Xopt and Badger: Advanced Optimization Algorithms for Science

Ryan Roussel

rroussel@slac.stanford.edu

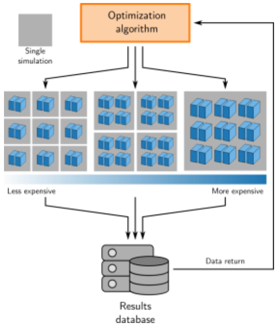
U.S. DEPARTMENT OF ENERGY | Stanford University

SLAC NATIONAL ACCELERATOR LABORATORY

Overview

Xopt algorithm implementation

https://github.com/xopt-org/Xopt




```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}
```

YAML file

Accelerator simulation

Production ready control

Online Control R&D

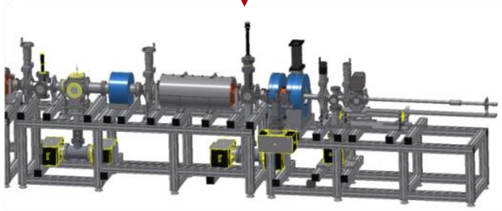
Python interface

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

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

X.data
```

Arbitrary problem

Badger GUI interface

https://github.com/xopt-org/Badger



Experiment facility

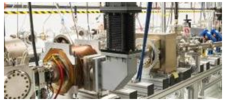
What is Xopt?

- 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

https://github.com/xopt-org/Xopt

Wide Community of Users

Argonne NATIONAL LABORATORY



Accel. Control R&D

SLAC NATIONAL ACCELERATOR LABORATORY

Xopt

ESRF The European Synchrotron

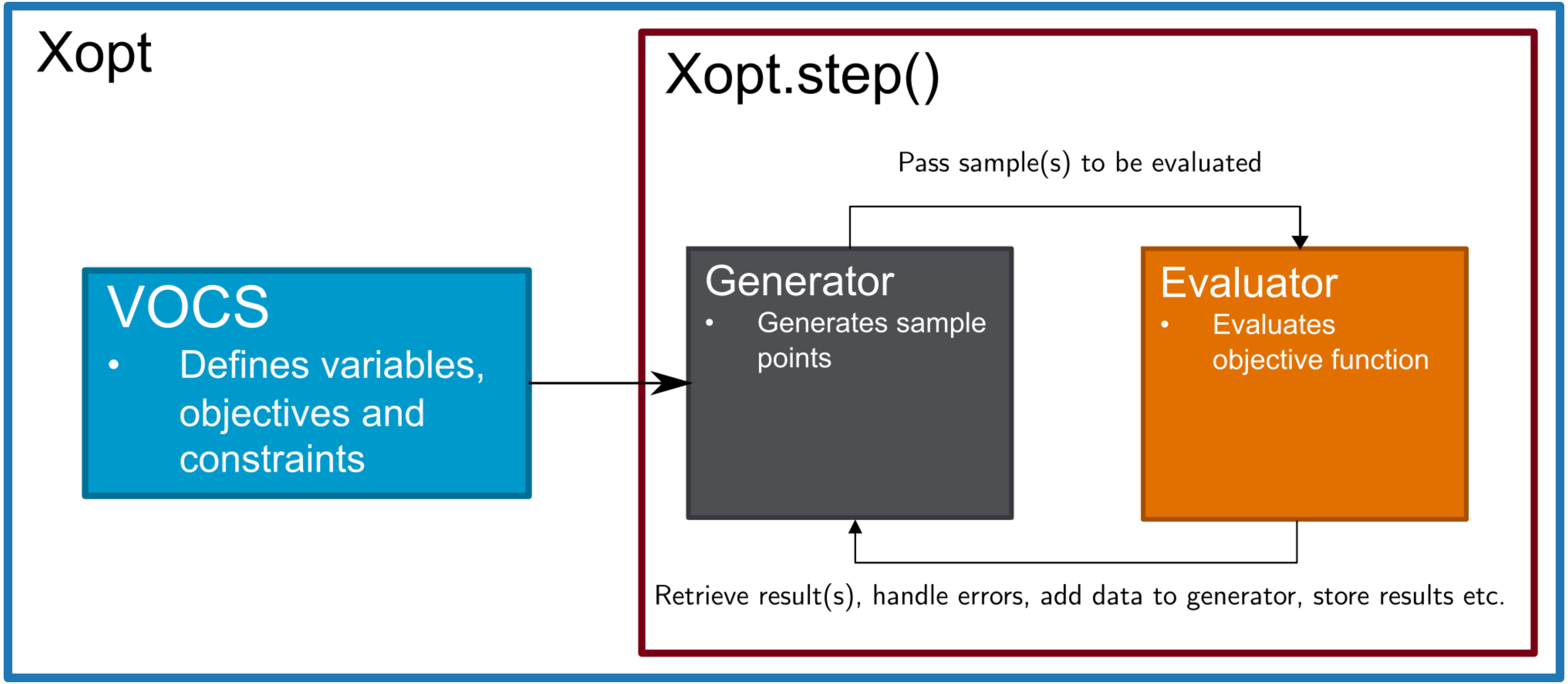
CORNELL UNIVERSITY FOUNDED A.D. 1865

Fermilab

Brookhaven National Laboratory

DESY.

Xopt structure



Note: this process can also be done asynchronously

Xopt Script Overview – Defining the problem

Define the domain/goals

$$x_1, x_2 \in [0, \pi]$$

$$\mathbf{x}^* = \arg\min f(\boldsymbol{x})$$

$$g(\boldsymbol{x}) \leq 0$$

In [2]:

```python
from xopt import VOCS
import math

vocs = VOCS(
    variables = {
        "x1": [0, math.pi],
        "x2": [0, math.pi]
    },
    objectives = {"f": "MINIMIZE"},
    constraints = {"g": ["LESS_THAN", 0]}
)
```

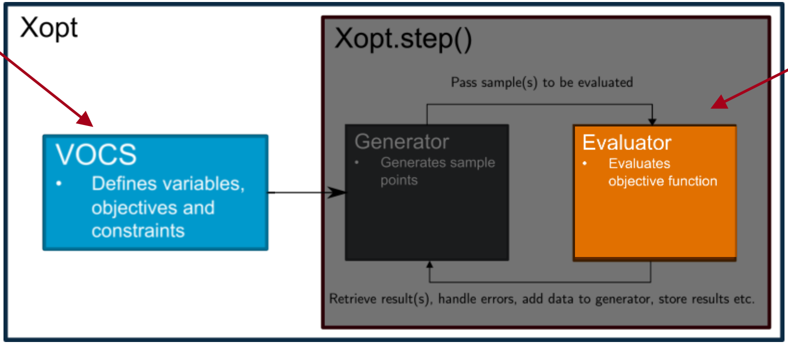
Define the objectives/constraints

$$f(x_1, x_2) = x_1^2 + x_2^2$$

$$g(x_1, x_2) = 1 - x_1^2 - x_2^2$$

In [1]:

```python
from xopt import Evaluator

def evaluate_function(inputs: dict) -> dict:
    objective_value = inputs["x1"]**2 + inputs["x2"]**2
    constraint_value = -inputs["x1"]**2 - inputs["x2"]**2 + 1
    return {"f": objective_value, "g": constraint_value}

evaluator = Evaluator(function=evaluate_function)
```



Xopt Script Overview – Defining the algorithm

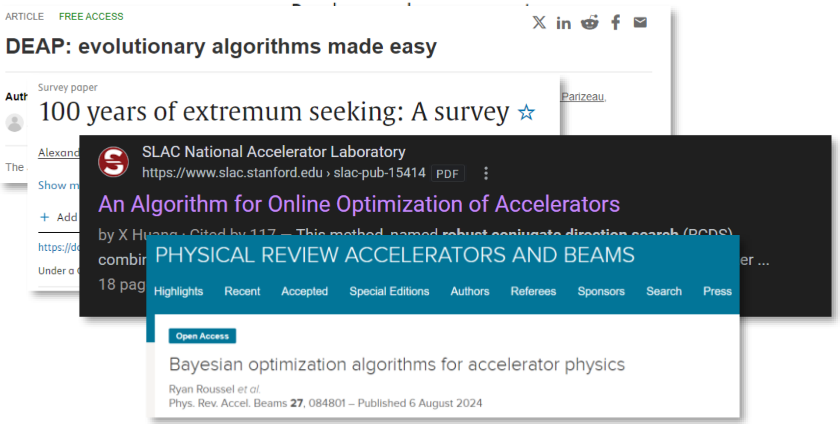
Choose from available generators
(or define your own)

In [3]:
```
from xopt.generators import list_available_generators
list_available_generators()
```

Out[3]:
```
['random',
 'mggpo',
 'neldermead',
 'upper_confidence_bound',
 'mobo',
 'bayesian_exploration',
 'time_dependent_upper_confidence_bound',
 'expected_improvement',
 'multi_fidelity',
 'cnsga',
 'extremum_seeking',
 'rcds']
```

Create the generator
(w/ available options)

In [4]:
```
from xopt.generators import get_generator
# get the docstring for the random generator
print(get_generator("random").__doc__)

# use the get generator method to get the random number generator
generator = get_generator("random")(vocs=vocs)
```



https://xopt.xopt.org/examples/basic/xopt_basic/

https://journals.aps.org/prab/abstract/10.1103/PhysRevAccelBeams.27.084801

Xopt Script Overview – Putting it all together

Create Xopt object

In [5]:
```
from xopt import Xopt
X = Xopt(vocs=vocs, generator=generator, evaluator=evaluator)
```

Evaluate explicit points

In [10]:
```
# evaluate some points additionally
points = {"x1": [1.0, 0.5, 2.25],"x2":[0,1.75,0.6]}
X.evaluate_data(points)
```

Visualize results

```
# view objective values
X.data.plot(y=X.vocs.objective_names)

# view variables values
X.data.plot(*X.vocs.variable_names, kind="scatter")
```

Run optimization

In [12]:
```
# Take one step (generate a single point)
X.step()
```

Example: Online Optimization at SLAC - Setup

SLAC

Create beam size objective function

In [9]:

```python
from epics import caput, caget_many
from time import sleep
import numpy as np
def eval_beamsize(inputs):
        global image_diagnostic
        # set PVs
        for k, v in inputs.items():
            print(f'CAPUT {k} {v}')
            caput(k, v)

        sleep(2.0)

        # get beam sizes from image diagnostic
        metadata = inputs
        results = image_diagnostic.measure_beamsize(5, **metadata)
        results["S_x_mm"] = np.array(results["Sx"]) * 1e-3
        results["S_y_mm"] = np.array(results["Sy"]) * 1e-3


        # add total beam size
        results["total_size"] = np.sqrt(np.array(results["Sx"]) ** 2 + np.array(results["Sy"]) ** 2)
        # results["total_size"] = np.sqrt(np.abs(np.array(results["Sx"])) * np.array(results["Sy"]))
        return results
```

Set beamline parameters

Wait for power supplies/feedback to settle

Measure beam size

Calculate the objective

Initialize defaults

In [11]:

```python
import pandas as pd

default = {'SOLN:IN20:121:BCTRL': 0.474877290758955,
 'QUAD:IN20:121:BCTRL': -0.0048398437,
 'QUAD:IN20:122:BCTRL': 0.0018,
 'QUAD:IN20:361:BCTRL': -3.16,
 'QUAD:IN20:371:BCTRL': 2.5352702,
 'QUAD:IN20:425:BCTRL': -1.1,
 'QUAD:IN20:441:BCTRL': -0.8118599,
 'QUAD:IN20:511:BCTRL': 3.6494056,
 'QUAD:IN20:525:BCTRL': -3.2522187,
}

X.evaluate_data(pd.DataFrame(default, index=[0]))
```

Run optimization

In [23]:

```python
for i in range(10):
    print(i)
    X.step()
```

Example: Online Optimization at SLAC - Results

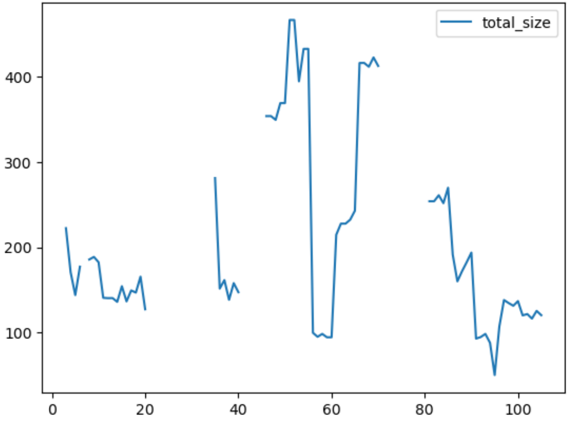
Results data frame (incl. metadata)

SOLN:IN20:121:BCTRL	…	…	0.474877
QUAD:IN20:121:BCTRL	…	…	-0.00484
QUAD:IN20:122:BCTRL	…	…	0.0018
QUAD:IN20:361:BCTRL	…	…	-3.16
QUAD:IN20:371:BCTRL	…	…	2.53527
QUAD:IN20:425:BCTRL	…	…	-1.1
QUAD:IN20:441:BCTRL	…	…	-0.81186
QUAD:IN20:511:BCTRL	…	…	3.649406
QUAD:IN20:525:BCTRL	…	…	-3.252219
Cx	…	…	479.324935
...	…	…	...
Sy	…	…	136.386527
bb_penalty	…	…	-145.364148
total_intensity	…	…	1245909.6
log10_total_intensity	…	…	6.095487
save_filename	…	…	/home/physics3/ml_tuni
S_x_mm	…	…	0.175659
S_y_mm	…	…	0.136387
total_size	…	…	222.390057
xopt_runtime	…	…	6.993356
xopt_error	…	…	False

Visualization

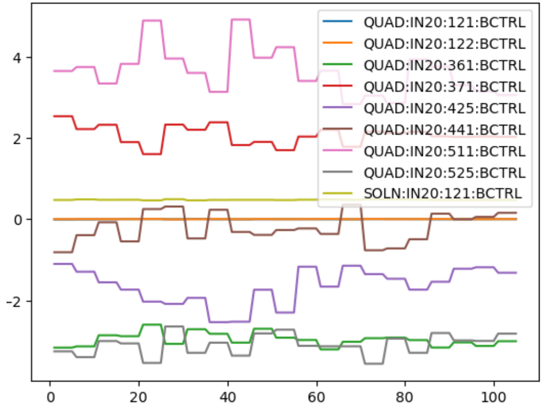
Objectives

```
X.data.plot(y="total_size")
```



Variables

```
X.data.plot(y=X.vocs.variable_names)
```



Pydantic Integration

Xopt objects are robustly validated and serialized/de-serialized via Pydantic

YAML file

```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}
```

Python object(s)

```python
X = Xopt.from_yaml(open("my_file.yml"))
fig, ax = X.generator.visualize_model(
    variable_names = X.vocs.variable_names
)
```

```python
X.dump()
```

YAML file

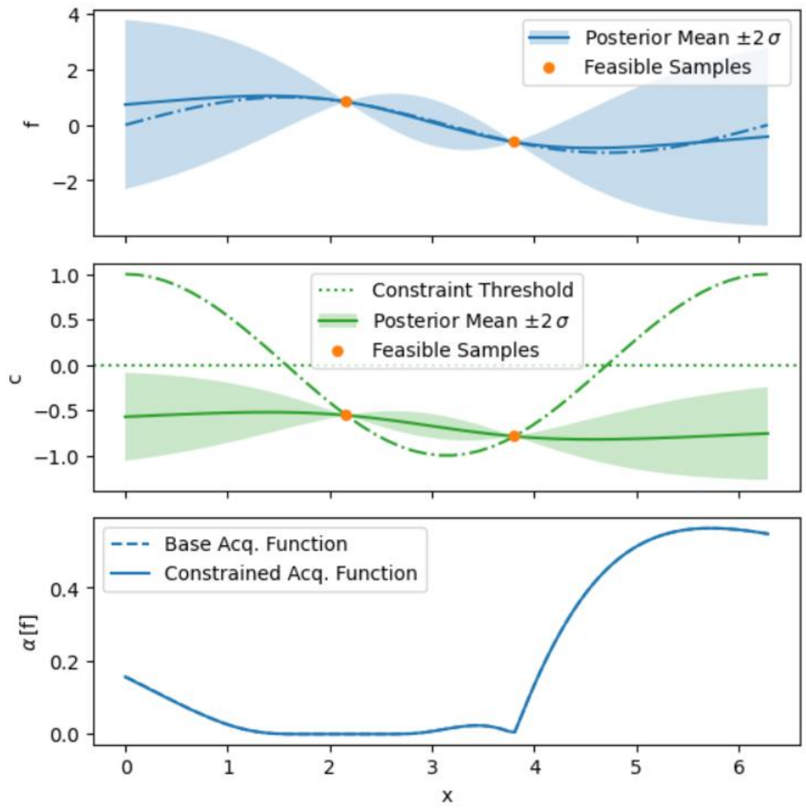
```yaml
xopt:
    max_evaluations: 6400

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, y
    constraints:
        c1: [GREATER_THAN, 0]
        c2: [LESS_THAN, 0.5]
    linked_variables: {x9: x1}
    constants: {a: dummy_constant}
```

```yaml
data:
  Cx:
    '1': 378.5739219281
    '10': 420.7214465998
    '100': 438.3514501154
    '101': 466.4557444371
```

Example: BO w/ introspection

```
from xopt.evaluator import Evaluator
from xopt.generators.bayesian import UpperConfidenceBoundGenerator
from xopt import Xopt

evaluator = Evaluator(function=sin_function)
generator = UpperConfidenceBoundGenerator(vocs=vocs)
X = Xopt(evaluator=evaluator, generator=generator, vocs=vocs)
```

```
for i in range(n_steps):

    model = X.generator.train_model()
    fig, ax = X.generator.visualize_model(n_grid=100)


    # add ground truth functions to plots
    out = test_function({"x": test_x})
    ax[0].plot(test_x, out["f"], "C0-.")
    ax[1].plot(test_x, out["c"], "C2-.")

    # do the optimization step
    X.step()
```



https://xopt.xopt.org/examples/single_objective_bayes_opt/bo_tutorial/

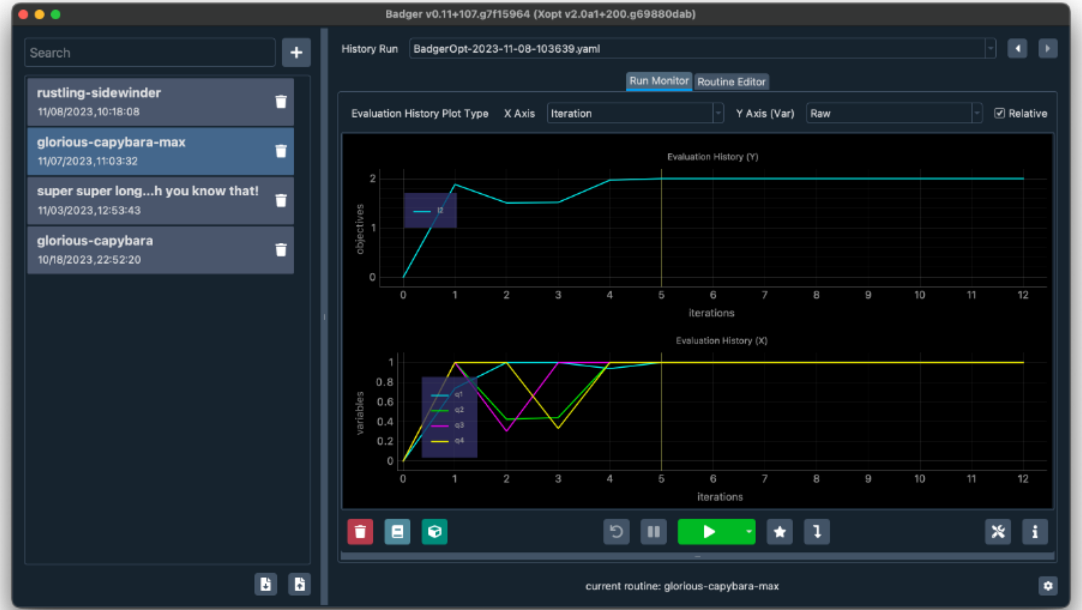
Demonstration

https://colab.research.google.com/drive/1EQfygnLQW_R-9YtE2hnTzf6KnHOUQ9_f?usp=sharing

Badger

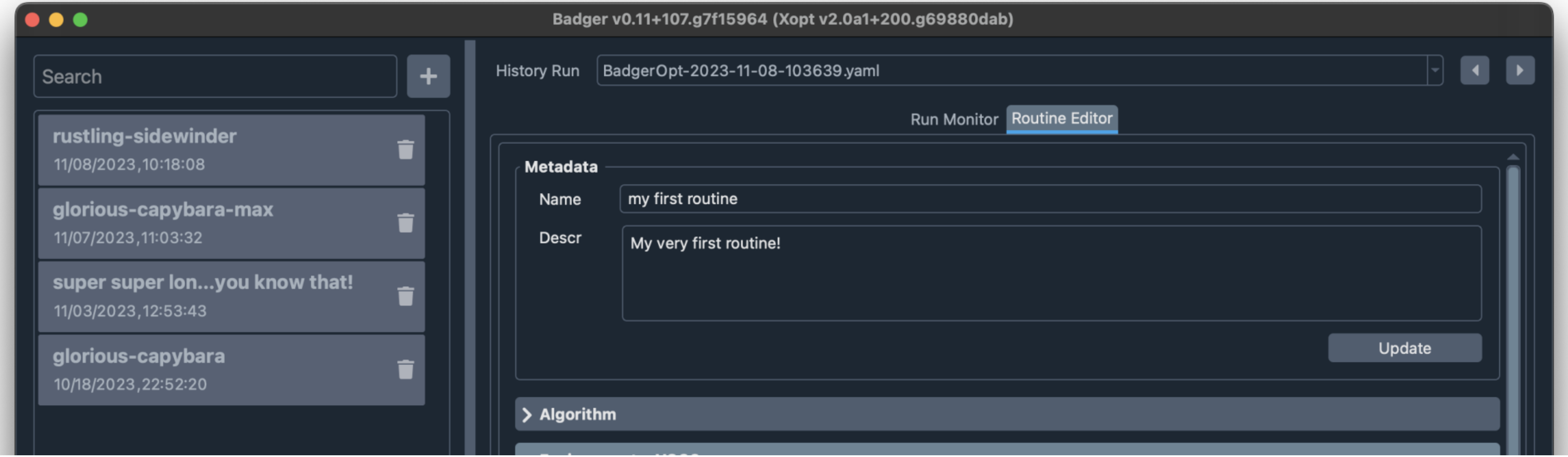
```
badger -g
```

You should be able to see the main GUI like below:



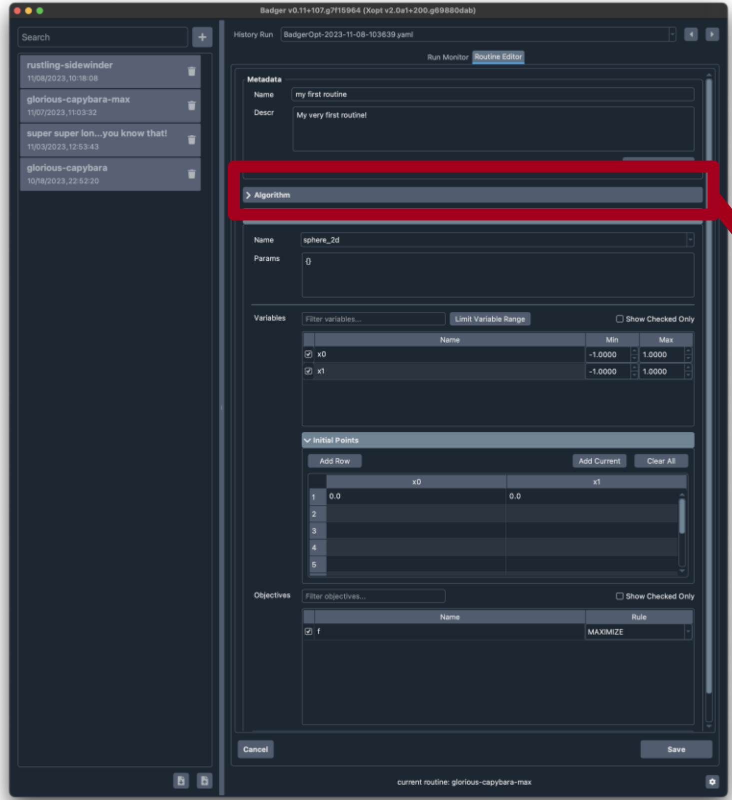
https://github.com/xopt-org/Badger

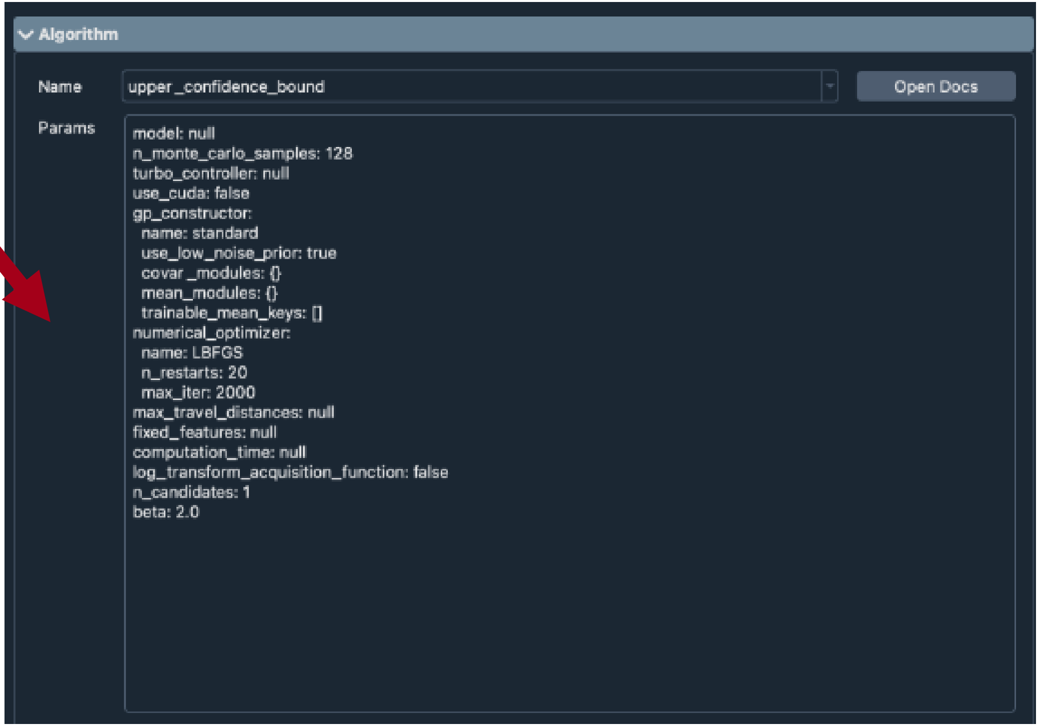
Badger Routine Editor



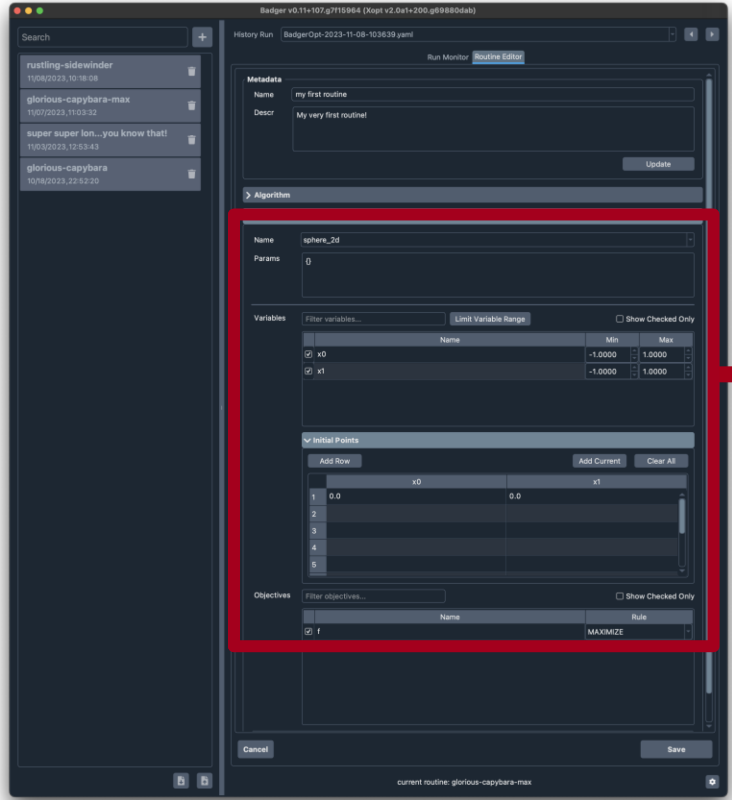
Creates a Badger routine object (subclass of Xopt object!)

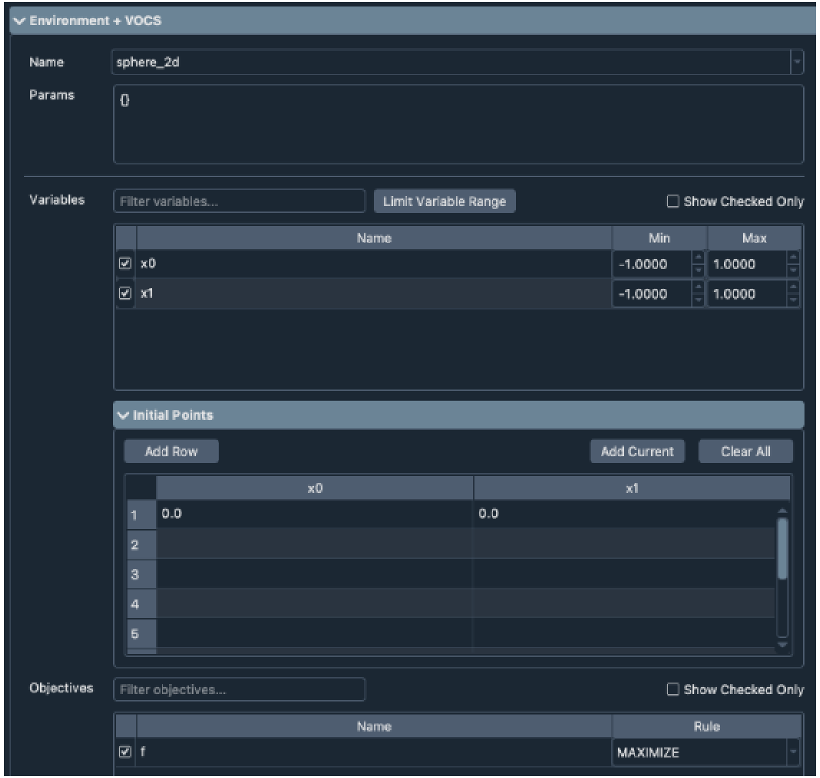
Badger Routine Editor – Algorithm (Generator)





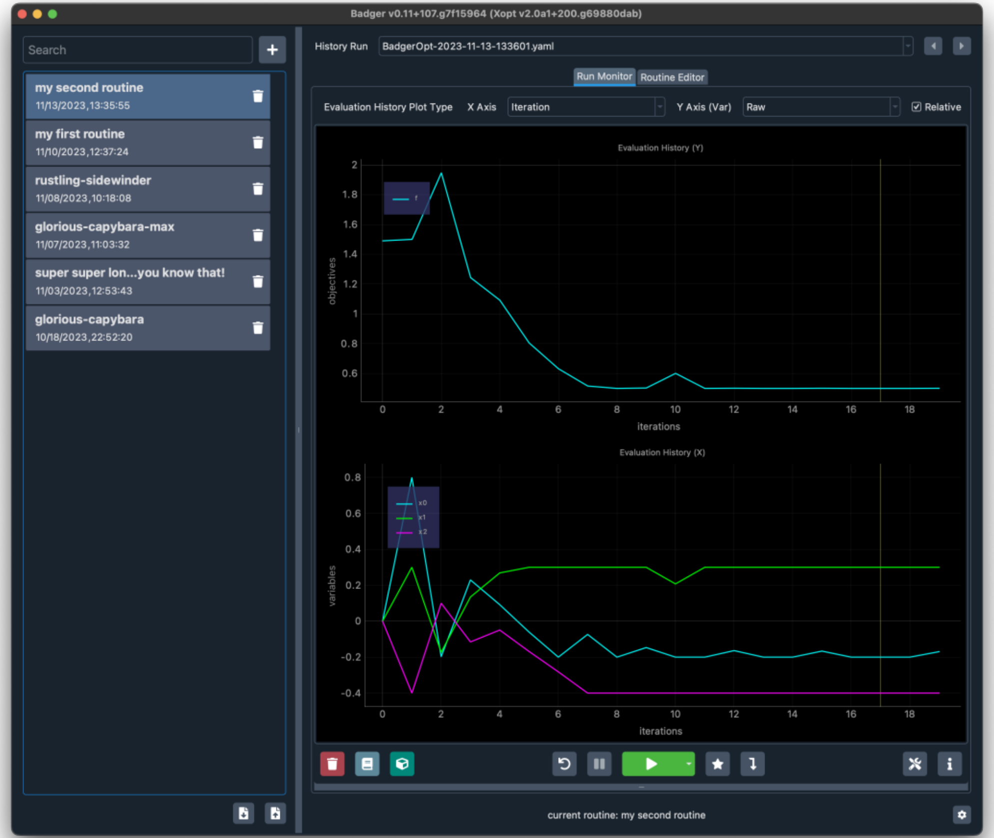
Badger Routine Editor - Environment





Run Monitor

Available routines



Objective(s)

Variable(s)

Play/Pause/Reset/Etc.

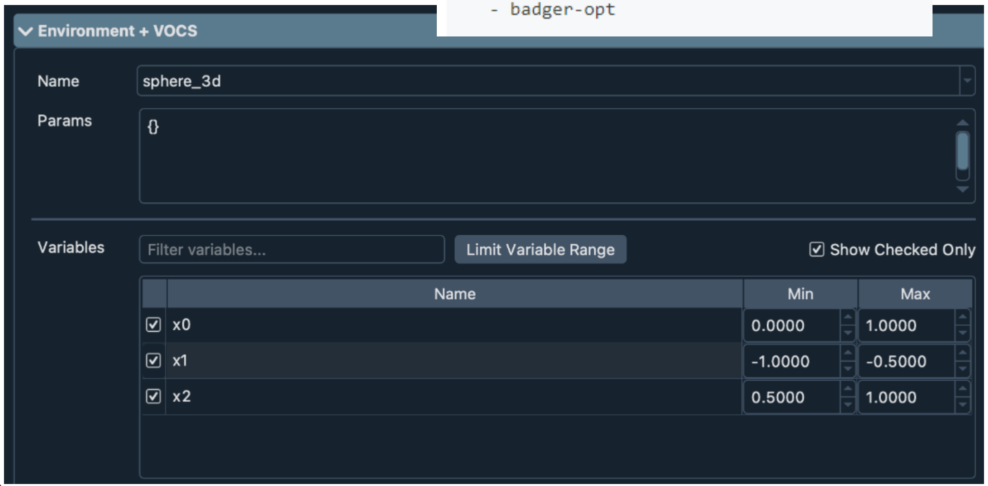
Defining an optimization environment

```python
from badger import environment


class Environment(environment.Environment):

    name = 'sphere_3d'  # name of the environment
    variables = {  # variables and their hard-limited ranges
        'x0': [-1, 1],
        'x1': [-1, 1],
        'x2': [-1, 1],
    }
    observables = ['f']  # measurements

    # Internal variables to store the current values of
    # the variables and observables
    _variables = {
        'x0': 0.0,
        'x1': 0.0,
        'x2': 0.0,
    }
    _observations = {
        'f': None,
    }
```

Add environment metadata file for badger to find

configs.yaml

```yaml
---
name: sphere_3d
description: "3D sphere test environment"
version: "0.1"
dependencies:
  - torch
  - badger-opt
```



https://slaclab.github.io/Badger/docs/getting-started/tutorial_0

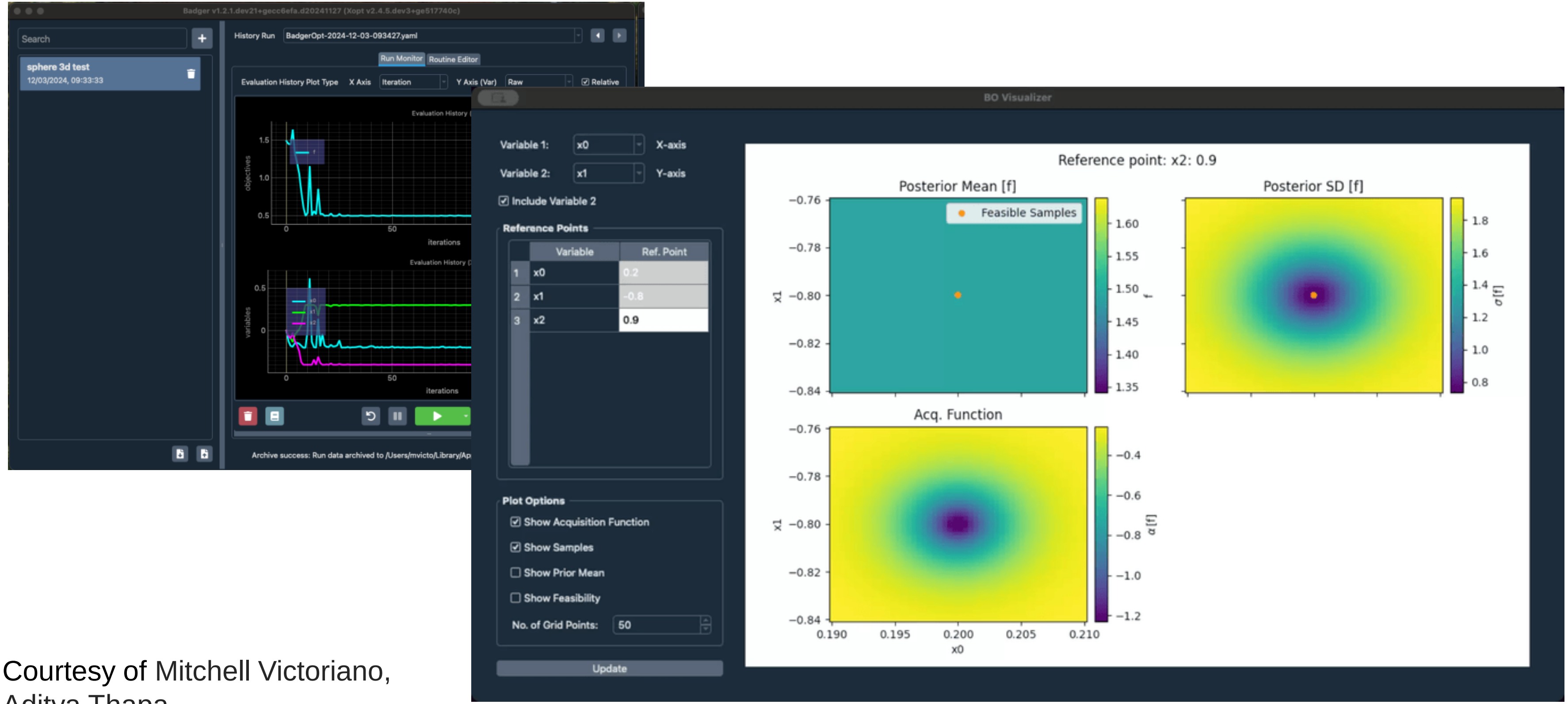
Defining an optimization environment

```python
from badger import environment


class Environment(environment.Environment):

    name = 'sphere_3d'  # name of the environment
    variables = {  # variables and their hard-limited ranges
        'x0': [-1, 1],
        'x1': [-1, 1],
        'x2': [-1, 1],
    }
    observables = ['f']  # measurements

    # Internal variables to store the current values of
    # the variables and observables
    _variables = {
        'x0': 0.0,
        'x1': 0.0,
        'x2': 0.0,
    }
    _observations = {
        'f': None,
    }
```

```python
    # Variable getter -- tells Badger how to get current values of the variables
    def get_variables(self, variable_names):
        variable_outputs = {v: self._variables[v] for v in variable_names}

        return variable_outputs

    # Variable setter -- how to set variables to the given values
    def set_variables(self, variable_inputs: dict[str, float]):
        for var, x in variable_inputs.items():
            self._variables[var] = x

        # Filling up the observations
        f = self._variables['x0'] ** 2 + self._variables['x1'] ** 2 + \
            self._variables['x2'] ** 2

        self._observations['f'] = [f]

    # Observable getter -- how to get current values of the observables
    def get_observables(self, observable_names):
        return {k: self._observations[k] for k in observable_names}
```

https://slaclab.github.io/Badger/docs/getting-started/tutorial_0

Coming soon™ to Badger - Interactive Visualization



Courtesy of Mitchell Victoriano, Aditya Thapa

Coming soon™ to Badger – Template Files

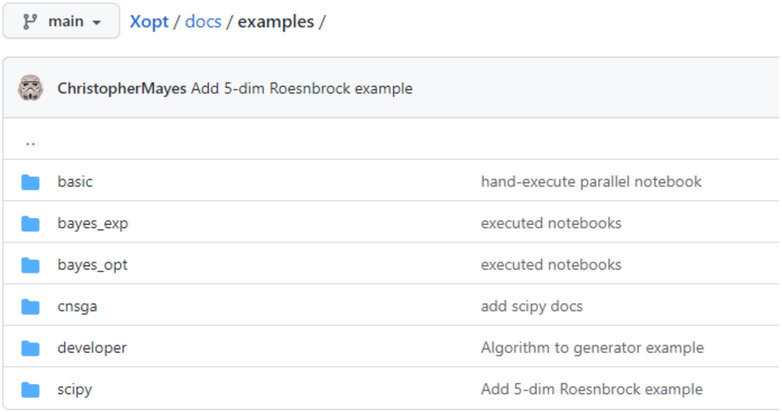
One click setup of common accelerator tasks to be loaded into Badger

- Starting point for running optimization
- Enables best practice set-up of tasks with custom settings
- Allows customization after loading

```
environment:
  name: lcls_ii
generator:
  name: expected_improvement
objectives:
sxr_pulse_intensity_p80: MAXIMIZE
observables:
    - beam_loss
variables:
    QUAD:HTR:120:BCTRL:
    - -2.8216707322764067
    - -2.5529401863453 2
    QUAD:HTR:140:BCTRL:
    - 2.1881688427214896
    - 2.4185024051132253
    QUAD:HTR:300:BCTRL:
    - 0.8507717426280459
    - 0.9403266629046825
    QUAD:HTR:320:BCTRL:
    - -1.9970261258813424
    - -1.8068331615116906
```

Conclusion

- **Look at the examples in docs/examples** !!!!
- Ask for invite to #xopt and #badger Slack channels
- Reach out to us at SLAC for help!



Additional Details

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!
- Functions can be defined at the module level and passed via string if they are in PYTHONPATH, they can also be passed inside the same python file (use __main__.my_function)
- 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:
    nam
    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
```

Evaluate function

- 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!

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

```python
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

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()))
```

```yaml
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}
```