Argonne NATIONAL LABORATORY

libEnsemble: A Library for Managing Dynamic Ensembles of Calculations

Stephen Hudson John-Luke Navarro Jeffrey Larson Stefan Wild

Argonne National Laboratory

July 7, 2022

What is libEnsemble

- libEnsemble is a Python toolkit for coordinating workflows of asynchronous and dynamic ensembles of calculations

What is libEnsemble

- libEnsemble is a Python toolkit for coordinating workflows of asynchronous and dynamic ensembles of calculations
- Developed to use massively parallel resources to accelerate the solution of design, decision, and inference problems

What is libEnsemble

- libEnsemble is a Python toolkit for coordinating workflows of asynchronous and dynamic ensembles of calculations
- Developed to use massively parallel resources to accelerate the solution of design, decision, and inference problems
- Developed to expand the class of problems that can benefit from increased computational concurrency levels

What is libEnsemble

- libEnsemble is a Python toolkit for coordinating workflows of asynchronous and dynamic ensembles of calculations
- Developed to use massively parallel resources to accelerate the solution of design, decision, and inference problems
- Developed to expand the class of problems that can benefit from increased computational concurrency levels
- libEnsemble uses a manager to allocate work to various workers

What is libEnsemble

- libEnsemble is a Python toolkit for coordinating workflows of asynchronous and dynamic ensembles of calculations
- Developed to use massively parallel resources to accelerate the solution of design, decision, and inference problems
- Developed to expand the class of problems that can benefit from increased computational concurrency levels
- libEnsemble uses a manager to allocate work to various workers
- A libEnsemble worker is the smallest indivisible unit. The number of libEnsemble workers is the maximum num to perform

libEnsemble requires of the user

`gen_f`: Generates inputs to `sim_f` and `alloc_f`

libEnsemble requires of the user

`gen_f`: Generates inputs to `sim_f` and `alloc_f`

`sim_f`: Evaluates a simulation (any user-defined function) using input defined by `gen_f`

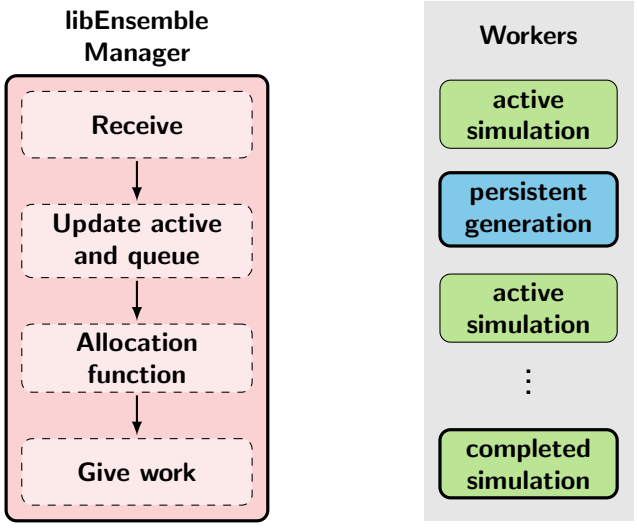
libEnsemble requires of the user

`gen_f`: Generates inputs to `sim_f` and `alloc_f`

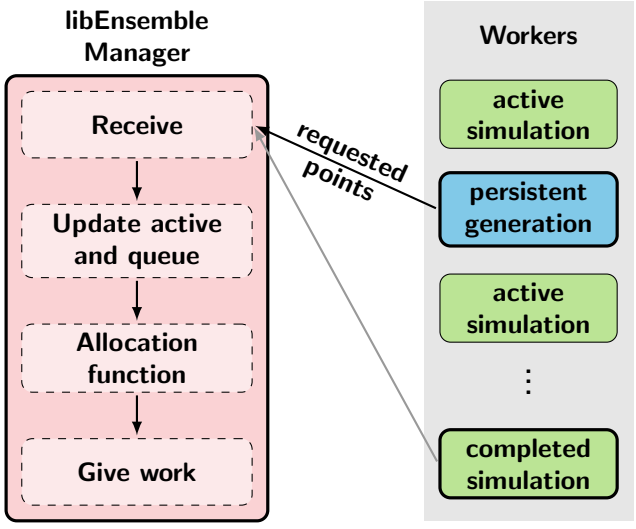
`sim_f`: Evaluates a simulation (any user-defined function) using input defined by `gen_f`

`alloc_f`: As workers become available, decides if a `sim_f` or `gen_f` should be called (and with what input/resources)

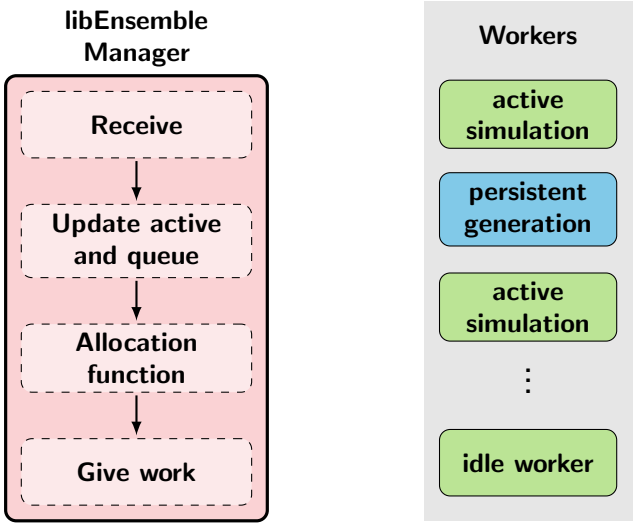
libEnsemble overview



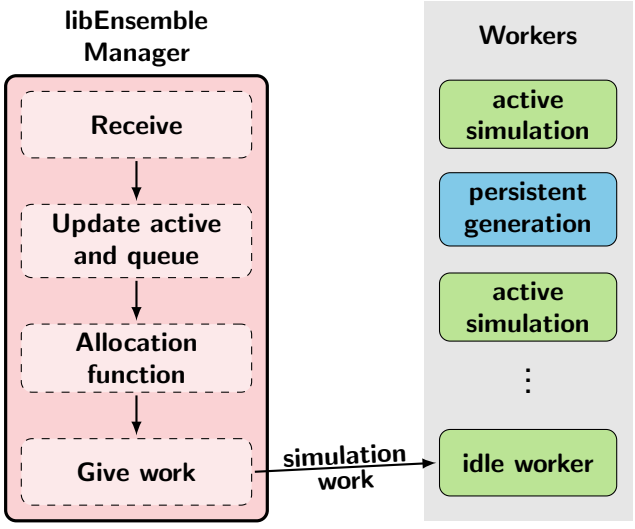
libEnsemble overview



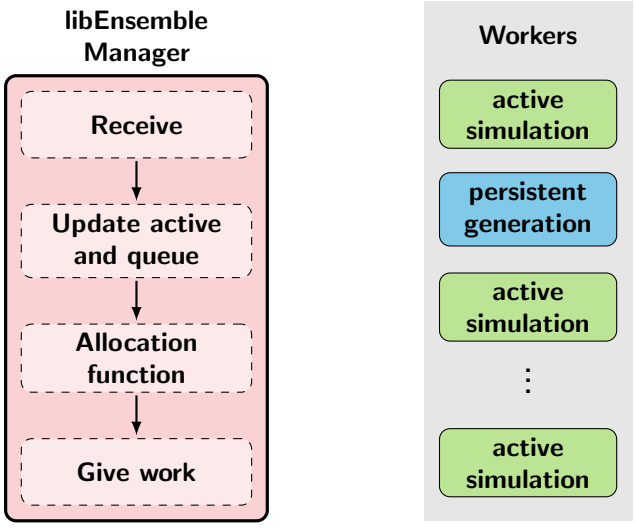
libEnsemble overview



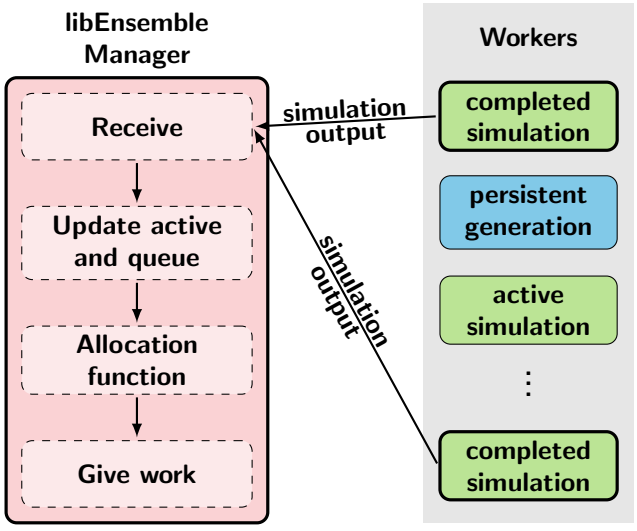
libEnsemble overview



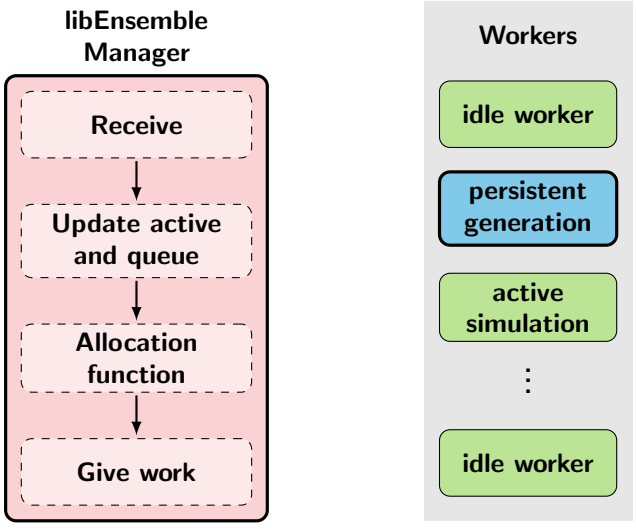
libEnsemble overview



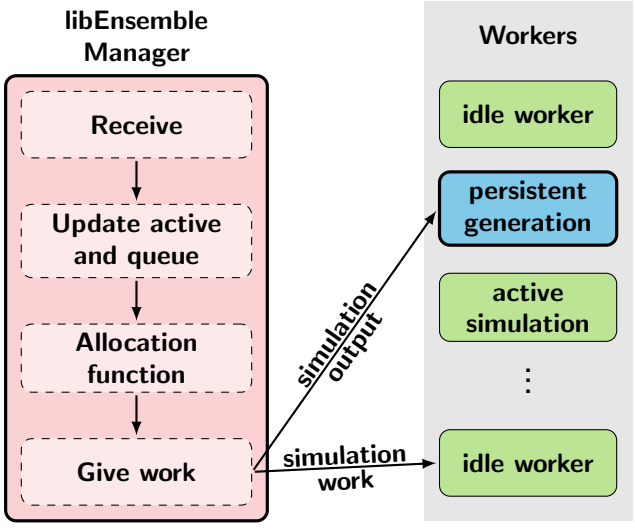
libEnsemble overview



libEnsemble overview



libEnsemble overview



`libEnsemble` dependencies

- Python 3.7+, NumPy, psutil, setuptools

`libEnsemble` dependencies

- Python 3.7+, NumPy, psutil, setuptools

- Communications in `libEnsemble` can be done using
  - MPI
  - Multiprocessing
  - TCP

`libEnsemble` dependencies

- Python 3.7+, NumPy, psutil, setuptools
- Communications in `libEnsemble` can be done using
  - MPI
  - Multiprocessing
  - TCP
- The repo contains example `gen_f`/`sim_f` functions that require NLopt, PETSc, SciPy, Tasmanian, etc.

Possible user requests of `libEnsemble`

- `sim_f`/`gen_f` calculations can employ/access parallel resources

Possible user requests of `libEnsemble`

- `sim_f`/`gen_f` calculations can employ/access parallel resources
  - Places requirements on user's environment and simulation/generation function

Possible user requests of `libEnsemble`

- `sim_f`/`gen_f` calculations can employ/access parallel resources
  - Places requirements on user's environment and simulation/generation function

- Termination of unresponsive simulation/generation calculations

Possible user requests of `libEnsemble`

- `sim_f`/`gen_f` calculations can employ/access parallel resources
  - Places requirements on user's environment and simulation/generation function
- Termination of unresponsive simulation/generation calculations
- Termination based on intermediate simulation/generation output

Possible user requests of `libEnsemble`

- Maintenance of calculation history, logging, and performance measures

Possible user requests of `libEnsemble`

- Maintenance of calculation history, logging, and performance measures
- Simulation/generation checkpoint and restart

Possible user requests of `libEnsemble`

- Maintenance of calculation history, logging, and performance measures
- Simulation/generation checkpoint and restart
- Execution on multiple LCFs (Summit/OLCF, Cori/NERSC, Theta/ALCF)

Possible user requests of `libEnsemble`

- Maintenance of calculation history, logging, and performance measures
- Simulation/generation checkpoint and restart
- Execution on multiple LCFs (Summit/OLCF, Cori/NERSC, Theta/ALCF)
- Thousands of concurrent workers

Why would you want to use `libEnsemble`?

- Easily take serial code and start running in parallel

Why would you want to use `libEnsemble`?

- Easily take serial code and start running in parallel
- Add another level of parallelism to a simulation that no longer scales

Why would you want to use `libEnsemble`?

- Easily take serial code and start running in parallel
- Add another level of parallelism to a simulation that no longer scales
- Don't have to write your own tracking code, `libEnsemble` will tell you which runs fail and start other work

Why would you want to use `libEnsemble`?

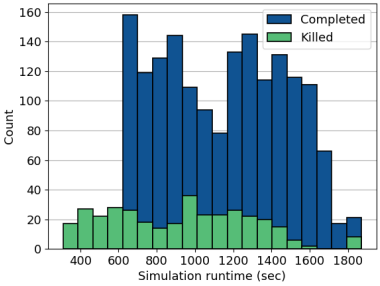
- Easily take serial code and start running in parallel
- Add another level of parallelism to a simulation that no longer scales
- Don't have to write your own tracking code, `libEnsemble` will tell you which runs fail and start other work
- Don't have to write your own kills, just complete `libEnsemble` templates

Why would you want to use `libEnsemble`?

- Easily take serial code and start running in parallel
- Add another level of parallelism to a simulation that no longer scales
- Don't have to write your own tracking code, `libEnsemble` will tell you which runs fail and start other work
- Don't have to write your own kills, just complete `libEnsemble` templates
- Want to add concurrency to a generator (e.g., multiple local optimizers.)

Use cases

- A user wants to evaluate a simulation at parameters randomly sampled from a domain of allowed values
- Many parameter sets will cause the simulation to fail
- `libEnsemble` can stop unresponsive evaluations, and recover computational resources for future evaluations
- `gen_f` can update the sampling after discovering regions where evaluations of simulation fail



Use cases

- To optimize a function that depends on a simulation
- The simulation is already using parallel resources, but not a large fraction of some computer
- `libEnsemble` can coordinate the concurrent evaluation of the simulation `sim_f` at various parameter values and `gen_f` would return candidate parameter values (possibly after each `sim_f` output)

Use cases

- To optimize a function that depends on a simulation
- The simulation is already using parallel resources, but not a large fraction of some computer
- `libEnsemble` can coordinate the concurrent evaluation of the simulation `sim_f` at various parameter values and `gen_f` would return candidate parameter values (possibly after each `sim_f` output)

```
git clone https://github.com/Libensemble/libensemble.git
```

APOSMM

Asynchronously Parallel Optimization Solver for Finding Multiple Minima

- Identify distinct, “high-quality”, local optima of some `sim_f` output

APOSMM

Asynchronously Parallel Optimization Solver for Finding Multiple Minima

- Identify distinct, "high-quality", local optima of some `sim_f` output
- Derivatives of `sim_f` may or may not be available

APOSMM

Asynchronously Parallel Optimization Solver for Finding Multiple Minima

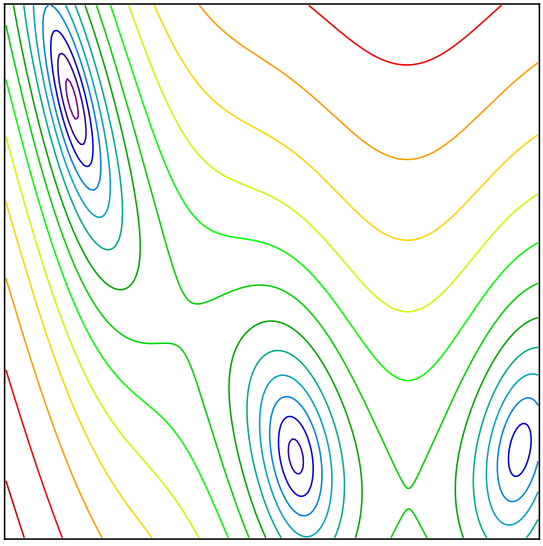
- Identify distinct, "high-quality", local optima of some `sim_f` output
- Derivatives of `sim_f` may or may not be available
- `sim_f` may use parallel resources, but not the entire machine

APOSMM

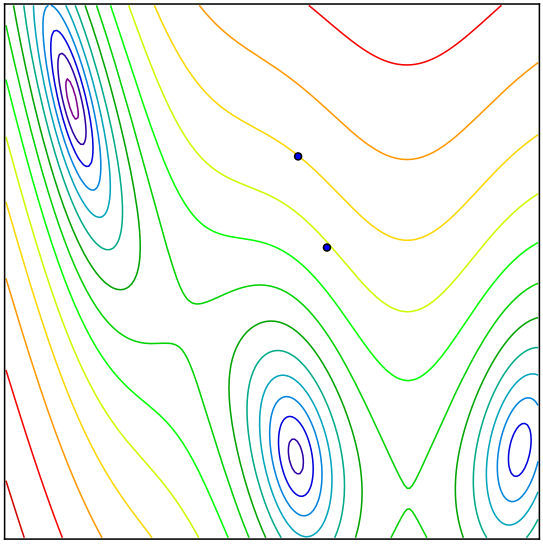
Asynchronously Parallel Optimization Solver for Finding Multiple Minima

- Identify distinct, "high-quality", local optima of some `sim_f` output
- Derivatives of `sim_f` may or may not be available
- `sim_f` may use parallel resources, but not the entire machine
- Possibly want a specialized local optimization method

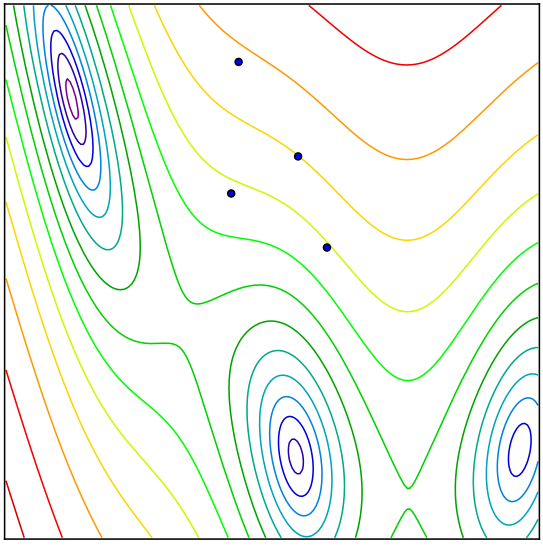
APOSMM



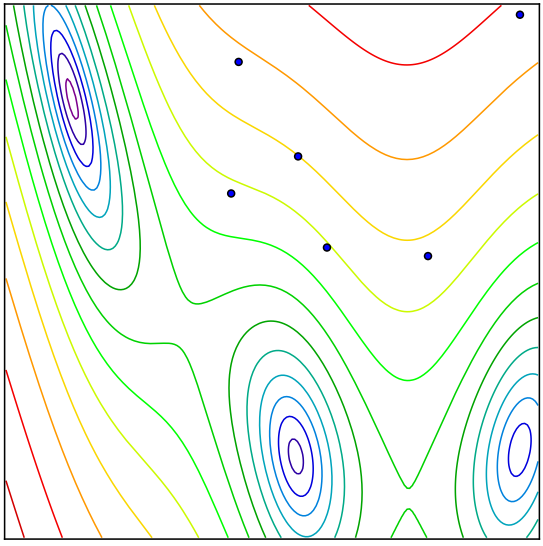
APOSMM



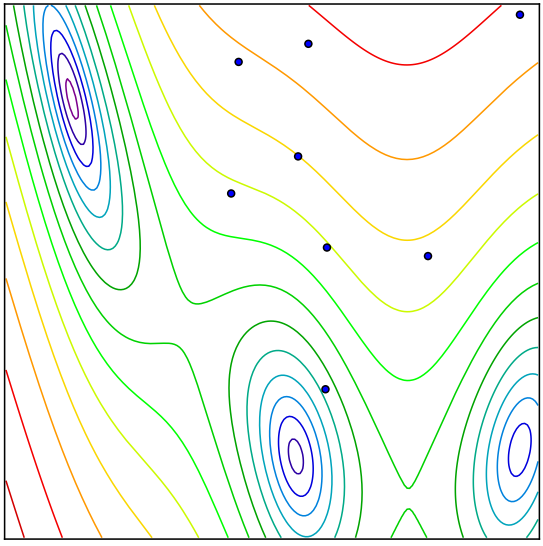
APOSMM



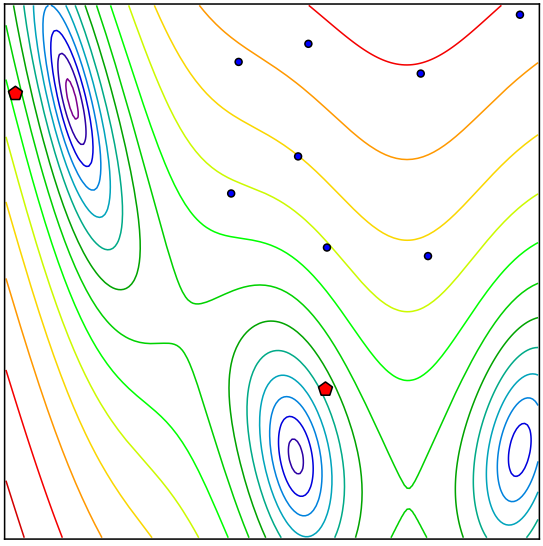
APOSMM



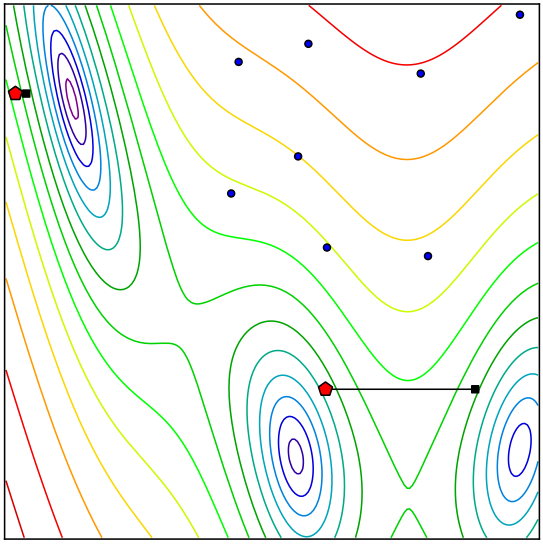
APOSMM



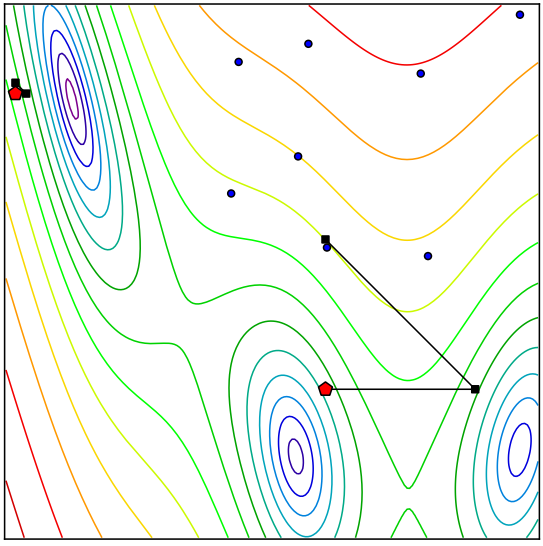
APOSMM



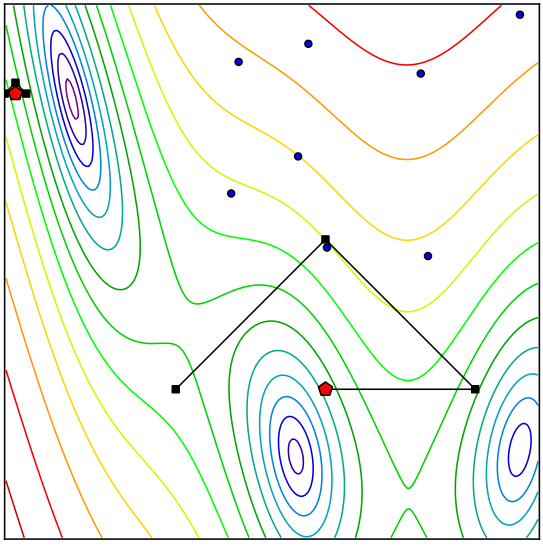
APOSMM



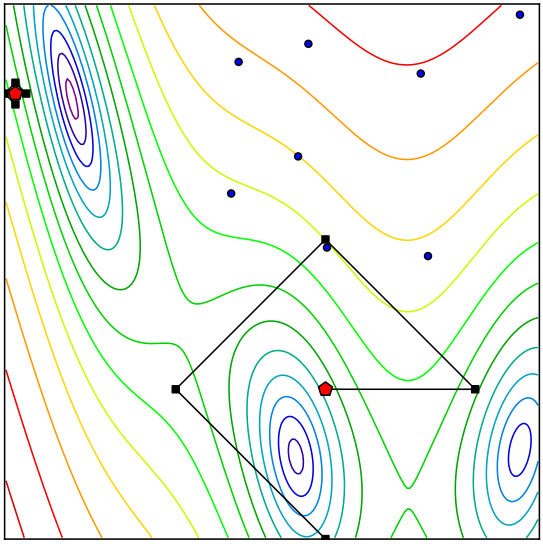
APOSMM



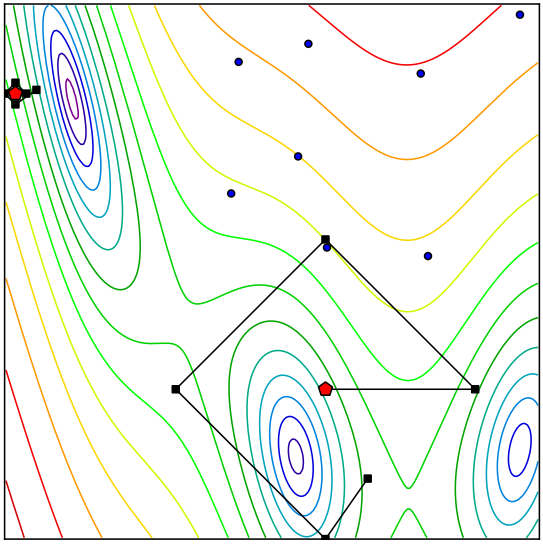
APOSMM



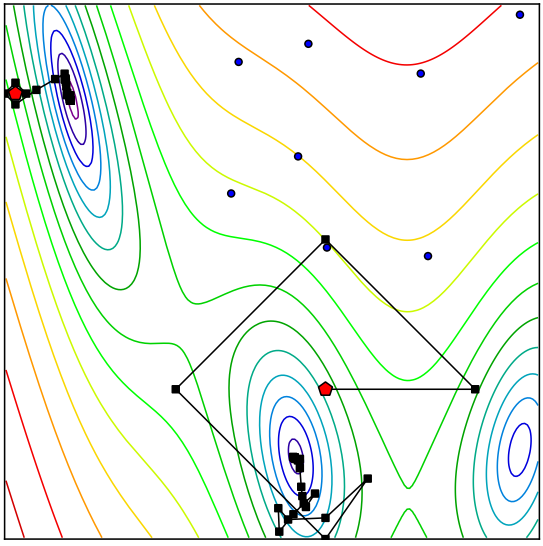
APOSMM



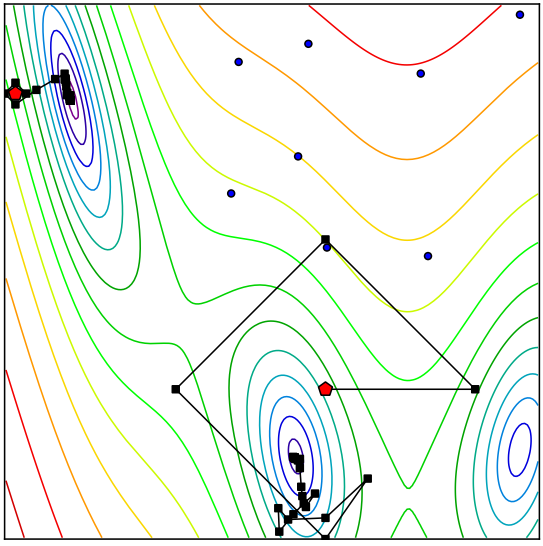
APOSMM



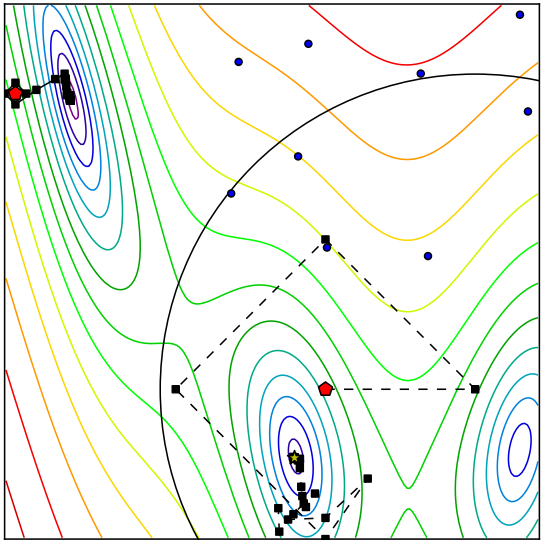
APOSMM



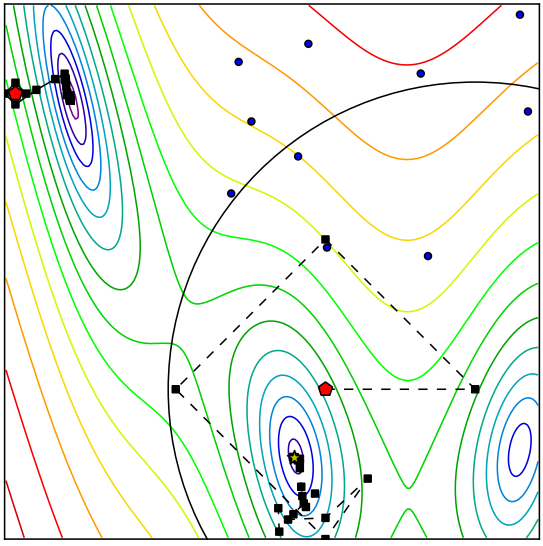
APOSMM



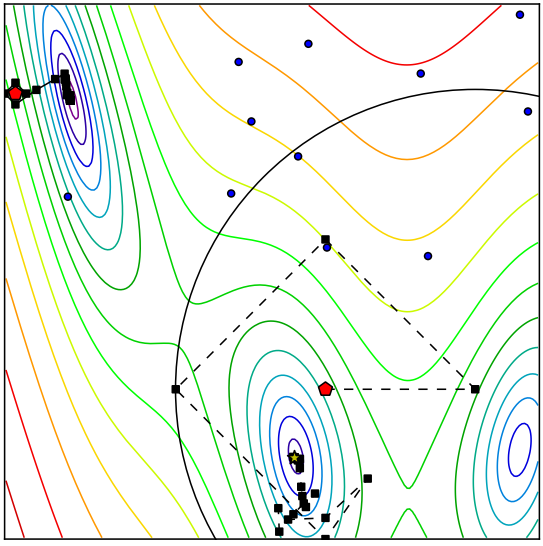
APOSMM



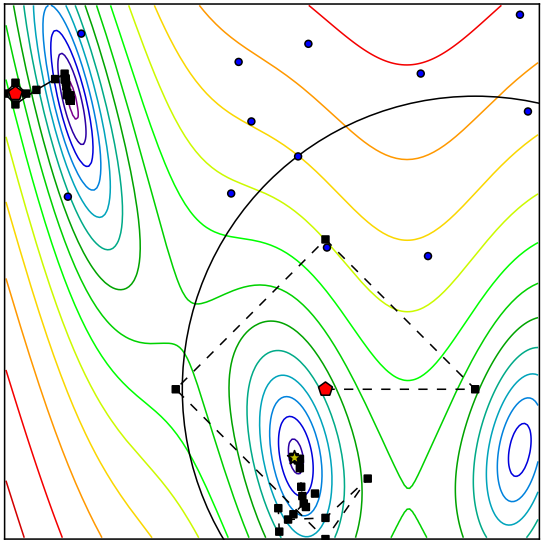
APOSMM



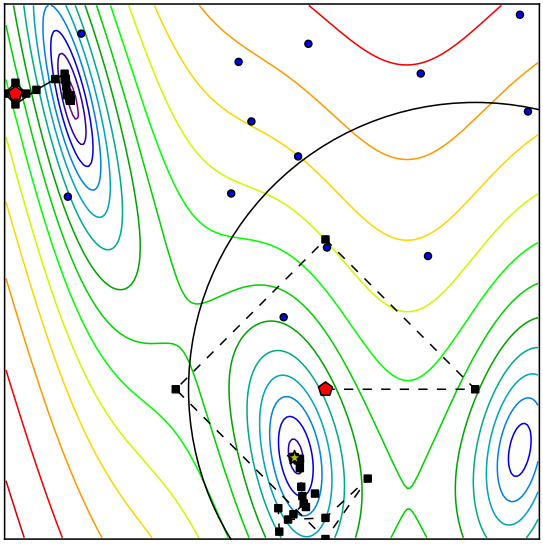
APOSMM



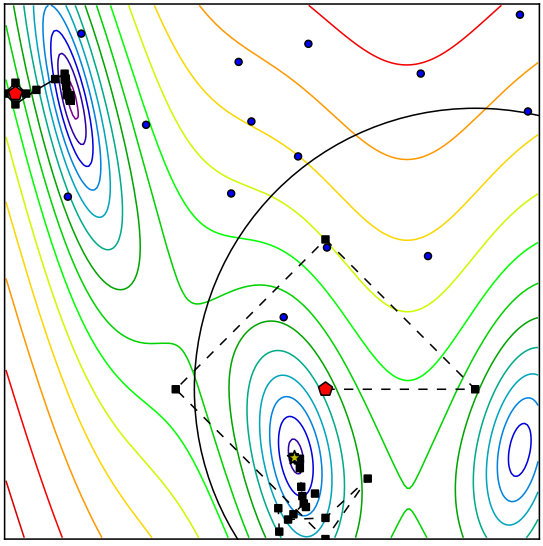
APOSMM



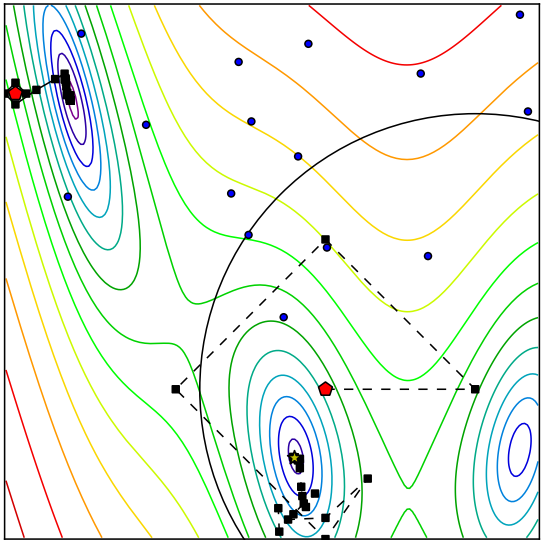
APOSMM



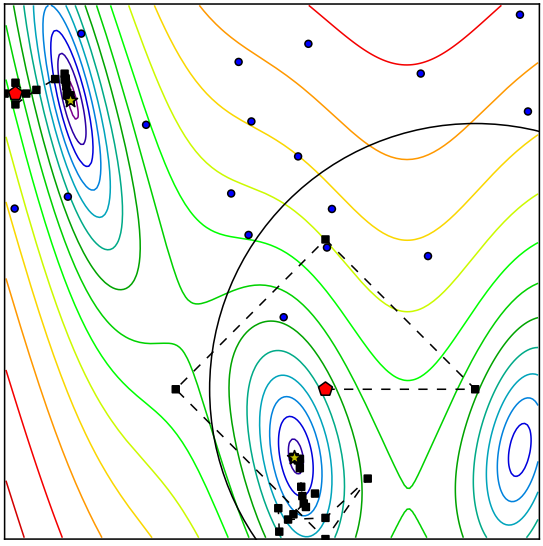
APOSMM



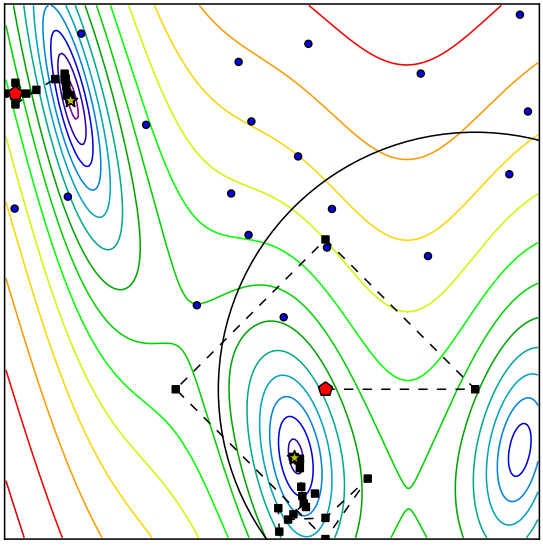
APOSMM



APOSMM



APOSMM



APOSMM

```
gen_out = [
    ("x", float, n),
    ("x_on_cube", float, n),
    ("sim_id", int),
    ("local_min", bool),
    ("local_pt", bool),
]

gen_specs = {
    "gen_f": aposmm,
    "persis_in": ["f", "fvec", "x"],
    "out": gen_out,
    "user": {
        "lb": (-2 - np.pi / 10) * np.ones(n),
        "ub": 2 * np.ones(n),
        "initial_sample_size": 100,
        "localopt_method": "dfols",
        "components": m,
        "dfols_kwargs": {
            "do_logging": False,
            "rhoend": 1e-5,
            "user_params": {
                "model.abs_tol": 1e-10,
                "model.rel_tol": 1e-4,
            },
        },
    },
}
```