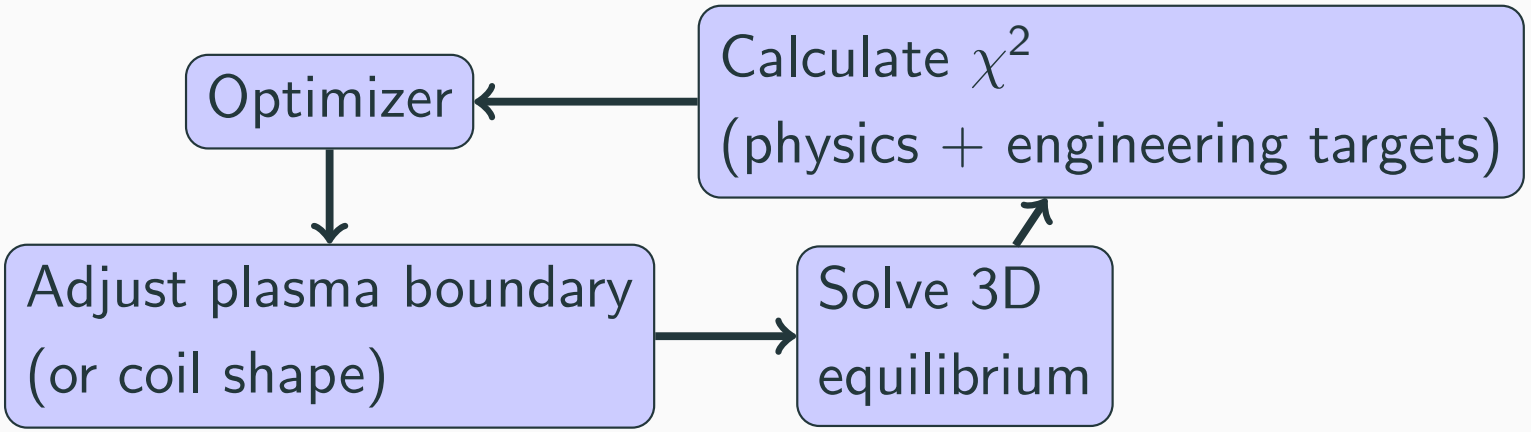
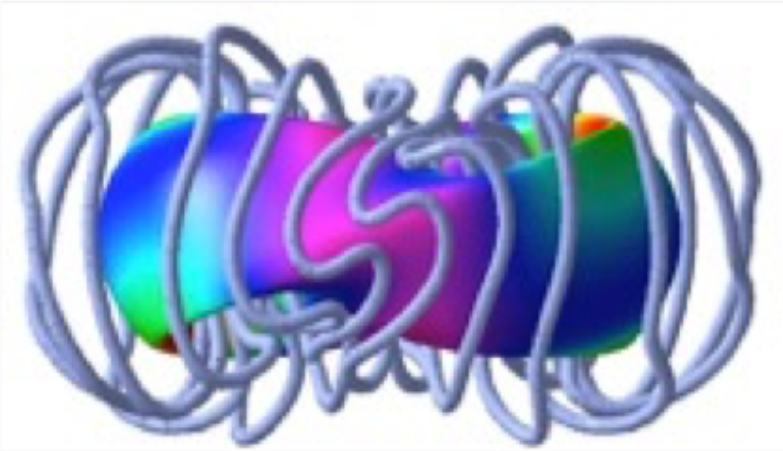
Software for Parallel Global Optimization

David Bindel

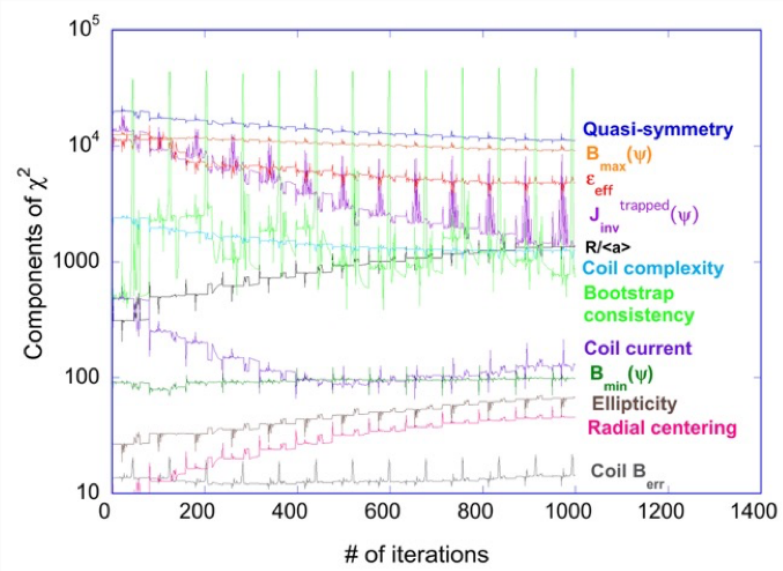
17 Oct 2019

Optimizing with Expensive Physics (Stellarators)





$$r(\phi,\theta) + iz(\phi,\theta) = \sum \alpha_{m,n} e^{i(m\phi - n\theta)}$$



Global optimization

$$\text{Find } x^* \in \Omega \text{ s.t. } f(x^*) \leq f(x), \forall x \in \Omega$$

Assumptions:

- $f : \Omega \to \mathbb{R}$ continuous, expensive, black-box (?)
  - "Expensive" means few evals allowed (10s? 100s?)
  - "Expensive" may be relative to optimizer computations
  - "Black-box" usually means without derivatives
  - ... but I'm really interested in auxiliary information!
- $\Omega \subset \mathbb{R}^d$ a hypercube
  - Constrain Ω further with penalties (not a great idea)?
  - We *will* generally insist on Ω compact
  - Some variants allow discrete variables as well

More complex problem formulations matter; ignore that for now.

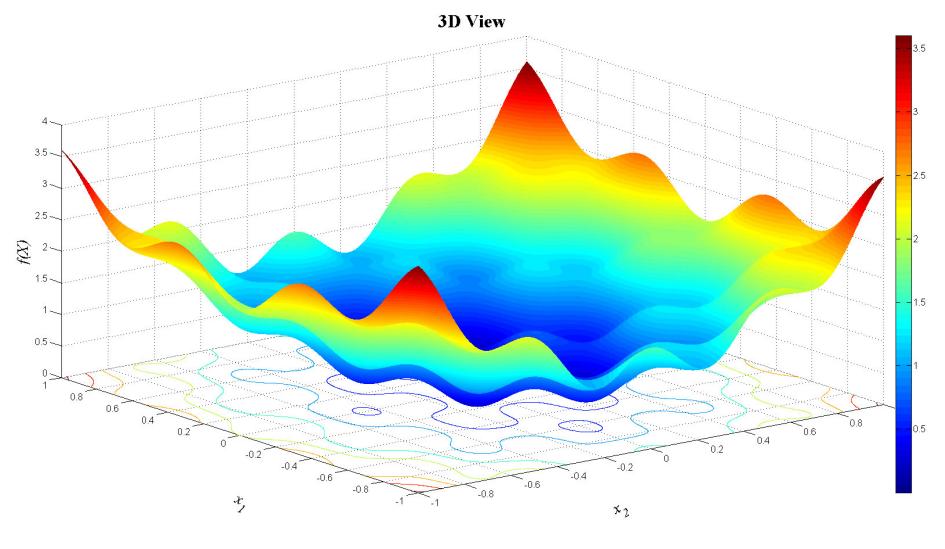
Difficulties with global optimizers

- (Multi-start) Gradient-based optimizers
  - **Examples:** Gradient descent, quasi-Newton
  - **Problems:** (Accurate) derivative computations, multi-modality
- (Multi-start) Derivative-free methods
  - **Examples**: Nelder-Mead, pattern search, misc Powell methods
  - **Problems:** Slow convergence, multi-modality
- Heuristic sampling methods
  - **Examples:** Genetic algorithms, simulated annealing
  - **Problems:** Requires many evaluations
- **Surrogate methods** — my favorite
  - **Examples:** Bayesian optimization, RBF-based solvers
  - **Problems:** Cost to maintain and use surrogate

Surrogate optimization

- Key: Use a surrogate $\hat{f}$ to approximate objective f
  - Assume that evaluating $\hat{f}$ is inexpensive
  - Interpolatory (e.g. RBF) or not (e.g. low-degree poly)
  - May come with some uncertainty measure
- Initial phase:
  - Pick a "good" experimental design $\{x_1, \ldots, x_m\} \in \Omega$
  - Evaluate $f(x)$ at experimental points
  - Fit initial surrogate $\hat{f}$
- Adaptive phase:
  - Use surrogate to guide next sample
    - Generally involves solving an auxiliary problem
    - Goal: balance exploration vs exploitation
  - Update surrogate with new evaluations

Optimal Optimization



Two meanings for "optimization:"

- Mathematical programming / operations research:

$$\min_{x \in \Omega} c(x) \text{ s.t. constraints}$$

- High performance computing / code tuning:

  Minimize run time (usually) subject to resources

Too Slow, Now What?

What if I decide my optimization code is too slow?

- Simplify the problem?
- Use better algorithms (discretizations or optimizers)?
- Improve the initial guess or add information?
- *Parallelize*?

In the Cloud

There is no cloud.
It's just someone else's computer.

- Renting physical or virtual machines + net (IaaS)
- Not *that* different from cluster!
- Key distinctions: elasticity and shared tenancy

Some Key Characteristics

Properties of STELLOPT relevant to parallel code performance:

- Non-convex global optimization problem
- Gray box optimization
- Computationally intensive rather than data intensive
- May not need massively parallel *simulations*

Why might this be a good fit for a cloud?

Parallelism

Want to speed up the computation via parallel resources

- Sources of parallelism
  - Global exploration (weak coupling)
    - Evaluate population / swarm / etc (GA and other heuristics)
    - Run from multiple starts (multi-start descent methods)
    - Evaluate experimental design (surrogate methods)
    - Amount of parallelism = size of population
  - Local exploration / gradient estimation
    - Finite difference derivative estimates
    - Evaluating points in a pattern
    - Amount of parallelism $\approx$ problem dimension
  - Parallelism within the feval or aux problem
- Styles of parallelism
  - Synchronous: within distinct phases/steps, pause at barriers
  - Asynchronous: no barriers

Performance Facts: Cloud Edition

There is no cloud.
It's just someone else's computer.

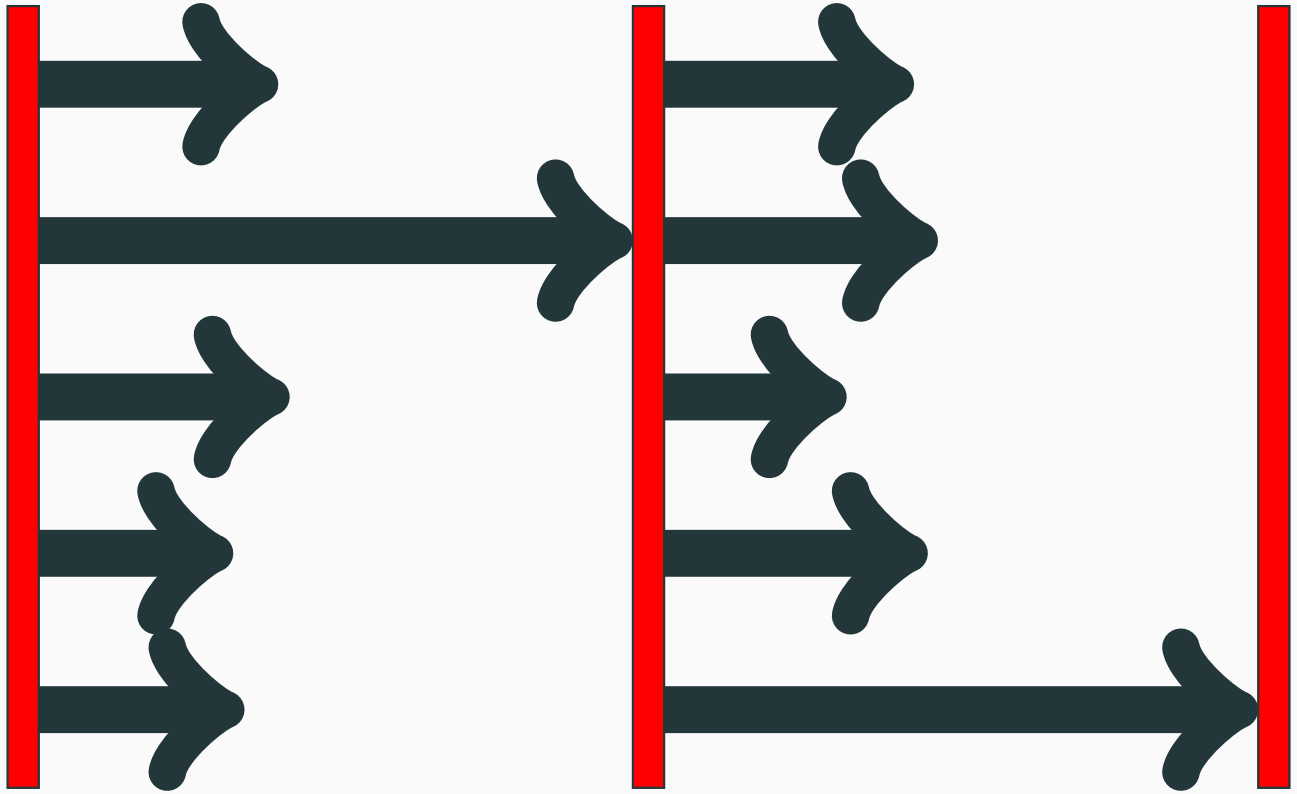
Overheads beyond a local compute cluster or supercomputer:

- Maybe virtualization (esp. NIC)
- Shared tenancy on nodes

Key woe: when these effects cause *high variance* in times
... at least, if tightly coupled

NB: Not unique to clouds, and not the only source of variance!

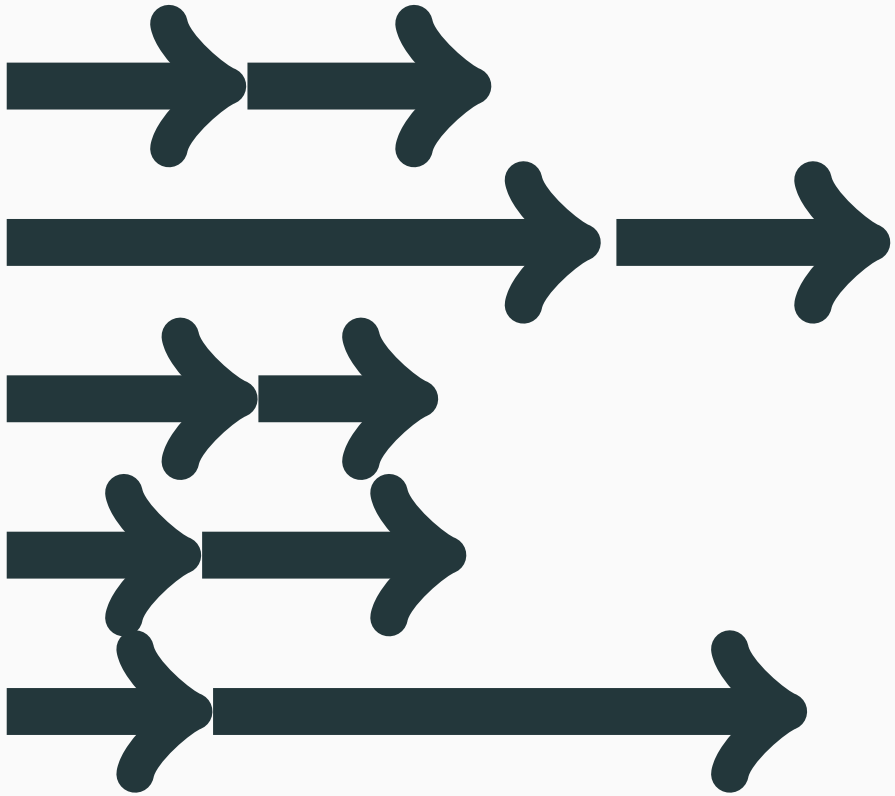
Dangerous Delays



Bad news for:

- Tightly coupled PDE solvers
- Standard step-by-step optimizers (Newton, BFGS, ...)

Breaking Barriers: Asynchronous Edition



The Challenge

Goal: Framework for surrogate-based optimization

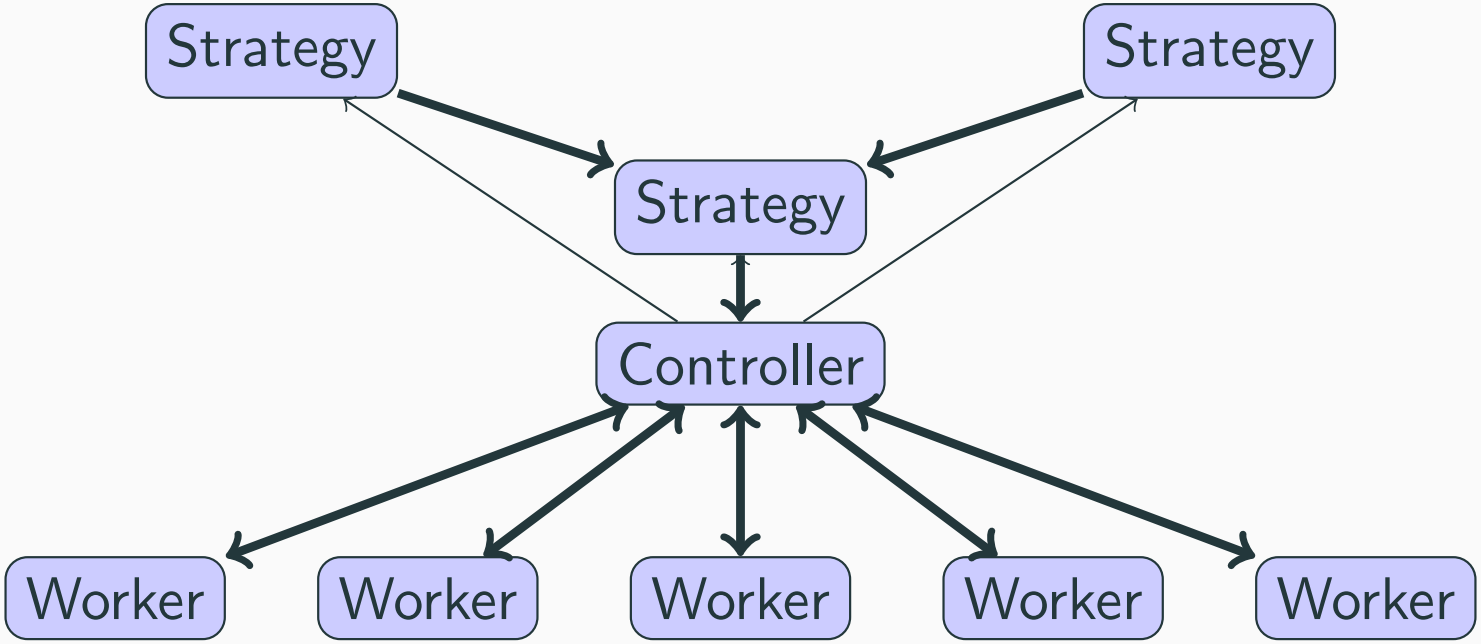
- Serial or parallel (synchronous or *asynchronous*)
- Incorporating bounds or partial evaluations
- Variable run time
- Early terminations and partial failures allowed

... and ideally not too hard to program.

A brief history

- **Phase 0 (2010-11)**:
  - **Late 2010**: Shoemaker asks Birman about parallel global optimization; Birman introduces Bindel.
  - **Dec 2010**: Proposal to NSF: "Parallel Global Optimization Algorithms with Asynchrony, Adaptive Re-Planning, and Response Surfaces for Costly Simulations."
  - **Aug 2011**: Notification of award from NSF.
- **Phase 1 (2012–2013)**:
  - 2012: Initial parallel code with Yilun Wang, Jungmin Yun, and Jesseon Chang as CS MEng + summer project.
  - 2013: Julie Müller extends + runs on Yellowstone.
- **Phase 2 (2014–2018)**:
  - Bindel rewrites "plumbing" layer POAP (Summer 2014);
  - Eriksson writes PySOT (Summer 2015);
  - Paul West adapts for cloud (Spring 2017);
  - Many updates up to David E graduation (Fall 2018).

POAP



- Plumbing for Optimization with Asynchronous Parallelism
- Provides an event-driven programming abstraction
- User writes *strategies* that get updates, request actions
- *Controller* handles action requests, manages workers

POAP

- Three main components
  1. A **strategy** to propose new evaluations.
  2. A set of **workers** that carry out evaluations.
  3. A **controller** that orchestrates workers.
- The controller is also responsible for
  1. Accepting/rejecting proposals by the strategy.
  2. Controlling and monitoring workers.
  3. Informing the strategy object of relevant events.
- Can naturally compose strategies.
- Workers and strategy communicate via the controller.

```
from poap.strategy import FixedSampleStrategy
from poap.strategy import CheckWorkStrategy
from poap.controller import ThreadController
from poap.controller import BasicWorkerThread

# samples = list of sample points ...

controller = ThreadController()
sampler = FixedSampleStrategy(samples)
controller.strategy = CheckWorkerStrategy(controller, sampler)

for i in range(NUM_WORKERS):
    t = BasicWorkerThread(controller, objective)
    controller.launch_worker(t)

result = controller.run()
print('Best result: 0 at 1'.format(result.value, result.params))
```

POAP Built-Ins

- Controllers for
  - Serial execution
  - Multi-threaded execution
  - MPI-based parallelism
  - TCP-based parallelism
  - Simulated-time parallel execution
  - Scripted execution (for testing)
- Adapters for several parallel programming styles
  - Serial evaluation (function call or coroutine)
  - Bulk evaluation
  - Promise objects
- Adapters for controlling/combining strategies (merge, retry, max eval)
- Tests and test support (including the `ChaosMonkeyStrategy`)

Intermission: POAP in Practice

Break to show some POAP test cases.

POAP and LibEnsemble

Fall 2018: Sabbatical semester at Argonne on LibEnsemble

- Middleware layer for ensemble calculations (opt, UQ)
- Primary target: big DOE HPC machines
- MPI only (when I started – not now)
- Shares some resemblance to POAP

Current project: Merge capabilities

PySOT: Python Surrogate Opt Toolbox

- Collection of surrogate optimization strategies for POAP.
- Comes with several optimization test problems.
- Easy to extend (we hope?!), several users already.

PySOT: Main components

1. Optimization problem specification.
2. Initial experimental design
3. Surrogate models
4. Adaptive sampling methods
   (DyCORS, SRBF, DDS, Gutmann, GA and gradient optimizers on surrogate)
5. Optimization strategies
   (serial/synchronous/asynchronous, continuous and integer variables, etc)
6. The PySOT GUI

Problem specification

Currently have:

- Domain (variable lower/upper bounds)
- Variable types (integer/continuous)
- Objective function

Would probably like:

- Objective pieces (least squares, multi-obj)
- Constraints
- Cost model
- Covariates
- Derivative interface

Experimental design

Currently have:

- LHD/SLHD: (Symmetric) Latin hypercube
- Full Factorial
- Box-Behnken

Would probably like: options involving prior info

Surrogate models

Currently have

- Interface: reset, add points, predict, deriv, stddev
- Radial basis function (RBF)
  - Allows: varying tail, kernel, regularization
  - Update cost is $O(kn^2)$ to add k points
- GP: Wraps scikit-learn
- Polynomial: Wraps scikit-learn
- MARS: Wraps py-earth

Would probably like

- Non-Gaussian stochastic modelt (BTG)
- Variable regularization / smoothing
- Independent implementations that re-use more work

Strategies

Currently have

- Stochastic RBF (SRBF)
- Dynamic coordinate search (DyCORS)
- Surrogate with Paretos center selection (SOP)
- Expected improvement (EI)
- Lower confidence bound (LCB)

Would probably like additional methods (KG, POUNDERS, ...)

Optimal Optimization

What do we want? Good points!

When do we want them? Now!

Different notions of efficient optimization

- Sample efficiency
  - Cost measured in number of evaluations
  - Want best possible design within eval budget
  - Usual measure for $p = 1$ workers
- Time efficiency
  - Cost measured in time
  - Want best possible design within time budget
  - Usual measure for $p = P > 1$ workers
  - Design Q: Parallelize within evals, or across?

In the Cloud

Cloud elasticity $\implies$ time and samples not proportional!

- Two budgets: money (or processor-hours) and time
- Money can pay for
  - Faster individual evaluations (up to a point)
  - More concurrent evaluations
- Latter is easier in cloud, but helps mostly for exploration
- Goal: Best expected design within both budgets

Seems like a hard problem — in the early stages of this work!

Call to Action

- Stellopt group
  - Trying out old optimizers / building new ones
  - Extend for multi-fidelity, multi-objective, etc
- Kernels and such
  - Faster versions of current stuff ("semi-expensive" case)
  - New interpolants (e.g. BTG)
- Anyone who wants
  - Smarter substrate
  - libEnsemble connection
  - Optimal optimization methods