Xopt: Flexible Black Box Optimization of Simulations and Experiments

Ryan Roussel

rroussel@slac.stanford.edu

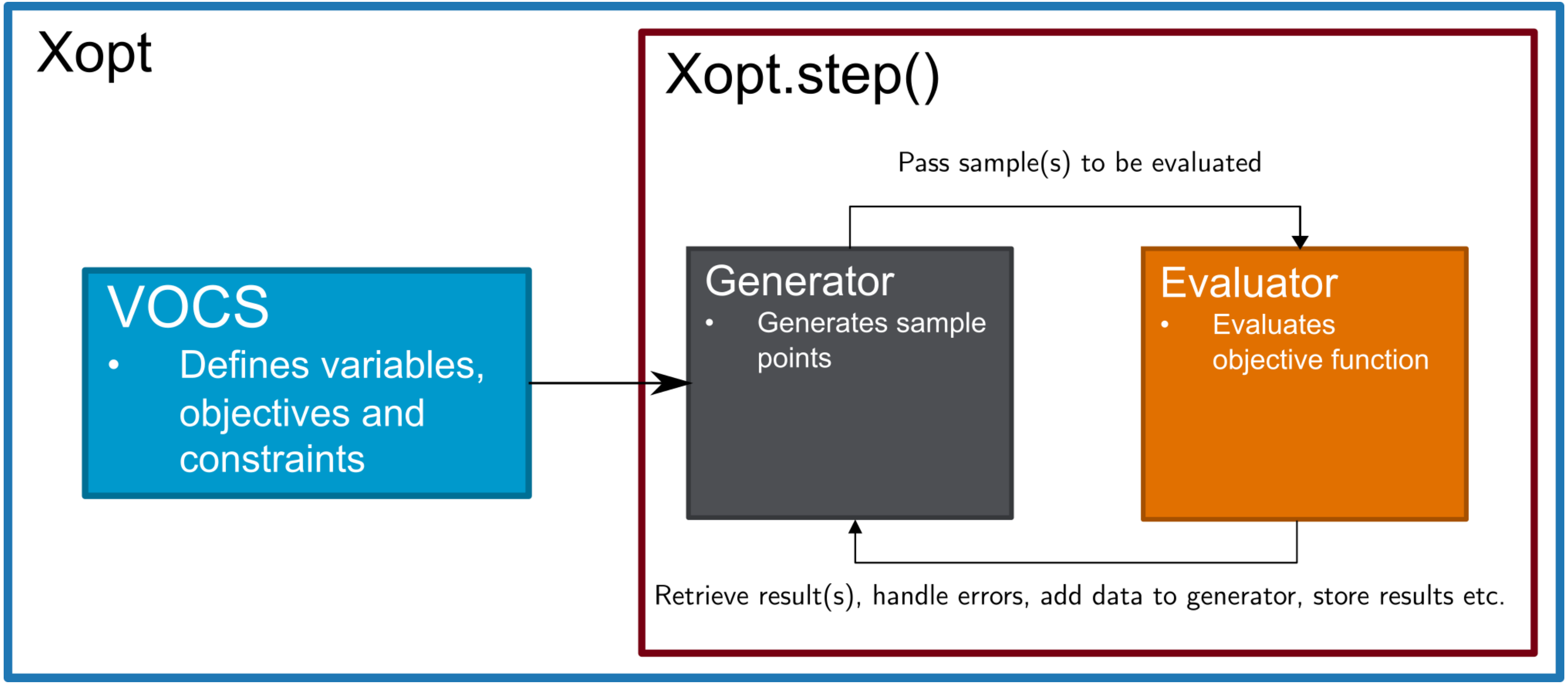
U.S. DEPARTMENT OF ENERGY | Stanford University

SLAC NATIONAL ACCELERATOR LABORATORY

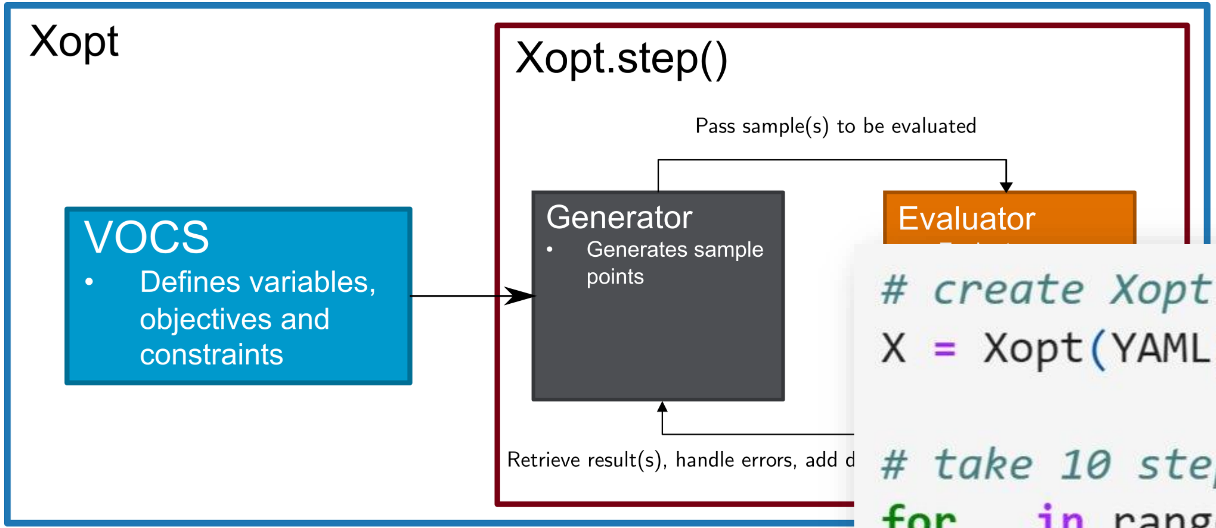
What is Xopt?

- Flexible **framework** for optimization of arbitrary problems using python
- **Independent** of problem type (simulation or experiment)
- **Independent** of optimization algorithm + easy to incorporate custom algorithms
- **Easy to use** text interface and/or advanced customized use for professionals

Xopt structure



Xopt usage




```python
# create Xopt object.
X = Xopt(YAML)

# take 10 steps and view data
for _ in range(10):
    X.step()

X.data
```

Xopt input

Via YAML file (validated by pydantic):


```yaml
xopt:
    max_evaluations: 6400

generator:
    name: cnsga
    population_size: 64
    population_file: test.csv
    output_path: .

evaluator:
    function: xopt.resources.test_functions.tnk.evaluate_TNK
    function_kwargs:
      raise_probability: 0.1

vocs:
    variables:
        x1: [0, 3.14159]
        x2: [0, 3.14159]
    objectives: {y1: MINIMIZE, y2: MINIMIZE}
    constraints:
        c1: [GREATER_THAN, 0]
        c2: [LESS_THAN, 0.5]
    linked_variables: {x9: x1}
    constants: {a: dummy_constant}
```


Via python code:

```python
evaluator = Evaluator(…)
generator = CNSGAGenerator(…)
vocs = MyVOCS(…)

X = Xopt(
        evaluator=evaluator,
        generator=generator,
        vocs=vocs
)
```

Evaluator specification

- Python function must accept/return dicts
- Input dict must have **at least** the keys specified in vocs variables/constants (see next slide)
  - You can include extra keyword args if needed!
- Output dict must have **at least** the keys specified in objectives/constraints (see next slide)
  - The function can output extra keys to be tracked!
- Evaluators inherit directly from python concurrent.futures so you can use this for parallel evaluation (see /xopt/docs/examples/basic/xopt_parallel)

```
xopt:
    max_evaluations: 6400

generator:
    name: cnsga
    pop
    pop
    output_path: .

evaluator:
    function: xopt.resources.test_functions.tnk.evaluate_TNK
    function_kwargs:
      raise_probability: 0.1

vocs:
    variables:
        x1: [0, 3.14159]
        x2: [0, 3.14159]
    objectives: {y1: MINIMIZE, y2: MINIMIZE}
    constraints:
        c1: [GREATER_THAN, 0]
        c2: [LESS_THAN, 0.5]
    linked_variables: {x9: x1}
    constants: {a: dummy_constant}
```

```
evaluate(inputs: dict) -> dict
```

Evaluator specification

- Python function must accept/return dicts
- Input dict must have **at least** the keys specified in vocs variables/constants (see next slide)
  - You can include extra keyword args if needed!
- Output dict must have **at least** the keys specified in objectives/constraints (see next slide)
  - The function can output extra keys to be tracked!
- Evaluators inherit directly from python concurrent.futures so you can use this for parallel evaluation (see /xopt/docs/examples/basic/xopt_parallel)

```
evaluate(inputs: dict) -> dict
```

```
from epics import caget, caput, cainfo
import time

outputs = ["XRMS","YRMS"]
def make_epics_measurement(input_dict):
  # set inputs
  for name, val in input_dict.items():
    caput(name, val)

  # wait for inputs to settle
  time.sleep(1)

  # get output values, current time
  output_dict = caget_many(outputs)
  output_dict["time"] = time.time()

  # compute geometeric avg of beamsizes
  output_dict["RMS"] = (
    output_dict["XRMS"]*\
    output_dict["YRMS"]
  )**0.5

  return output_dict
```

VOCS Specification

- Variables: input domain limits and names
- Objectives: objective names and goals (minimize/maximize)
- Constraints: constraint names and conditions (greater than/less than)
- Constants: constant values


```yaml
xopt:
    max_evaluations: 6400

generator:
    name: cnsga
    population_size: 64
    population_file: test.csv
    output_path: .

evaluator:
    function: xopt.resources.test_functions.tnk.evaluate_TNK
    function_kwargs:
      raise_probability: 0.1
```

```yaml
vocs:
    variables:
        x1: [0, 3.14159]
        x2: [0, 3.14159]
    objectives: {y1: MINIMIZE, y2: MINIMIZE}
    constraints:
        c1: [GREATER_THAN, 0]
        c2: [LESS_THAN, 0.5]
    linked_variables: {x9: x1}
    constants: {a: dummy_constant}
```

Generator specification

- Use built-in generators by name
  - optimization algorithms:
    - `cnsga` Continuous NSGA-II with constraints.
    - `bayesian_optimization` Single objective Bayesian optimization (w/ or w/o constraints, serial or parallel).
    - `mobo` Multi-objective Bayesian optimization (w/ or w/o constraints, serial or parallel).
    - `bayesian_exploration` Bayesian exploration.
  - sampling algorithms:
    - `random sampler`
- Each generator has its own specific options
- Locate the default options in the docs or via

```python
from xopt.utils import get_generator_and_defaults
gen, options = get_generator_and_defaults("upper_confidence_bound")
print(yaml.dump(options.dict()))
```

```
acq:
  beta: 2.0
  monte_carlo_samples: 512
  proximal_lengthscales: null
model:
  use_conservative_prior_lengthscale: false
  use_conservative_prior_mean: false
  use_low_noise_prior: false
n_initial: 3
optim:
  num_restarts: 5
  raw_samples: 20
  sequential: true
```

```yaml
xopt:
    max_evaluations: 6400

generator:
    name: cnsga
    population_size: 64
    population_file: test.csv
    output_path: .

evaluator:
    function: xopt.resources.test_functions.tnk.evaluate_TNK
    function_kwargs:
      raise_probability: 0.1

vocs:
    variables:
        x1: [0, 3.14159]
        x2: [0, 3.14159]
    objectives: {y1: MINIMIZE, y2: MINIMIZE}
    constraints:
        c1: [GREATER_THAN, 0]
        c2: [LESS_THAN, 0.5]
    linked_variables: {x9: x1}
    constants: {a: dummy_constant}
```

Data storage

- Data is stored by xopt in the `data` attribute
- Set dump_file in xopt options to dump data and xopt config to yaml file after every evaluation step
- Dump file can be used to restart xopt

```
# view the data
X.data
```

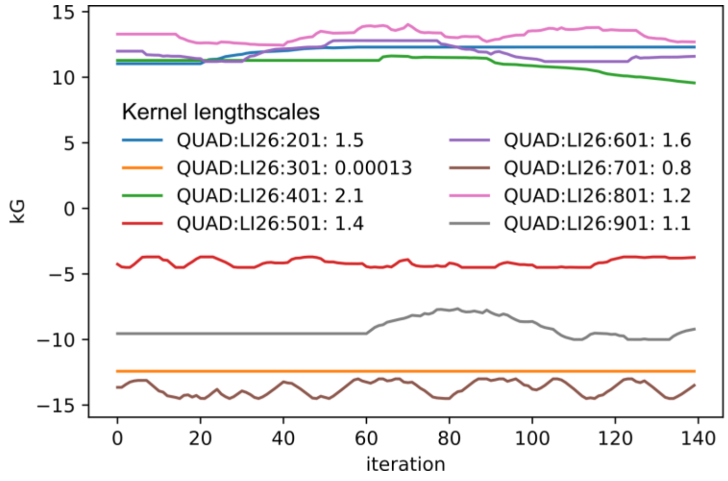
	x1	x2	y1	y2	c1	c2	some_array	xopt_error	xopt_error_str	a
1	1.000000	0.750000	1.000000	0.750000	0.626888	0.312500	[1, 2, 3]	False		NaN
2	0.750000	1.000000	0.750000	1.000000	0.626888	0.312500	[1, 2, 3]	False		NaN
3	0.796389	0.807321	0.796389	0.807321	0.186596	0.182292	[1, 2, 3]	False		dummy_constant
4	0.871085	0.943368	0.871085	0.943368	0.568348	0.334279	[1, 2, 3]	False		dummy_constant
5	1.067732	0.797750	1.067732	0.797750	0.843056	0.410974	[1, 2, 3]	False		dummy_constant
6	0.995019	0.879029	0.995019	0.879029	0.707805	0.388707	[1, 2, 3]	False		dummy_constant
7	0.803822	1.022336	0.803822	1.022336	0.724145	0.365142	[1, 2, 3]	False		dummy_constant
8	0.656282	0.952071	0.656282	0.952071	0.434474	0.228792	[1, 2, 3]	False		dummy_constant
9	0.566763	0.935263	0.566763	0.935263	0.271920	0.193911	[1, 2, 3]	False		dummy_constant
10	0.547152	1.008562	0.547152	1.008562	0.326474	0.260859	[1, 2, 3]	False		dummy_constant
11	0.617813	1.081140	0.617813	1.081140	0.594283	0.351603	[1, 2, 3]	False		dummy_constant
12	0.491363	1.027666	0.491363	1.027666	0.231751	0.278506	[1, 2, 3]	False		dummy_constant

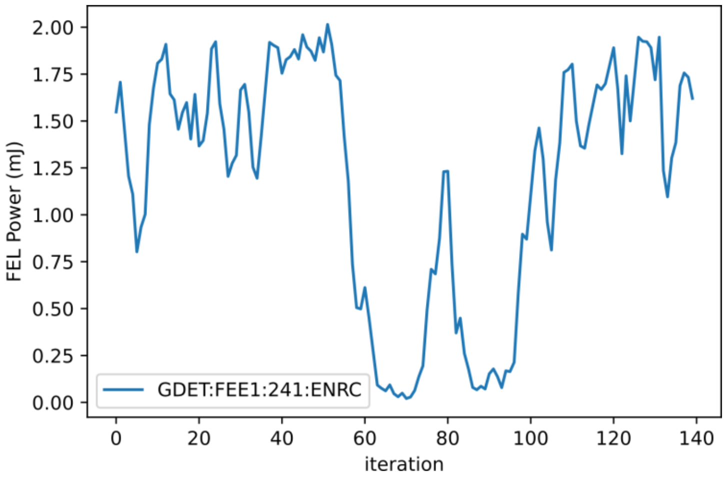
```
xopt:
    dump_file: dump.yaml
```

```
config = yaml.safe_load(open("dump.yaml"))
X2 = Xopt(config)
print(X2.options)
print(X2.generator)
print(X2.evaluator)
```

Example Application: LCLS FEL Power Characterization

- **Proximal biasing** to reduce exploration step size and **constraints** to prevent charge loss.
- **Custom evaluate function** captures 80th percentile FEL power over 100 shots.
- Data stored in Pandas DataFrame objects, exported to text file with Xopt configuration
- FEL sensitivity is captured in the GP model lengthscales inside the generator object.
- Entirely executed from an **interactive Jupyter notebook.**





Badger: Missing Optimizer in the Accelerator Control Room

Zhe Zhang

zhezhang@slac.stanford.edu

U.S. DEPARTMENT OF ENERGY | Stanford University

SLAC NATIONAL ACCELERATOR LABORATORY

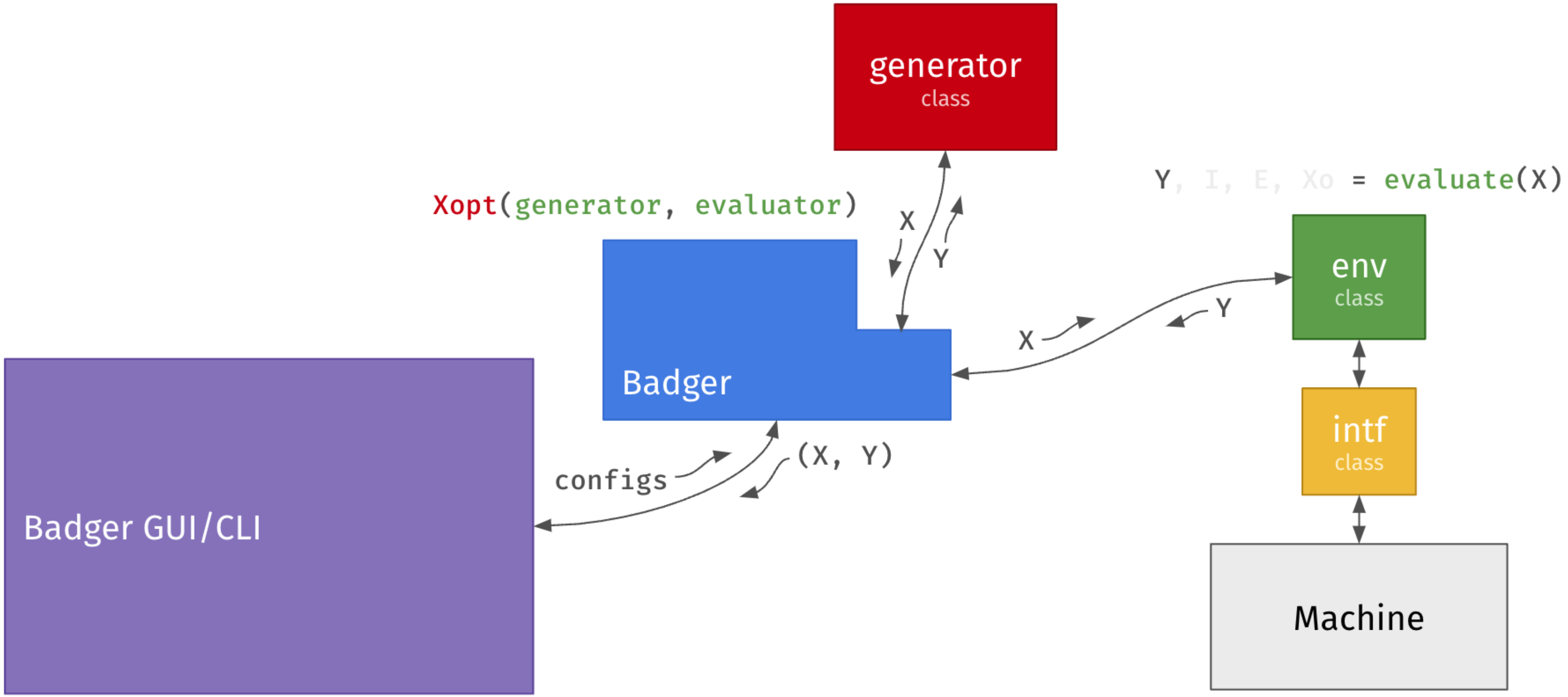
What is Badger?

- **Optimization interface** between users and machine, the spiritual successor to Ocelot-optimizer, powered by Xopt
- **Easy to use** one click/cmd to re-run an optimization task
- **Fast to extend** plugin-based, create your own custom environment in minutes
- **Multiple modes** use Badger as a python library, a command line tool, or a GUI application

What can Badger do?

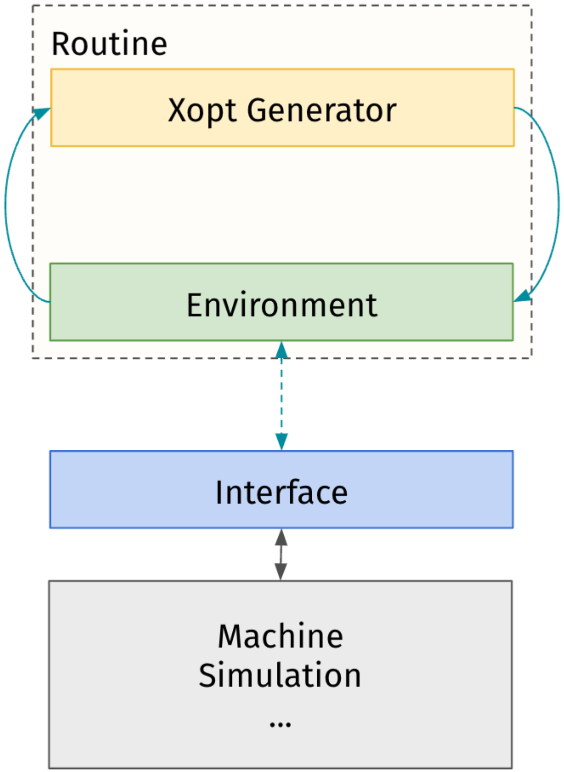
- **General features**
  - Control the optimization flow (pause/resume/terminate)
  - Monitor/browse the runs
  - Archive/explore the data
- **Accelerator control room (ACR) oriented features**
  - Send run summary to the logbook
  - Jump/set to optimal solution
  - Recover machine state after run
  - Support soft/hard constraints & tracked states
  - Preserve all raw data
  - *Continue/rollback a run (planned)

Badger architecture



Badger architecture

- **Generator** algorithms provided by Xopt
- **Environment** defines the observations and the variables
- **Interface** layer between the environment and the machine, all data in a run would flow through the interface
- **Routine** contains complete information about one optimization task



GUI mode (browse/run a routine)

Search bar

Routine filters

Routine list

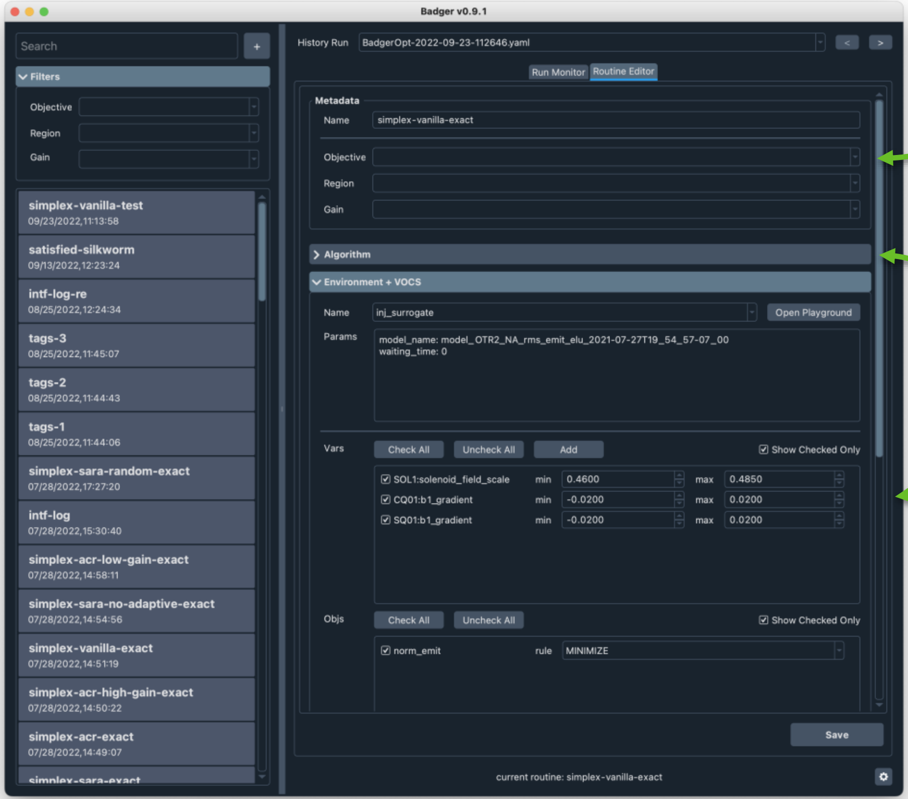
Historical run navigator

Run monitor

Control bar

Data browser

GUI mode (create/edit/view a routine)



Routine configs

Algorithm configs

Environment/VOCS configs

CLI mode

- `badger` to get general information
- `badger algo/env/intf` to list all the algorithms/environments/interfaces
- `badger algo/env/intf NAME` to investigate a specific plugin
- `badger routine` to list all saved routines
- `badger routine NAME` to review the routine
- `badger routine NAME -r` to run the routine
- `badger run` to create and run a routine
- `badger install env/intf` to list/install the environment/interface plugins

```
fish /
~ badger routine intf-log -r
Please review the routine to be run:

=== Optimization Routine ===
name: intf-log
algo: silly
env: silly
algo_params:
  start_from_current: true
  max_iter: 42
env_params: null
config:
  variables:
    - q1: 0.0 -> 1.0
    - q2: 0.0 -> 1.0
    - q3: 0.0 -> 1.0
    - q4: 0.0 -> 1.0
  objectives:
    - l2: MINIMIZE
  constraints: null
  states: null
  domain_scaling: null

Proceed ([y]/n)?

|    iter    |     l2     |     q1     |     q2     |     q3     |     q4     |
--------------------------------------------------------------------------------
|  1         |  0.0       |  0.0       |  0.0       |  0.0       |  0.0       |
|  2         |  1.372     |  0.8838    |  0.6329    |  0.07155   |  0.8343    |
|  3         |  0.864     |  0.2203    |  0.7632    |  0.07581   |  0.3312    |
|  4         |  1.384     |  0.06517   |  0.4959    |  0.9432    |  0.8809    |
|  5         |  1.043     |  0.1173    |  0.2418    |  0.6492    |  0.7702    |
|  6         |  1.172     |  0.05347   |  0.6336    |  0.9677    |  0.182     |
|  7         |  1.606     |  0.9123    |  0.7953    |  0.6401    |  0.839     |
|  8         |  0.6882    |  0.3282    |  0.2254    |  0.05388   |  0.5587    |
|  9         |  0.5209    |  0.128     |  0.04482   |  0.4601    |  0.203     |
|  10        |  1.305     |  0.7387    |  0.9853    |  0.006447  |  0.4321    |
|  11        |  1.378     |  0.5078    |  0.2893    |  0.7706    |  0.9816    |
|  12        |  1.206     |  0.1058    |  0.5512    |  0.8809    |  0.6024    |
|  13        |  1.272     |  0.8485    |  0.1888    |  0.2865    |  0.8837    |
|  14        |  1.287     |  0.8688    |  0.8435    |  0.3678    |  0.2327    |
|  15        |  1.579     |  0.7412    |  0.5197    |  0.8937    |  0.9358    |
|  16        |  1.295     |  0.6825    |  0.1887    |  0.5444    |  0.9381    |
|  17        |  1.318     |  0.6172    |  0.1534    |  0.7868    |  0.8455    |
|  18        |  0.42      |  0.3756    |  0.04849   |  0.1375    |  0.1187    |
|  19        |  1.467     |  0.3577    |  0.2782    |  0.9826    |  0.9911    |
|  20        |  1.461     |  0.8778    |  0.6194    |  0.9771    |  0.1608    |
|  21        |  0.7706    |  0.6571    |  0.01575   |  0.06577   |  0.3969    |
|  22        |  0.7706    |  0.2097    |  0.2199    |  0.6972    |  0.1239    |
^COptimization paused. Press Enter to resume or Ctrl/Cmd + C to terminate: ^C
================================================================================
~
```

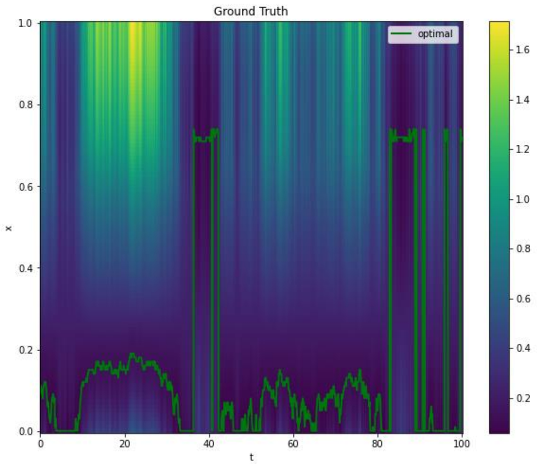
API mode

SLAC

- **Use `get_algo` to get an algorithm**
  - Unified user interface/consistent user experience
  - No need to deal with algorithm setup
- **Use `get_env` to load an environment**
  - Set variables to the environment and get observations from it
  - Embed the env in the workflow
  - No need to setup the simulation/experiment configs again and again

```python
[1]: from badger.factory import get_env, get_algo

Environment, configs = get_env('inj_surrogate')
env = Environment(configs['params'], None)
env.load_model()

def target(x, t):
    # x is normalized

    _X = denorm(torch.hstack((drift(t), mapping_x(x))))
    ref_point = torch.from_numpy(env.ref_point[0]).to(dtype)
    X_in = ref_point.repeat(len(t), 1)
    indices = [env.model.loc_in[var] for var in list_vars]
    X_in[:, indices] = _X
    Y_out = torch.from_numpy(env.model.pred_machine_units(X_in))

    y_1 = Y_out[:, env.model.loc_out['sigma_x']] * 1e3
    y_2 = Y_out[:, env.model.loc_out['sigma_y']] * 1e3
    y = y_1

    return y.reshape(-1, 1)
```

...



Create a custom environment

- **Think about a list of variables/observations that could be involved in your optimization problem**
  - Variables are the tuning knobs
  - Observations are the measurements – including objectives, constraints, or system states to be tracked
- **Inherit the base Environment class and implements:**
  - `name` name of the environment
  - `list_vars` return a list of all the variables
  - `list_obses` return a list of all the observations
  - `_get_var` get the current value of the variable
  - `_set_var` set the variable to the given value
  - `_get_obs` get the current value of the observation

```
class Environment(ABC):

    @property
    @abstractmethod
    def name(self) -> str:
        pass

    @abstractmethod
    def __init__(self, interface: Interface, params=None):
        self.interface = interface
        self.params = merge_params(self.get_default_params(), params)

    # List all available variables
    @staticmethod
    @abstractmethod
    def list_vars() -> List[str]:
        pass

    # List all available observations
    @staticmethod
    @abstractmethod
    def list_obses() -> List[str]:
        pass

    # Get the default params of the environment
    @staticmethod
    def get_default_params() -> dict:
        return None

    # Get current variable
    # Unsafe version (var won't be checked beforehand)
    @abstractmethod
    def _get_var(self, var: str):
        pass

    # Set variable
    # Unsafe version
    @abstractmethod
    def _set_var(self, var: str, x):
        pass
```

Get started

```
pip install badger-opt
```

or

```
conda install badger-opt
```

then

```
badger –ga
```

for more information, please check out
https://slac-ml.github.io/Badger/docs/getting-started/installation

Tutorial

Run and save an optimization

Create a yaml file under your `pwd` (where you would run an optimization with Badger) with the following content:

config.yaml

```yaml
variables:
  - x2
objectives:
  - c1
```

To run and save an optimization, run:

```
badger run -a silly -e TNK -c config.yaml -s helloworld
```

Badger will ask you to review the optimization routine:

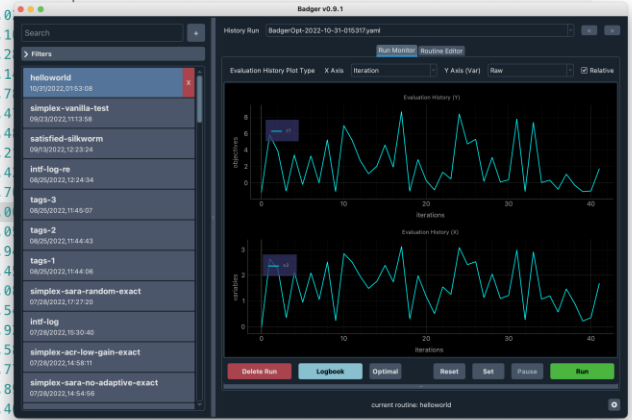
output

```
Please review the routine to be run:

=== Optimization Routine ===
name: mottled-sloth
algo: silly
env: TNK
algo_params:
  dimension: 1
  max_iter: 42
env_params: null
config:
  variables:
    - x2: 0 -> 3.14159
  objectives:
    - c1: MINIMIZE
  constraints: null

Proceed ([y]/n)?
```

Hit return to confirm. Badger will print out a table of all the evaluated solutions along the run:

output

```
|    iter    |     c1     |     x2     |
----------------------------------------
|  1         | -1.094     | 0.07432    |
|  2         |  3.563     | 2.159      |
|  3         |  8.749     | 3.138      |
|  4         |  5.351     | 2.54       |
|  5         |  8.17      | 3.045      |
|  6         |  6.536     | 2.763      |
|  7         |  3.007     | 2.0
|  8         | -1.089     | 0.1
|  9         |  4.127     | 2.2
|  10        |  3.519     | 2.1
|  11        |  6.647     | 2.7
|  12        |  1.074     | 1.4
|  13        | -0.8621    | 0.4
|  14        |  3.821     | 2.2
|  15        | -0.9228    | 0.4
|  16        |  6.205     | 2.7
|  17        | -1.1       | 0.0
|  18        |  8.224     | 3.0
|  19        |  7.584     | 2.9
|  20        | -0.8961    | 0.4
|  21        | -1.093     | 0.0
|  22        |  1.293     | 1.5
|  23        |  2.593     | 1.9
|  24        |  5.563     | 2.5
|  25        |  2.046     | 1.7
|  26        |  2.501     | 1.8
|  27        | -0.8853    | 0.4
|  28        | -0.5459    | 0.7444     |
```



for more information, please check out
https://slac-ml.github.io/Badger/docs/getting-started/tutorial

Badger resources

- **Badger homepage**
  https://slac-ml.github.io/Badger/
- **Badger core**
  https://github.com/SLAC-ML/Badger
- **Badger plugins**
  https://github.com/SLAC-ML/Badger-Plugins
- **Badger hands-on**
  https://github.com/SLAC-ML/Badger-Handson
- **Badger on PyPI**
  https://pypi.org/project/badger-opt/
- **Badger on conda**
  https://anaconda.org/conda-forge/badger-opt
- **Badger on Slack**
  #badger
  #badger-handsome