Simple Sine with Multiprocessing

Quick Tutorial

`https://libensemble.readthedocs.io/en/main/tutorials/local_sine_tutorial.html`

generator()

simulator()

my_ensemble.py

```
def generator(H_in, persis_info, gen_specs, _):
```

- **H_in**: A History array selection
- **persis_info**: Per-worker persistent info
- **gen_specs**: Generator fields and parameters

```python
import numpy as np

def gen_random_sample(H_in, persis_info, gen_specs, _):

    # Pull out user parameters
    user_specs = gen_specs['user']

    # Get lower and upper bounds
    lower = user_specs['lower']
    upper = user_specs['upper']

    # Determine how many values to generate
    num = len(lower)
    batch_size = user_specs['gen_batch_size']

    # Create empty array of 'batch_size' zeros. Array dtype should match 'out' fields
    out = np.zeros(batch_size, dtype=gen_specs['out'])

    # Set the 'x' output field to contain random numbers, using random stream
    out['x'] = persis_info['rand_stream'].uniform(lower, upper, (batch_size, num))

    # Send back our output and persis_info
    return out, persis_info
```

Super-quick Exercise:

Write a generator that instead produces random integers using:

```
numpy.random.RandomState.randint(low, high, size)
```

(Modify the provided generator function)

```
def simulator(H_in, persis_info, sim_specs, _):
```

- **H_in**: A History array selection

- **persis_info**: Per-worker persistent info

- **sim_specs**: Simulator fields and parameters

```python
import numpy as np

def sim_find_sine(H_in, persis_info, sim_specs, _):

    # Create an output array of a single zero
    out = np.zeros(1, dtype=sim_specs['out'])

    # Set the zero to the sine of the input value stored in H
    out['y'] = np.sin(H_in['x'])

    # Send back our output and persis_info
    return out, persis_info
```

Super-quick Exercise:

Write a simulator that instead calculates the cosine using:

```
numpy.cos(x)
```

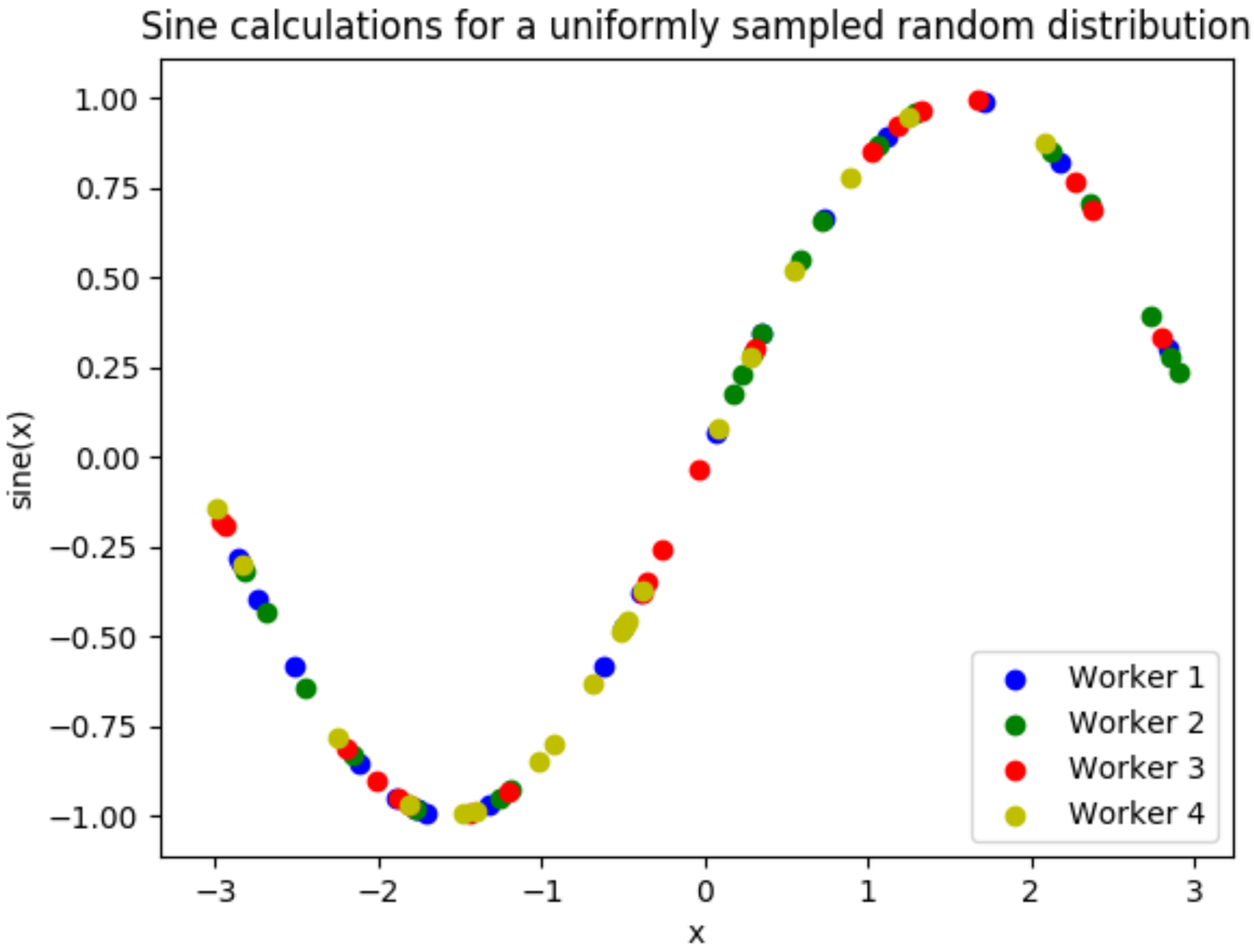
(Modify the provided simulator function)

```
import numpy as np
from libensemble.libE import libE
from generator import gen_random_sample
from simulator import sim_find_sine
from libensemble.tools import add_unique_random_streams

nworkers = 4
libE_specs = {'nworkers': nworkers, 'comms': ‘local'} # multiprocessing comms

. . .
```

```
. . .

gen_specs = {'gen_f': gen_random_sample,    # Our generator function
             'out': [('x', float, (1,))],   # gen_f output (name, type, size)
             'user': {
                'lower': np.array([-3]),    # lower boundary for random sampling
                'upper': np.array([3]),     # upper boundary for random sampling
                'gen_batch_size': 5         # number of x's gen_f generates per call
                }
             }

sim_specs = {'sim_f': sim_find_sine,        # Our simulator function
             'in': ['x'],                   # Input field names. 'x' from gen_f output
             'out': [('y', float)]}         # sim_f output. 'y' = sine('x')


. . .
```

```
. . .

# Create per-worker dictionaries each containing a random stream
persis_info = add_unique_random_streams({}, nworkers+1)

# Stop libEnsemble after 80 simulations
exit_criteria = {'sim_max': 80}

# Launch libEnsemble. H will contain all computed points
H, persis_info, flag = libE(sim_specs, gen_specs, exit_criteria, persis_info,
                            libE_specs=libE_specs)


. . .
```

```
$ python my_ensemble.py
```


Sine calculations for a uniformly sampled random distribution
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
sine(x)
−3
−2
−1
0
1
2
3
x
Worker 1
Worker 2
Worker 3
Worker 4

```
. . .

from mpi4py import MPI

libE_specs = {'comms': 'mpi'} # MPI comms
nworkers = MPI.COMM_WORLD.Get_size() - 1
is_manager = (MPI.COMM_WORLD.Get_rank() == 0) # Manager process has MPI rank 0

. . .

# Launch libEnsemble. H will contain all computed points
H, persis_info, flag = libE(sim_specs, gen_specs, exit_criteria, persis_info,
                            libE_specs=libE_specs)

. . .

if is_manager:  # So only one process evaluates output
   process_History(H)
```

```
$ mpirun -n 5 python my_ensemble.py
```

Use same code with multiprocessing or MPI:

```
from libensemble.tools import parse_args()

. . .

nworkers, is_manager, libE_specs, _ = parse_args()

. . .

# Launch libEnsemble. H will contain all computed points
H, persis_info, flag = libE(sim_specs, gen_specs, exit_criteria, persis_info,
                            libE_specs=libE_specs)
```

```
$ python my_ensemble.py —-comms local —-nworkers 4
```

or

```
$ mpirun -n 5 python my_ensemble.py
```

Questions?

https://libensemble.readthedocs.io/en/main/index.html

https://github.com/Libensemble/libensemble