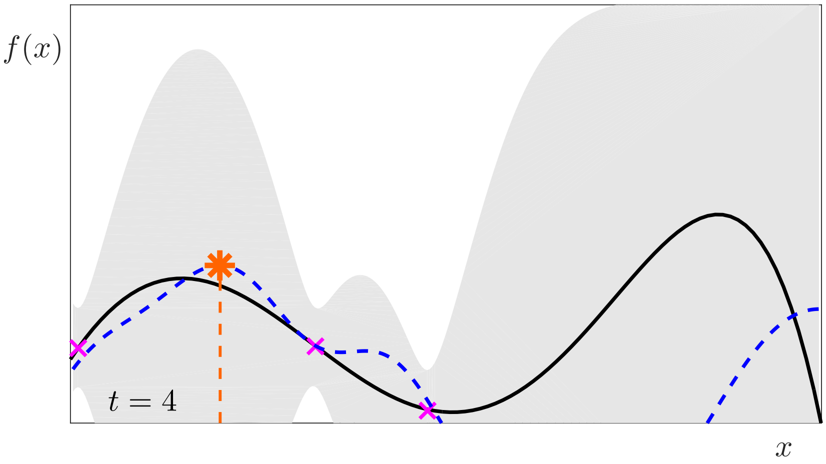


Posterior Sampling – A Simulation (Thompson *1933*)

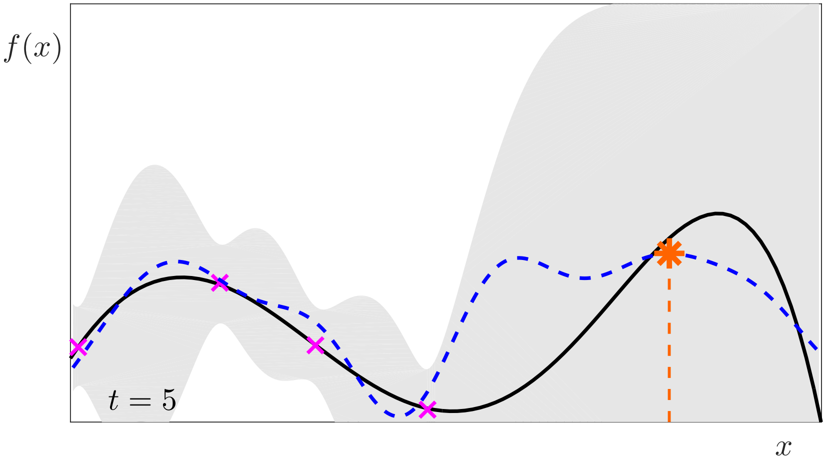


Posterior Sampling – A Simulation

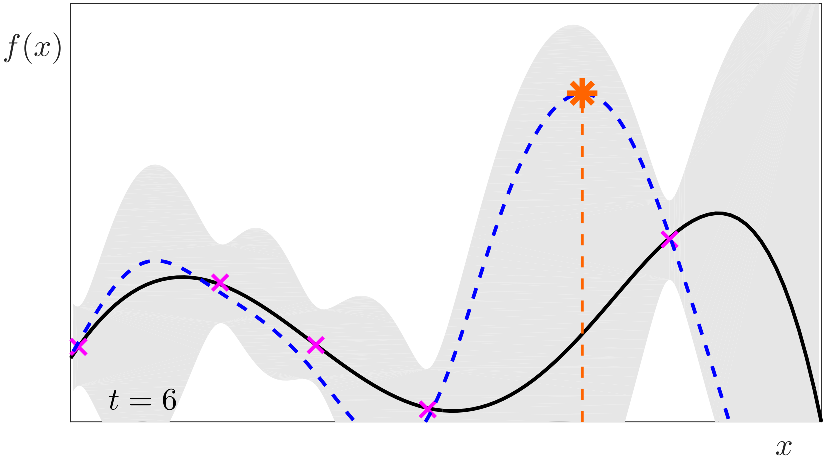
(Thompson *1933*)



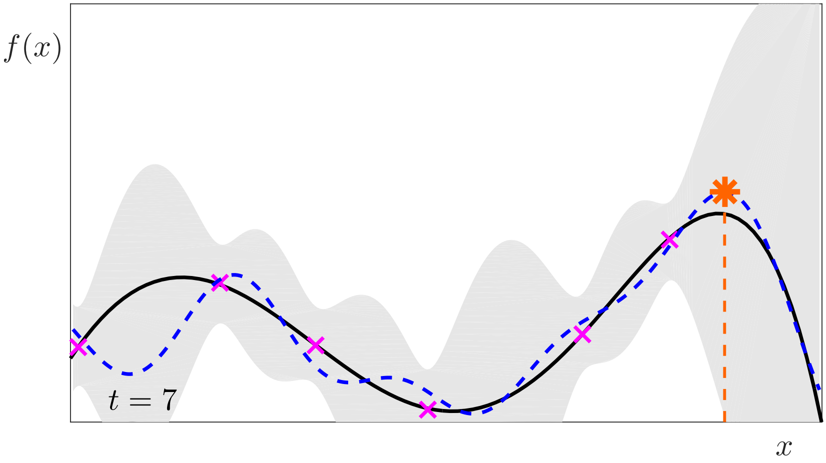
Posterior Sampling – A Simulation (Thompson *1933*)



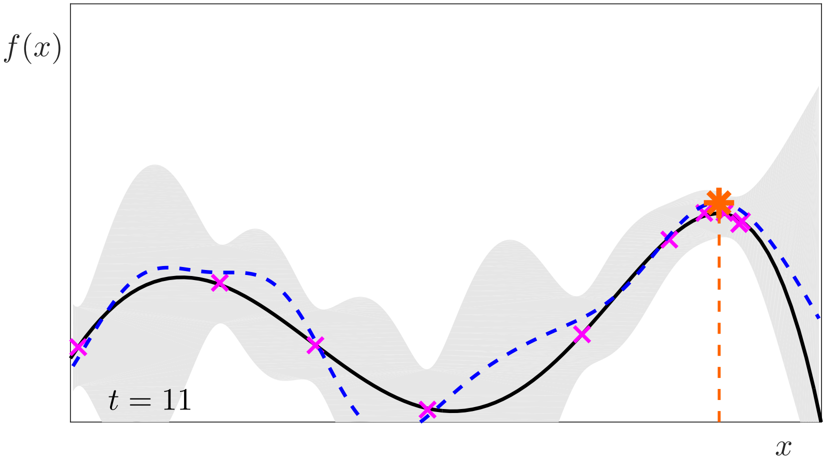
Posterior Sampling – A Simulation (Thompson *1933*)



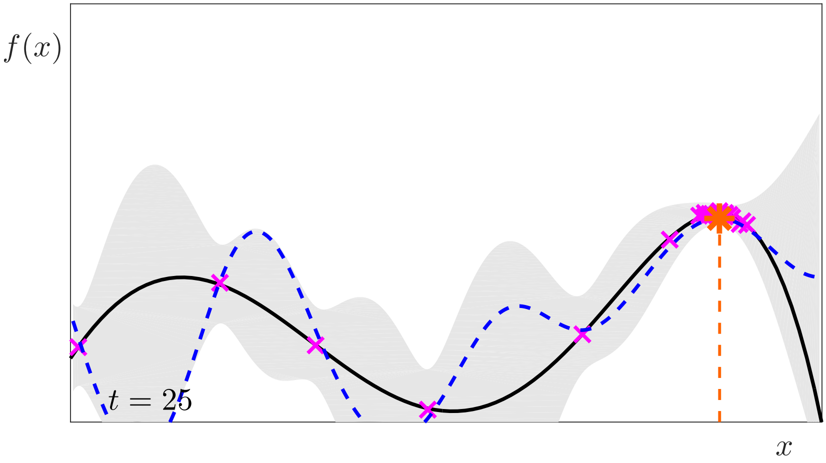
Posterior Sampling – A Simulation (Thompson *1933*)



Posterior Sampling – A Simulation (Thompson *1933*)



Posterior Sampling – A Simulation (Thompson *1933*)



Bayesian Optimisation

Other criteria for selecting x_t:

- Upper Confidence Bounds (Auer et al. 2003, Srinivas et al. 2010)
- Expected improvement (Jones et al. 1998)
- Probability of improvement (Kushner et al. 1964)
- Entropy search (Hernández-Lobato et al. 2014, Wang et al. 2017)
- . . . and a few more.

Bayesian Optimisation

Other criteria for selecting x_t:

- Upper Confidence Bounds (Auer et al. 2003, Srinivas et al. 2010)
- Expected improvement (Jones et al. 1998)
- Probability of improvement (Kushner et al. 1964)
- Entropy search (Hernández-Lobato et al. 2014, Wang et al. 2017)
- . . . and a few more.

Bayesian models for f:

- Gaussian Processes (Jones et al. 1998)
- Neural networks (Snoek et al. 2015)
- Random forests (Hutter 2009)
- Customised models

Outline

1. Blackbox optimisation, Bayesian Models, and Bayesian Optimisation
2. New Frontiers in Bayesian Optimisation:
   Parallel evaluations, High dimensional optimisation, Multi-fidelity optimisation, Multi-objective optimisation
3. Dragonfly: An Open Source Bayesian Optimisation Implementation & Experiments
4. General Settings for Adaptive Goal Oriented Design of Experiments

Outline

1. Blackbox optimisation, Bayesian Models, and Bayesian Optimisation

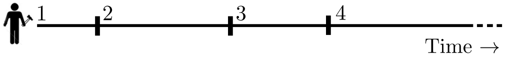
2. New Frontiers in Bayesian Optimisation:
   Parallel evaluations, High dimensional optimisation, Multi-fidelity optimisation, Multi-objective optimisation

3. Dragonfly: An Open Source Bayesian Optimisation Implementation & Experiments

4. General Settings for Adaptive Goal Oriented Design of Experiments

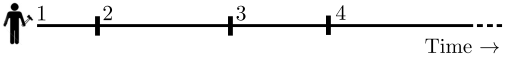
2.1 Parallel Evaluations

Sequential evaluations with one worker

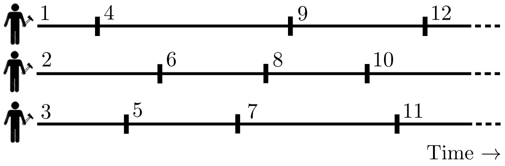


2.1 Parallel Evaluations

Sequential evaluations with one worker

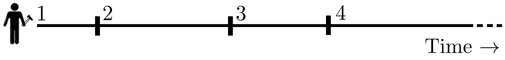


Parallel evaluations with M workers (Asynchronous)

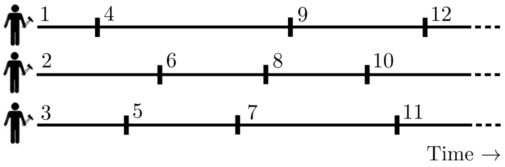


2.1 Parallel Evaluations

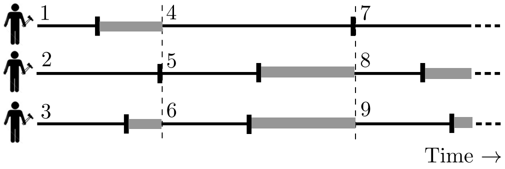
Sequential evaluations with one worker



Parallel evaluations with M workers (Asynchronous)



Parallel evaluations with M workers (Synchronous)



Parallel Evaluations

Direct application of Thompson sampling works! (**K**KSP *AISTATS'18*)

- Conceptually and computationally simple in practice.

Parallel Evaluations

Direct application of Thompson sampling works! (**K**KSP *AISTATS'18*)

- Conceptually and computationally simple in practice.

Theorem (Informal): Parallel posterior sampling

- Both synchronous and asynchronous posterior sampling are almost as good as sequential posterior sampling after n function evaluations.

Parallel Evaluations

Direct application of Thompson sampling works! (**K**KSP *AISTATS'18*)

- Conceptually and computationally simple in practice.

Theorem (Informal): Parallel posterior sampling

- Both synchronous and asynchronous posterior sampling are almost as good as sequential posterior sampling after n function evaluations.
- If evaluation times are the same, the synchronous version is marginally better than asynchronous version.

Parallel Evaluations

Direct application of Thompson sampling works! (KKSP *AISTATS'18*)

- Conceptually and computationally simple in practice.

Theorem (Informal): Parallel posterior sampling

- Both synchronous and asynchronous posterior sampling are almost as good as sequential posterior sampling after n function evaluations.
- If evaluation times are the same, the synchronous version is marginally better than asynchronous version.
- When there is high variability in evaluation times, asynchronous version is better than synchronous.

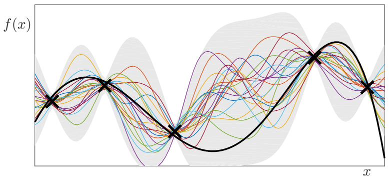
2.2 High Dimensional Bayesian Optimisation

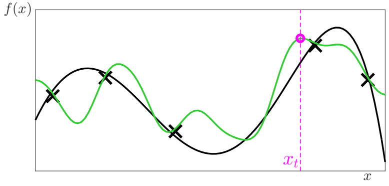
Optimise $f : [0,1]^d \to \mathbb{R}$ when d is very large.

2.2 High Dimensional Bayesian Optimisation

Optimise $f : [0,1]^d \to \mathbb{R}$ when d is very large.

At each time step

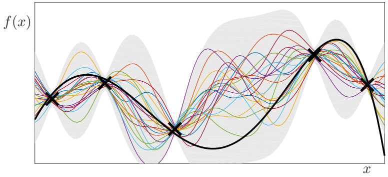


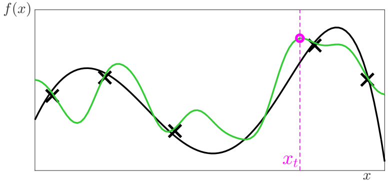


2.2 High Dimensional Bayesian Optimisation

Optimise $f : [0,1]^d \to \mathbb{R}$ when d is very large.

At each time step





1. **Statistical Difficulty:** estimating a high dimensional GP requires several samples.
2. **Computational Difficulty:** maximising a high dimensional acquisition (e.g. upper confidence bound) φ_t.

Additive Models for High Dimensional BO

(KSP *ICML'15*)

$$f(x) = f^{(1)}(x^{(1)}) + f^{(2)}(x^{(2)}) + \ldots + f^{(M)}(x^{(M)}).$$

$x^{(j)}$'s are p-dimensional, $p \ll d$.

Additive Models for High Dimensional BO

(KSP *ICML'15*)

$$f(x) = f^{(1)}(x^{(1)}) + f^{(2)}(x^{(2)}) + \ldots + f^{(M)}(x^{(M)}).$$

$x^{(j)}$'s are p-dimensional, $p \ll d$.

- Theory: Dependence on dimension improves from exponential to linear.
- Better bias-variance trade-off even if f is not additive.
- Add-GP-UCB: algorithm with attractive computational properties.

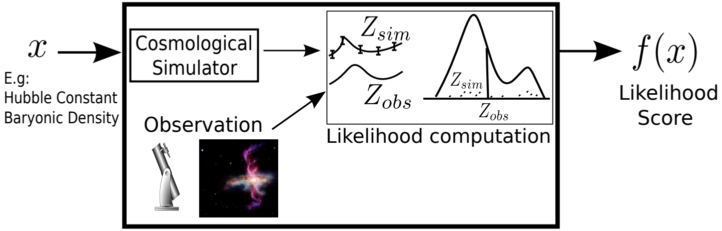
2.3 Multi-fidelity Optimisation

Motivating question:
What if we have cheap approximations to f?

2.3 Multi-fidelity Optimisation

Motivating question:
What if we have cheap approximations to f?

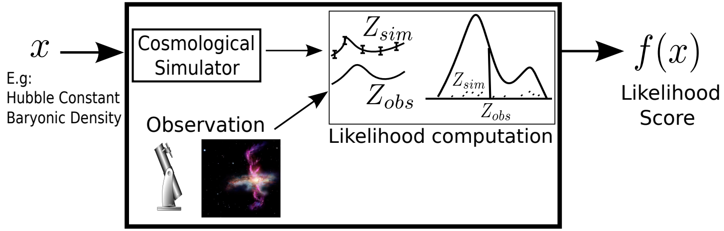
1. In many computational models: simulations and numerical computations with varying levels of granularity.



2.3 Multi-fidelity Optimisation

Motivating question:
What if we have cheap approximations to f?

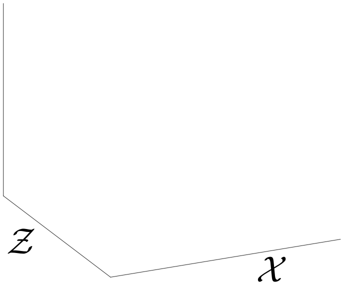
1. In many computational models: simulations and numerical computations with varying levels of granularity.



2. In many applications:
Laborotary experiment $>$ Expensive simulation $>$ Simple computational model

Multi-fidelity Bandits

(KDSP *ICML'17*)



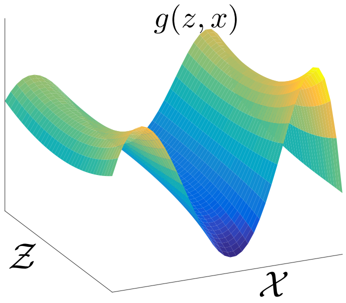
A fidelity space $\mathcal{Z}$ and domain $\mathcal{X}$

$\mathcal{Z} \leftarrow$ all granularity values

$\mathcal{X} \leftarrow$ space of cosmological parameters

Multi-fidelity Bandits

(KDSP *ICML'17*)



A fidelity space $\mathcal{Z}$ and domain $\mathcal{X}$

$\mathcal{Z} \leftarrow$ all granularity values

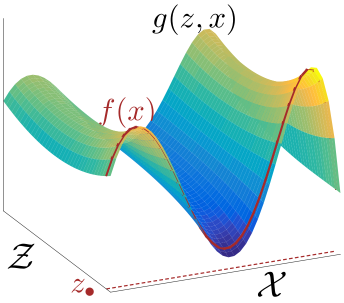
$\mathcal{X} \leftarrow$ space of cosmological parameters

$g : \mathcal{Z} \times \mathcal{X} \to \mathbb{R}$.

$g(z,x) \leftarrow$ likelihood score when performing simulations with granularity z z at cosmological parameters x.

Multi-fidelity Bandits

(KDSP *ICML'17*)



A fidelity space $\mathcal{Z}$ and domain $\mathcal{X}$

$\mathcal{Z} \leftarrow$ all granularity values

$\mathcal{X} \leftarrow$ space of cosmological parameters

$g : \mathcal{Z} \times \mathcal{X} \rightarrow \mathbb{R}$.

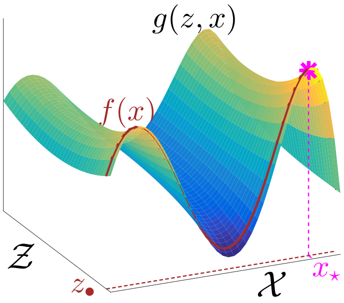
$g(z,x) \leftarrow$ likelihood score when performing simulations with granularity z z at cosmological parameters x.

Denote $f(x) = g(z_\bullet, x)$ where $z_\bullet \in \mathcal{Z}$.

$z_\bullet$ = highest grid size.

Multi-fidelity Bandits

(KDSP *ICML'17*)



A fidelity space $\mathcal{Z}$ and domain $\mathcal{X}$

$\mathcal{Z} \leftarrow$ all granularity values

$\mathcal{X} \leftarrow$ space of cosmological parameters

$g : \mathcal{Z} \times \mathcal{X} \to \mathbb{R}$.

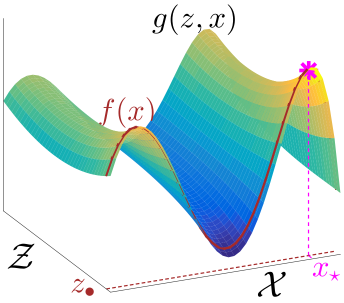
$g(z,x) \leftarrow$ likelihood score when performing simulations with granularity z z at cosmological parameters x.

Denote $f(x) = g(z_\bullet, x)$ where $z_\bullet \in \mathcal{Z}$. $z_\bullet$ = highest grid size.

End Goal: Find $x_\star = \operatorname{argmax}_x f(x)$.

Multi-fidelity Bandits

(KDSP *ICML'17*)



A fidelity space $\mathcal{Z}$ and domain $\mathcal{X}$

$\mathcal{Z} \leftarrow$ all granularity values

$\mathcal{X} \leftarrow$ space of cosmological parameters

$g : \mathcal{Z} \times \mathcal{X} \to \mathbb{R}$.

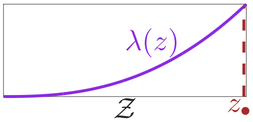
$g(z,x) \leftarrow$ likelihood score when performing simulations with granularity z z at cosmological parameters x.

Denote $f(x) = g(z_\bullet, x)$ where $z_\bullet \in \mathcal{Z}$. $z_\bullet =$ highest grid size.

End Goal: Find $x_\star = \operatorname{argmax}_x f(x)$.

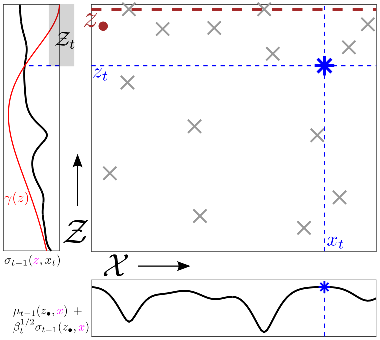
A cost function, $\lambda : \mathcal{Z} \to \mathbb{R}_+$.

$\lambda(z) = \mathcal{O}(z^p)$ (say).



Algorithm: BOCA

(KDSP *ICML'17*)



Model $g \sim \mathcal{GP}(0, \kappa)$ and compute posterior $\mathcal{GP}$:

mean	$\mu_{t-1} : \mathcal{Z} \times \mathcal{X} \to \mathbb{R}$
std-dev	$\sigma_{t-1} : \mathcal{Z} \times \mathcal{X} \to \mathbb{R}_+$

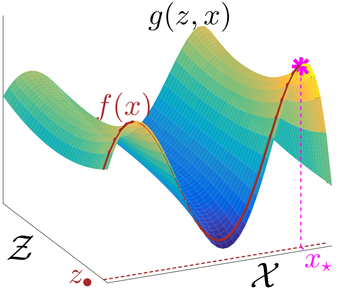
(1) $x_t \leftarrow$ maximise upper confidence bound for $f(x) = g(z_\bullet, x)$.

$$x_t = \underset{x \in \mathcal{X}}{\text{argmax}} \;\; \mu_{t-1}(z_\bullet, x) + \beta_t^{1/2}\sigma_{t-1}(z_\bullet, x)$$

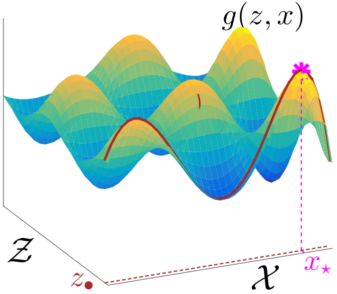
(2) $$\mathcal{Z}_t \approx \{z_\bullet\} \cup \left\{ z \,:\, \sigma_{t-1}(z, x_t) \geq \gamma(z) = \left(\frac{\lambda(z)}{\lambda(z_\bullet)}\right)^q \xi(z) \right\}$$

(3) $$z_t = \underset{z \in \mathcal{Z}_t}{\text{argmin}} \; \lambda(z) \qquad \text{(cheapest } z \text{ in } \mathcal{Z}_t\text{)}$$

Theoretical Results for BOCA

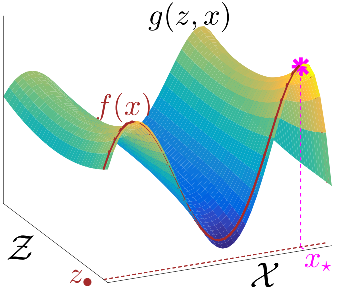


“good”

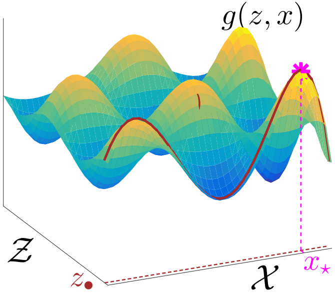


“bad”

Theoretical Results for BOCA



"good"

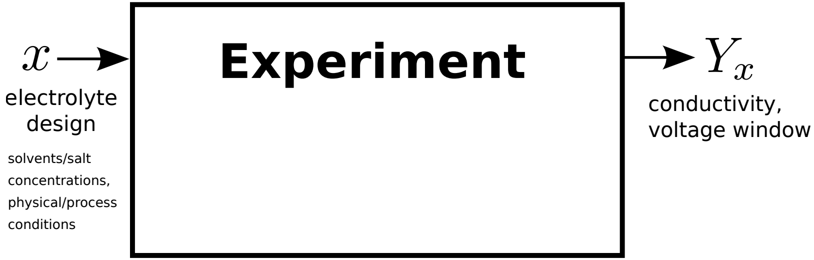


"bad"

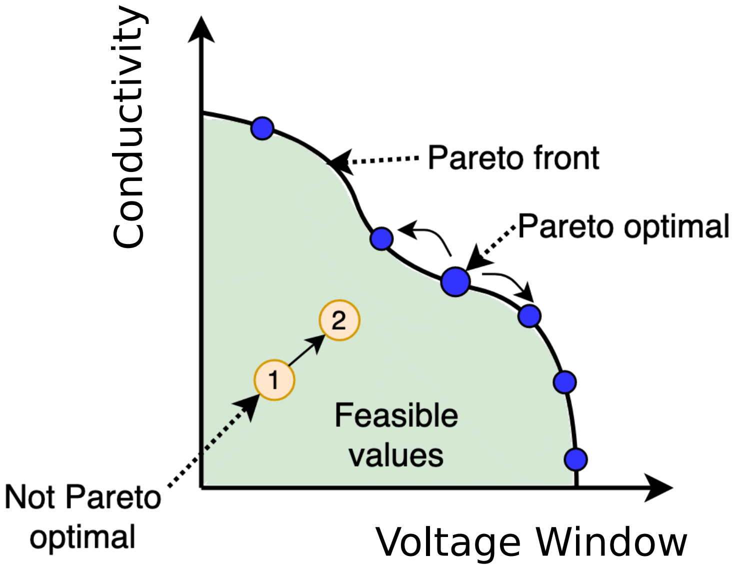
Theorem: (Informal)
BOCA does better, i.e. achieves better Simple regret, than GP-UCB. The improvements are better in the "good" setting when compared to the "bad" setting.

2.4 Multi-objective Optimisation

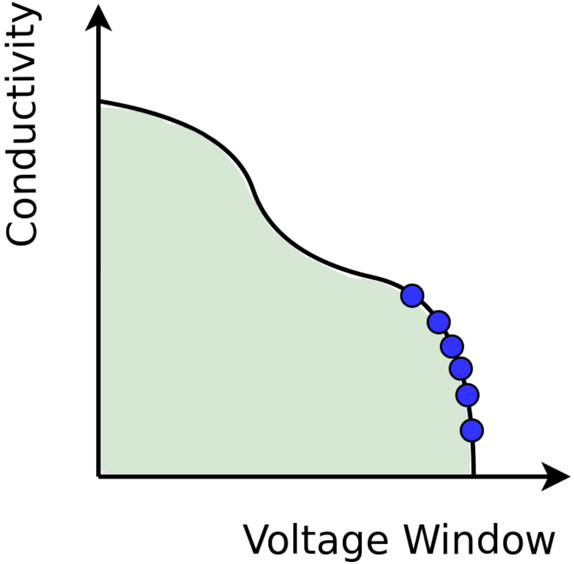
Can we optimise for multiple objectives?


x
electrolyte design
solvents/salt concentrations, physical/process conditions
Experiment
Y_x
conductivity, voltage window

Pareto-optimality



Pareto-optimality



Multi-objective Bayesian Optimisation via Random Scalarisations

(PKP *UAI'19*)

A **scalarisation function** produces a scalar value from multiple objective values.
E.g. linear scalarisation, $s_\lambda(x) = \lambda_1 f_1(x) + \lambda_2 f_2(x)$.
Other examples: Tsebychev scalarisation

For all $\lambda = (\lambda_1, \lambda_2)$, $x^\star_\lambda := \mathrm{argmax}_{x\in\mathcal{X}}\, s_\lambda(x)$ is Pareto optimal.

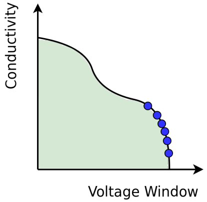
Multi-objective Bayesian Optimisation via Random Scalarisations

(PKP *UAI'19*)

A **scalarisation function** produces a scalar value from multiple objective values.
E.g. linear scalarisation, $s_\lambda(x) = \lambda_1 f_1(x) + \lambda_2 f_2(x)$.
Other examples: Tsebychev scalarisation

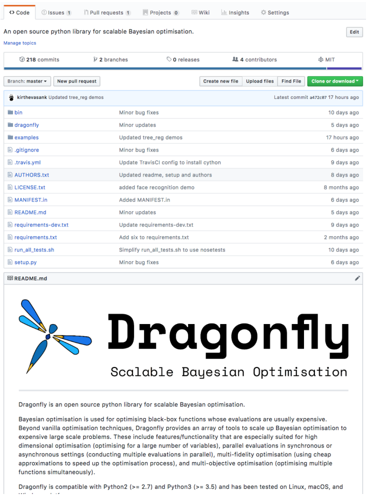
For all $\lambda = (\lambda_1, \lambda_2)$, $x_\lambda^\star := \operatorname{argmax}_{x \in \mathcal{X}} s_\lambda(x)$ is Pareto optimal.

- By randomly sampling λ, we can explore the Pareto front.
- By choosing the sampling distribution, we can control the region of the Pareto front we want to explore.



Outline

Dragonfly (KVNPCSPX *Arxiv'19*)



An open source python library for scalable Bayesian optimisation.

kirthevasank Updated tree_reg demos — Latest commit a472c07 17 hours ago

bin	Minor bug fixes	10 days ago
dragonfly	Minor updates	5 days ago
examples	Updated tree_reg demos	17 hours ago
.gitignore	Minor bug fixes	6 days ago
.travis.yml	Update TravisCI config to install cython	9 days ago
AUTHORS.txt	Updated readme, setup and authors	8 days ago
LICENSE.txt	added face recognition demo	8 months ago
MANIFEST.in	Added MANIFEST.in	6 days ago
README.md	Minor updates	5 days ago
requirements-dev.txt	Update requirements-dev.txt	9 days ago
requirements.txt	Add six to requirements.txt	2 months ago
run_all_tests.sh	Simplify run_all_tests.sh to use nosetests	10 days ago
setup.py	Minor bug fixes	6 days ago

README.md

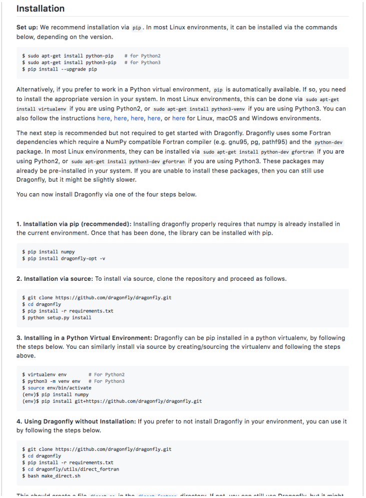
Dragonfly

Scalable Bayesian Optimisation

Dragonfly is an open source python library for scalable Bayesian optimisation.

Bayesian optimisation is used for optimising black-box functions whose evaluations are usually expensive. Beyond vanilla optimisation techniques, Dragonfly provides an array of tools to scale up Bayesian optimisation to expensive large scale problems. These include features/functionality that are especially suited for high dimensional optimisation (optimising for a large number of variables), parallel evaluations in synchronous or asynchronous settings (conducting multiple evaluations in parallel), multi-fidelity optimisation (using cheap approximations to speed up the optimisation process), and multi-objective optimisation (optimising multiple functions simultaneously).

Dragonfly is compatible with Python2 (>= 2.7) and Python3 (>= 3.5) and has been tested on Linux, macOS, and



Installation

Set up: We recommend installation via `pip`. In most Linux environments, it can be installed via the commands below, depending on the version.

```
$ sudo apt-get install python-pip    # for Python2
$ sudo apt-get install python3-pip   # for Python3
$ pip install --upgrade pip
```

Alternatively, if you prefer to work in a Python virtual environment, `pip` is automatically available. If so, you need to install the appropriate version in your system. In most Linux environments, this can be done via `sudo apt-get install virtualenv` if you are using Python2, or `sudo apt-get install python3-venv` if you are using Python3. You can also follow the instructions here, here, here, here, or here for Linux, macOS and Windows environments.

The next step is recommended but not required to get started with Dragonfly. Dragonfly uses some Fortran dependencies which require a NumPy compatible Fortran compiler (e.g. gnu95, pg, pathf95) and the `python-dev` package. In most Linux environments, they can be installed via `sudo apt-get install python-dev gfortran` if you are using Python2, or `sudo apt-get install python3-dev gfortran` if you are using Python3. These packages may already be pre-installed in your system. If you are unable to install these packages, then you can still use Dragonfly, but it might be slightly slower.

You can now install Dragonfly via one of the four steps below.

1. Installation via pip (recommended): Installing dragonfly properly requires that numpy is already installed in the current environment. Once that has been done, the library can be installed with pip.

```
$ pip install numpy
$ pip install dragonfly-opt -v
```

2. Installation via source: To install via source, clone the repository and proceed as follows.

```
$ git clone https://github.com/dragonfly/dragonfly.git
$ cd dragonfly
$ pip install -r requirements.txt
$ python setup.py install
```

3. Installing in a Python Virtual Environment: Dragonfly can be pip installed in a python virtualenv, by following the steps below. You can similarly install via source by creating/sourcing the virtualenv and following the steps above.

```
$ virtualenv env        # For Python2
$ python3 -m venv env   # For Python3
$ source env/bin/activate
(env)$ pip install numpy
(env)$ pip install git+https://github.com/dragonfly/dragonfly.git
```

4. Using Dragonfly without Installation: If you prefer to not install Dragonfly in your environment, you can use it by following the steps below.

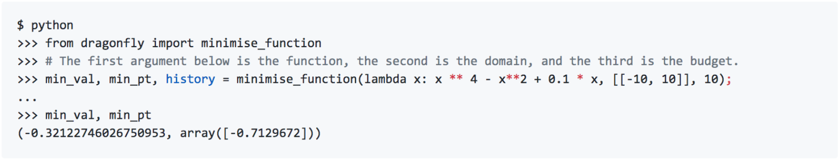
```
$ git clone https://github.com/dragonfly/dragonfly.git
$ cd dragonfly
$ pip install -r requirements.txt
$ cd dragonfly/utils/direct_fortran
$ bash make_direct.sh
```

This should create a file `direct.so` in the `direct_fortran` directory. If not, you can still use Dragonfly, but it might

dragonfly.github.io

`pip install dragonfly-opt`

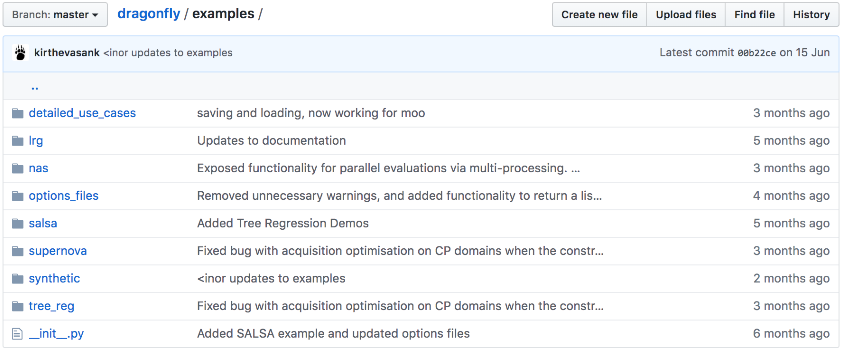
Dragonfly (KVNPCSPX *Arxiv'19*)



```
$ python
>>> from dragonfly import minimise_function
>>> # The first argument below is the function, the second is the domain, and the third is the budget.
>>> min_val, min_pt, history = minimise_function(lambda x: x ** 4 - x**2 + 0.1 * x, [[-10, 10]], 10);
...
>>> min_val, min_pt
(-0.32122746026750953, array([-0.7129672]))
```

Dragonfly (KVNPCSPX *Arxiv'19*)

```
$ python
>>> from dragonfly import minimise_function
>>> # The first argument below is the function, the second is the domain, and the third is the budget.
>>> min_val, min_pt, history = minimise_function(lambda x: x ** 4 - x**2 + 0.1 * x, [[-10, 10]], 10);
...
>>> min_val, min_pt
(-0.32122746026750953, array([-0.7129672]))
```

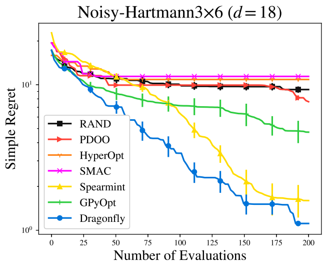


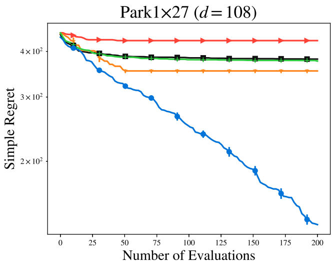
kirthevasank <inor updates to examples — Latest commit 00b22ce on 15 Jun

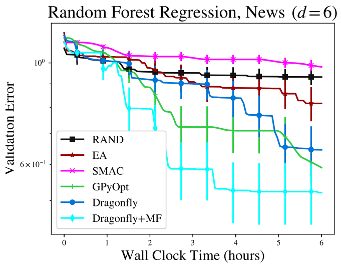
..		
detailed_use_cases	saving and loading, now working for moo	3 months ago
lrg	Updates to documentation	5 months ago
nas	Exposed functionality for parallel evaluations via multi-processing. …	3 months ago
options_files	Removed unnecessary warnings, and added functionality to return a lis…	4 months ago
salsa	Added Tree Regression Demos	5 months ago
supernova	Fixed bug with acquisition optimisation on CP domains when the constr…	3 months ago
synthetic	<inor updates to examples	2 months ago
tree_reg	Fixed bug with acquisition optimisation on CP domains when the constr…	3 months ago
__init__.py	Added SALSA example and updated options files	6 months ago

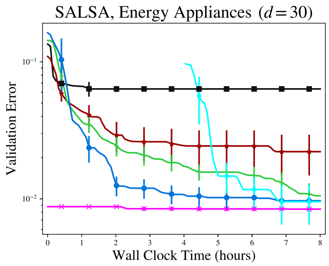
Dragonfly

(KVNPCSPX *Arxiv'19*)







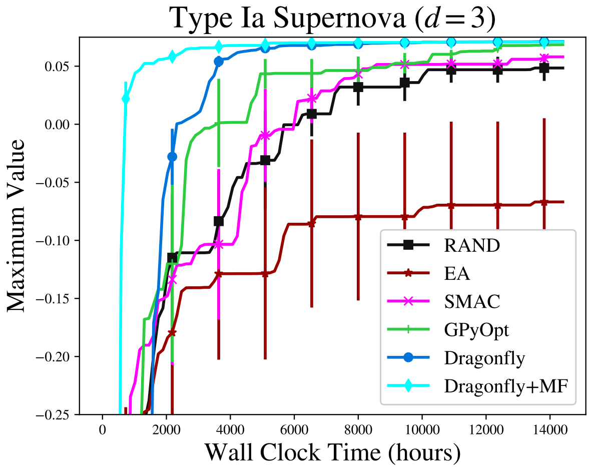


Dragonfly: Cosmological inference on Type-1a supernovae

Estimate Hubble constant, dark matter fraction & dark energy fraction using data on Type-1a supernovae. Approximate using less data and/or less granular grid for numerical integration.

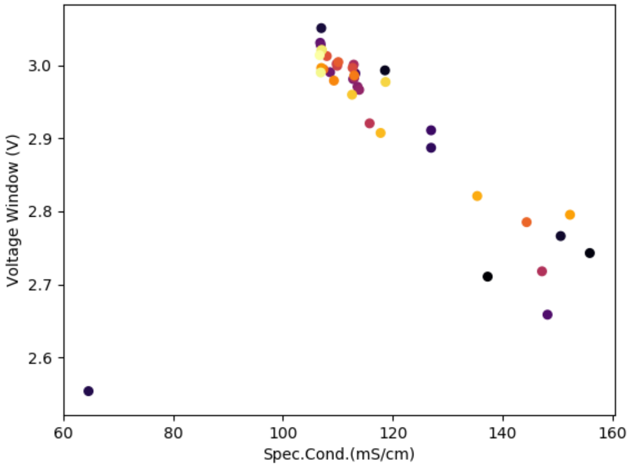
Dragonfly: Cosmological inference on Type-1a supernovae

Estimate Hubble constant, dark matter fraction & dark energy fraction using data on Type-1a supernovae. Approximate using less data and/or less granular grid for numerical integration.



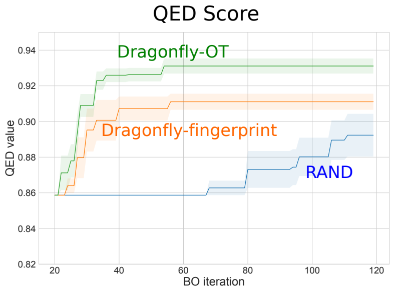
Dragonfly: Electrolyte Design

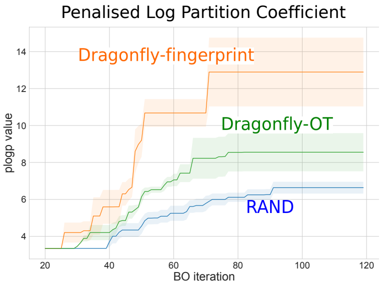
Multi-objective optimisation (conductivity, voltage window). Optimising for concentrations of $LiNO_3$, Li_2SO_4, and $NaClO_4$ in an aqueous medium.



Dragonfly: Optimising Small Molecules (KXKNSPX *Arxiv'19*)

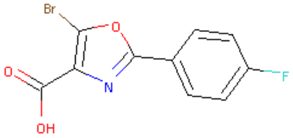
Discover organic small molecules with high drug-likeness scores (QED score, penalised log partition coefficient etc.).



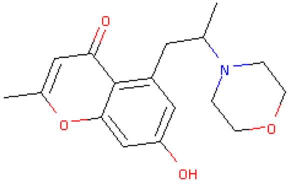


* Also uses synthesis predictors (e.g. RexGen, (CJRJJGBJ *'19*)) to provide a synthesis recipe along with each recommendation.

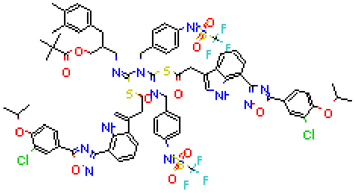
Dragonfly: Optimising Small Molecules (KXKNSPX *Arxiv'19*)



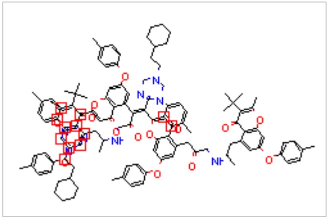
QED = 0.92145



QED = 0.94087



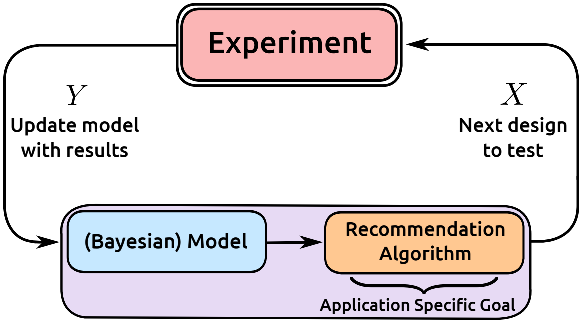
P-logP = 11.988



P-logP = 11.270

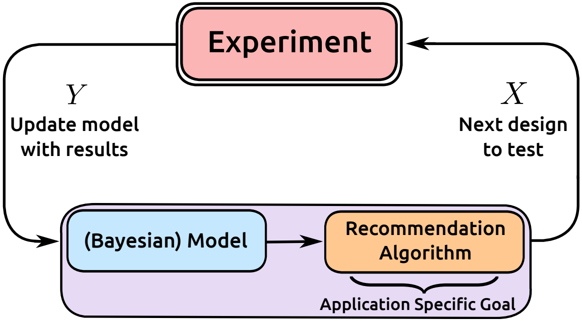
Outline

Adaptive Goal Oriented Design of Experiments



- Blackbox Optimisation
- Active Learning
- Active Quadrature (Osborne et al. 2012)
- Active Level Set Estimation (Gotovos et al. '13)
- Active Search (Ma et al. '17)
- Active Posterior Estimation (Kandasamy et al. '15)

Adaptive Goal Oriented Design of Experiments



- Blackbox Optimisation
- Active Learning
- Active Quadrature (Osborne et al. 2012)
- Active Level Set Estimation (Gotovos et al. '13)
- Active Search (Ma et al. '17)
- Active Posterior Estimation (Kandasamy et al. '15)

Issues:

- New goal/setting $\implies$ New algorithm?
- Algorithms tend to depend on the model and vice versa.

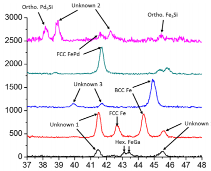
Phase Identification in Alloys

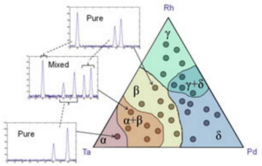
$$\begin{tikzcd} x \arrow[r] & \boxed{\text{X-Ray Diffraction Studies}} \arrow[r] & Y_x \end{tikzcd}$$

$\mathcal{X}$: composition of alloy

Diffraction patterns

$\mathcal{Y} \subset \mathbb{R}^K$





Goal: Identify changes in crystal structure in an alloy.

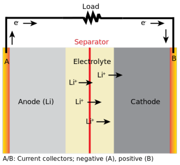
Multiple Goals in Electrolyte Design

$$x \longrightarrow \boxed{\begin{array}{c}\text{MD/DFT Simulations,}\\ \text{Viscometer tests,}\\ \text{Impedance spectroscopy,}\\ \text{UV-vis spectroscopy}\end{array}} \longrightarrow Y_x$$

$\mathcal{X}$:
solvents/salt concentrations, physical/process conditions

Y_x
solubility, viscosity, conductivity

$\mathcal{Y} \subset \mathbb{R}^K$



Goal: Actively learn viscosity and solubility, while simultaneously optimising conductivity.

Adaptive Goal Oriented Design of Experiments

(KNZKSP *ICML'19*)

1. **System:**
   - An *unknown* parameter θ completely specifies the system.
   - A prior $\mathbb{P}(\theta)$ and a likelihood $\mathbb{P}(Y|X,\theta)$.

Adaptive Goal Oriented Design of Experiments

(KNZKSP *ICML'19*)

1. **System:**
   - An *unknown* parameter θ completely specifies the system.
   - A prior $\mathbb{P}(\theta)$ and a likelihood $\mathbb{P}(Y|X,\theta)$.
2. **Goal:**
   - Collect data $D_n = \{(x_t, y_{x_t})\}_{t=1}^{n}$ to maximise a user specified reward function $\lambda(\theta, D_n)$.

Algorithm: Myopic Posterior Sampling (MPS)

Inspired by Posterior Sampling.

Algorithm: MPS

```
- Set $D_0 \leftarrow$ initial data.
- For $t = 1, 2, \ldots,$ do
  1. Sample $\theta' \sim \mathbb{P}(\theta | D_{t-1})$.
  2. Choose $x_t = \mathrm{argmax}_{x \in \mathcal{X}} \lambda^+(\theta', D_{t-1}, x)$.
  3. $y_{x_t} \leftarrow$ conduct experiment at $x_t$.
  4. Set $D_t \leftarrow D_{t-1} \cup \{(x_t, y_{x_t})\}$.
```

Algorithm: Myopic Posterior Sampling (MPS)

Inspired by Posterior Sampling.

Algorithm: MPS

```
- Set $D_0 \leftarrow$ initial data.
- For $t = 1, 2, \ldots,$ do
  1. Sample $\theta' \sim \mathbb{P}(\theta | D_{t-1})$.
  2. Choose $x_t = \mathrm{argmax}_{x \in \mathcal{X}} \lambda^+(\theta', D_{t-1}, x)$.
  3. $y_{x_t} \leftarrow$ conduct experiment at $x_t$.
  4. Set $D_t \leftarrow D_{t-1} \cup \{(x_t, y_{x_t})\}$.
```

Only require that we can sample from the posterior $\mathbb{P}(\theta | D_{t-1})$.

- Many probabilistic programming tools available today.

Algorithm: Myopic Posterior Sampling (MPS)

Inspired by Posterior Sampling.

Algorithm: MPS

```
- Set $D_0 \leftarrow$ initial data.
- For $t = 1, 2, \dots,$ do
  1. Sample $\theta' \sim \mathbb{P}(\theta | D_{t-1})$.
  2. Choose $x_t = \mathrm{argmax}_{x \in \mathcal{X}} \lambda^+(\theta', D_{t-1}, x)$.
  3. $y_{x_t} \leftarrow$ conduct experiment at $x_t$.
  4. Set $D_t \leftarrow D_{t-1} \cup \{(x_t, y_{x_t})\}$.
```

Only require that we can sample from the posterior $\mathbb{P}(\theta | D_{t-1})$.

- Many probabilistic programming tools available today.

$\lambda^+(\theta', D, x) \leftarrow$ expected next step reward if θ' was the system, we already have data D, and we were to conduct an experiment at x:

$$\lambda^+(\theta', D, x) = \mathbb{E}_{Y_x \sim \mathbb{P}(Y|x, \theta')} \Big[\lambda\big(\theta',\, D \cup \{(x, Y_x)\}\big) \Big].$$

Theory

Theorem (Informal): Under certain conditions, MPS is competitive with a *globally* optimal oracle that *knows* θ.

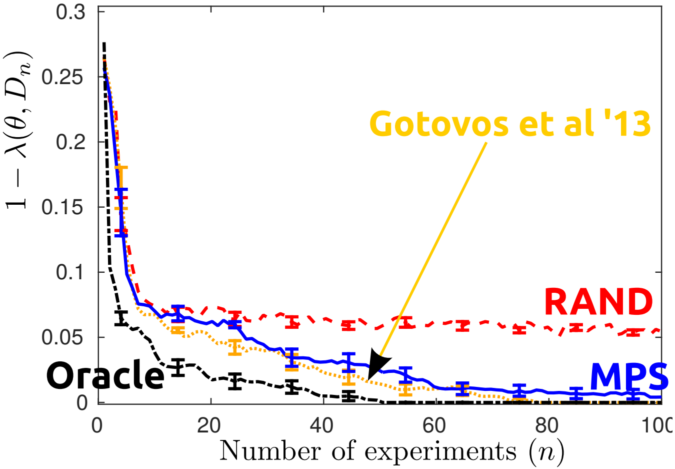
$$\mathbb{E}[\lambda(\theta, D_n) | D_n \sim \pi_{\mathrm{M}}^{\mathrm{PS}}] \geq$$

$$(1-\gamma)\mathbb{E}[\lambda(\theta, D_{\gamma n}^{\star}) | D_{\gamma n}^{\star} \sim \pi_{\mathrm{G}}^{\star}] - \sqrt{\frac{|\mathcal{X}|\tau_n \Psi_n}{2n}}.$$

Proof ideas from
- Adaptive Submodularity
- Reinforcement Learning
- Bandits

Experiment: Active Level Set Estimation

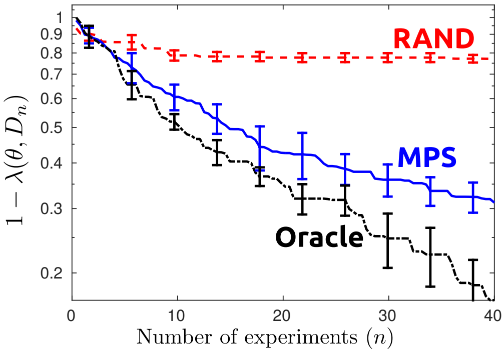
$$\lambda(\theta_\star, D_n) = -\mathrm{vol}(\mathbb{1}\{S_{\theta_\star,L} \neq \hat{S}_{D_n,L}\})$$



Experiment: Custom Goal in Electrolyte Design

An experiment measures solubility, viscosity and conductivity of an electrolyte design.
Goal: Optimise conductivity while learning solubility and viscosity.

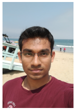
$$\lambda(\theta_\star, D_n) = \|f_{\mathrm{dissol}} - \hat{f}_{\mathrm{dissol}}(D_n)\|^2 + \|f_{\mathrm{vis}} - \hat{f}_{\mathrm{vis}}(D_n)\|^2 + \\ (\max f_{\mathrm{con}} - \max_{X_t, t \le n} f_{\mathrm{con}}(X_t)),$$



Adarsh

Akshay

Barnabás


Biswajit

Chris

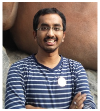
Eric

Gautam

Jay

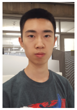
Jeff

Junier


Karun

Ksenia

Rajat


Reed

Sailun

Sanjay

Venkat

Willie

Thank You

Slides:

`people.eecs.berkeley.edu/~kandasamy/talks/maryland_slides_aug2019.pdf`

Summary

Bayesian models allow quantifying uncertainty system given experimental results $\rightarrow$ called the posterior.
- Use posterior to plan future experiments.

Bayesian Optimisation: used for optimising black-box systems.

- Conduct multiple parallel function calls. (**K**KSP *AISTATS'18*)
- Multi-fidelity optimisation: Use cheap approximations to a an expensive experiment to speed up optimisation. (**K**DSP *NeurIPS'16a*, **K**DOSP *NeurIPS'16b*, **K**DSP *ICML'17*)
- Find Pareto front when optimising multiple criteria (P**K**P *UAI'19*)
- Additive models have favourable statistical and computational properties in high dimensional optimisation. (**K**SP *ICML'15*)

Dragonfly: A library for scalable Bayesian optimisation. Applied to problems in electrolyte design, drug discovery etc. (**K**VNPCSPX *Arxiv'19*)

Bayesian methods for Goal Oriented Design of Experiments: (**K**NZKSP *ICML'19*)