Tsuchinoko Documentation

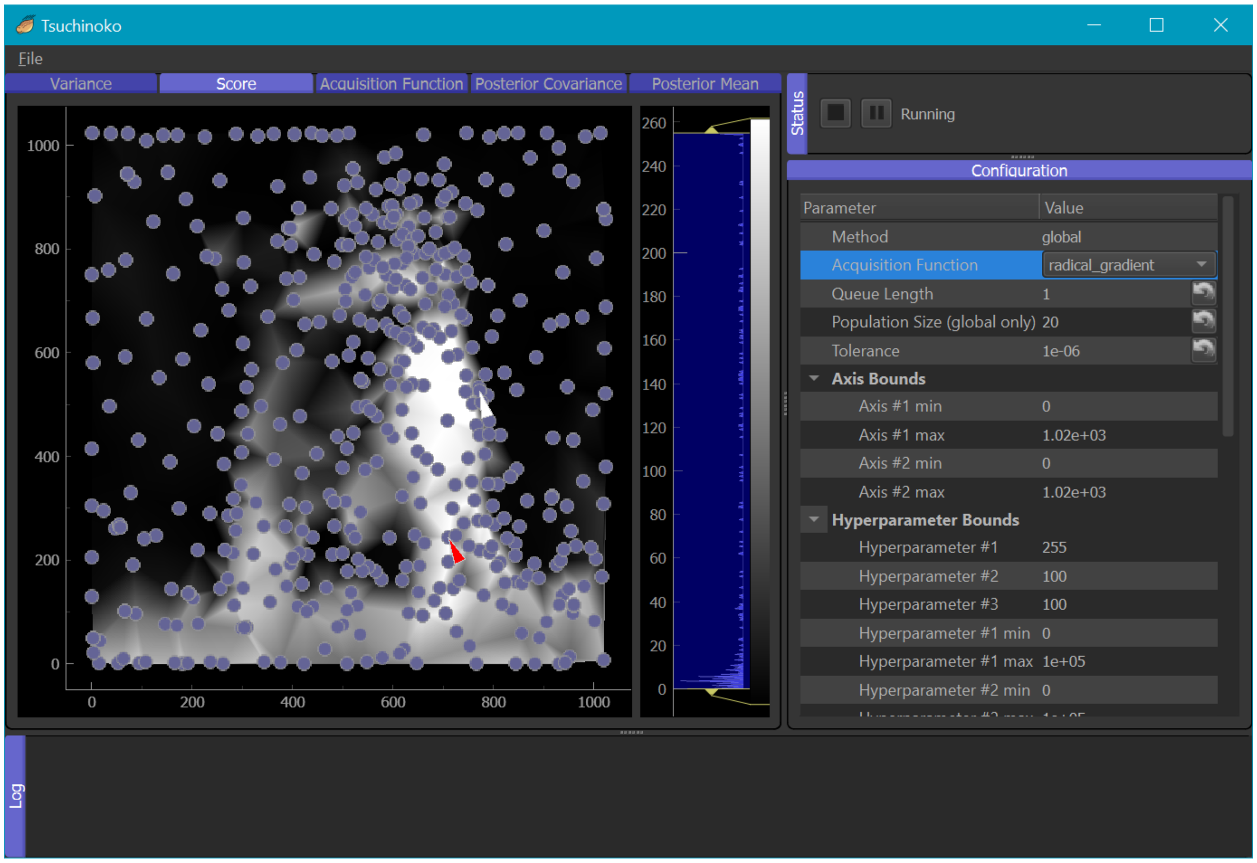
Release 1.1.16.dev5+g3a82630

Contributors

Mar 12, 2024

CONTENTS

Tsuchinoko is a Qt application for adaptive experiment execution and tuning. Live visualizations show details of measurements, and provide feedback on the adaptive engine's decision-making process. The parameters of the adaptive engine can also be tuned live to explore and optimize the search procedure.



Parameter	Value
Method	global
Acquisition Function	radical_gradient
Queue Length	1
Population Size (global only)	20
Tolerance	1e-06
Axis Bounds	
Axis #1 min	0
Axis #1 max	1.02e+03
Axis #2 min	0
Axis #2 max	1.02e+03
Hyperparameter Bounds	
Hyperparameter #1	255
Hyperparameter #2	100
Hyperparameter #3	100
Hyperparameter #1 min	0
Hyperparameter #1 max	1e+05
Hyperparameter #2 min	0

While Tsuchinoko is designed to allow custom adaptive engines to drive experiments, the gpCAM engine is a featured inclusion. This tool is based on a flexible and powerful Gaussian process regression at the core.

A Tsuchinoko system includes 4 distinct components: the GUI client, an adaptive engine, and execution engine, and a core service. These components are separable to allow flexibility with a variety of distributed designs.

CHAPTER ONE

INSTALLATION

The latest stable Tsuchinoko version is available on PyPI, and is installable with `pip`. It is recommended that you use Python 3.10 for this installation.

```
pip install tsuchinoko
```

For more information, see the *installation documentation*.

CHAPTER TWO

EASY INSTALLATION

For Mac OSX and Windows, pre-packaged installers are available. These do not require a base Python installation. See the installation documentation for more details.

- Latest Windows Installer
- Latest Mac OSX Installer

CHAPTER

THREE

GETTING STARTED WITH YOUR OWN ADAPTIVE EXPERIMENTS

You can find demos in the Tsuchinoko Github repository's examples folder. It is suggested to first try running both `server_demo.py` and `client_demo.py` simultaneously. This demo performs a simulated adaptive experiment by making "measurements" sampled from an image file. It is recommended as a first run to follow the *Getting Started* guide.

CHAPTER

FOUR

ABOUT THE NAME

Japanese folklore describes the Tsuchinoko as a wide and short snake-like creature living in the mountains of western Japan. This creature has a cultural following similar to the Bigfoot of North America. Much like the global optimum of a non-convex function, its elusive nature is infamous.

4.1 Windows/Mac Installers

Pre-packaged installers are available for Windows/Mac. These provide a quick and simple way to get started, as they do require a base Python installation to be pre-installed.

- Latest Windows Installer
- Latest Mac OSX Installer

4.1.1 Limitations

Although these installers provide all necessary components to run the included demos and many custom adaptive experiments, these installers are limited in what Python packages are included. If your experiment requires a Python package that is not included, you may want to use a *standard installation*.

4.2 Getting Started

This is a quick-start guide that will help you install Tsuchinoko and explore a simulated adaptive experiment.

For more in-depth documentation for developing custom adaptive experiments see:

- *AdaptiveEngine documentation*
- *ExecutionEngine documentation*
- *Core service documentation*

4.2.1 Install Tsuchinoko

To begin, you will need to install Python 3.10. It is generally recommended that you next create a virtual environment to contain the installation.

```
$ python -m venv tsuchinoko-venv
```

You will then need to activate the virtual environment. This varies by operating system.

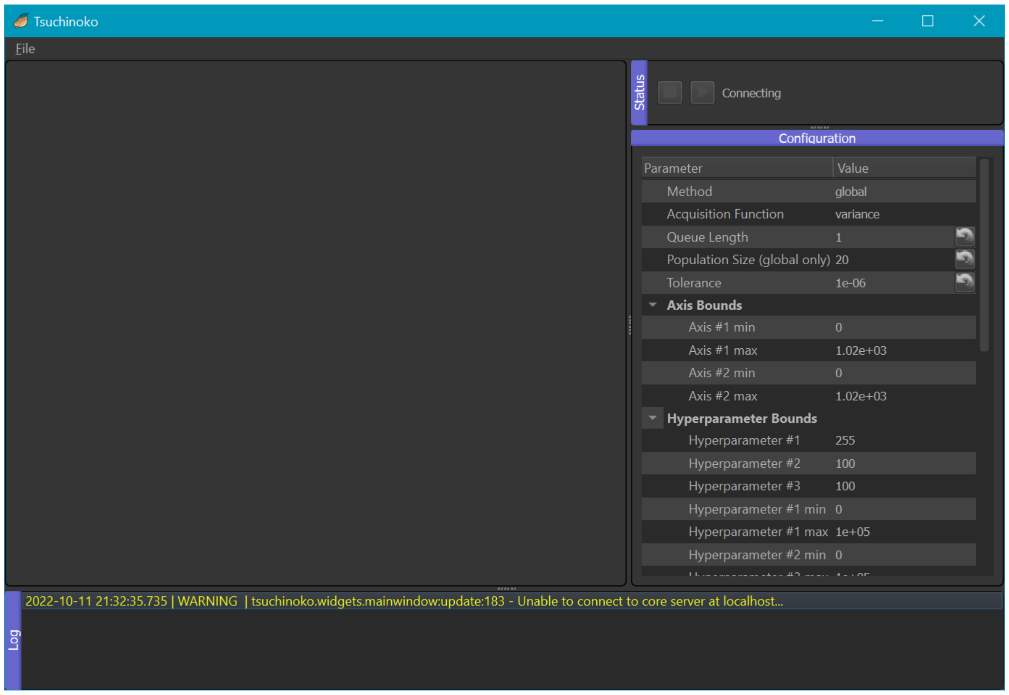
Platform	Shell	Command to activate virtual environment
POSIX	bash/zsh	$ source tsuchinoko-venv/bin/activate
	fish	$ source tsuchinoko-venv/bin/activate.fish
	csh/tcsh	$ source tsuchinoko-venv/bin/activate.csh
	PowerShell Core	$ tsuchinoko-venv/bin/Activate.ps1
Windows	cmd.exe	C:> tsuchinoko-venv\Scripts\activate.bat
	PowerShell	PS C:> tsuchinoko-venv\Scripts\Activate.ps1

With the virtual environment active, you will then install Tsuchinoko:

```
$ pip install tsuchinoko
```

Tsuchinoko should now be installed! You can run the Tsuchinoko client to quickly test the installation. Note that a running Tsuchinoko server will be needed to run any experiment.

```
$ tsuchinoko
```



Parameter	Value
Method	global
Acquisition Function	variance
Queue Length	1
Population Size (global only)	20
Tolerance	1e-06
Axis Bounds	
Axis #1 min	0
Axis #1 max	1.02e+03
Axis #2 min	0
Axis #2 max	1.02e+03
Hyperparameter Bounds	
Hyperparameter #1	255
Hyperparameter #2	100
Hyperparameter #3	100
Hyperparameter #1 min	0
Hyperparameter #1 max	1e+05
Hyperparameter #2 min	0

```
2022-10-11 21:32:35.735 | WARNING  | tsuchinoko.widgets.mainwindow:update:183 - Unable to connect to core server at localhost...
```

According to your preferences, components of Tsuchinoko can also be distributed across multiple systems to accommodate a distributed design which leverages different hardware resources. If you plan to do this, you'll need a Tsuchinoko installation on each system.

Did something go wrong? See *Installation Troubleshooting*.

4.2.2 Running Tsuchinoko with a Simulated Experiment

Let's try out a simulated adaptive experiment now! In this example, Tsuchinoko will adaptively sample data from a source image to create a reconstruction by simulated measurements. You'll need the server example script and an image to be sampled from. Download both of these files. We'll discuss the contents of the example server script later.

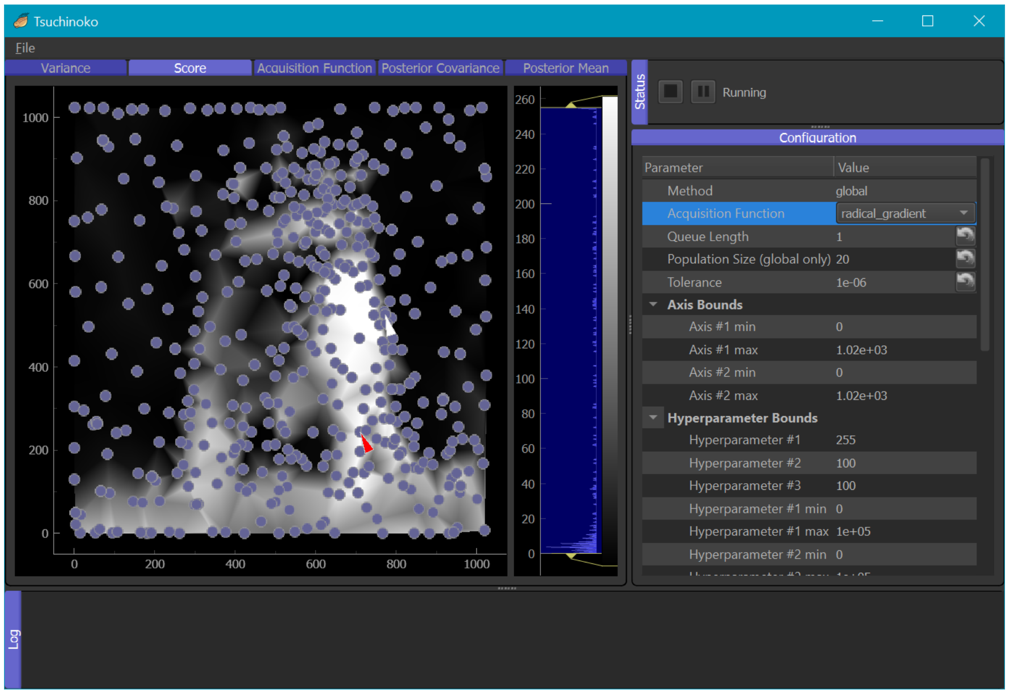
With the virtual environment active and both these files in the current directory, start the Tsuchinoko core server:

```
$ python server_demo.py
```

The core server will wait for a client to connect. Now, start a Tsuchinoko client in another shell:

```
$ tsuchinoko
```

The client will automatically connect to the server. From the parameter table on the right, select an Acquisition Function to test (try starting with `gradient`). To start the experiment, click the 'run' () button.

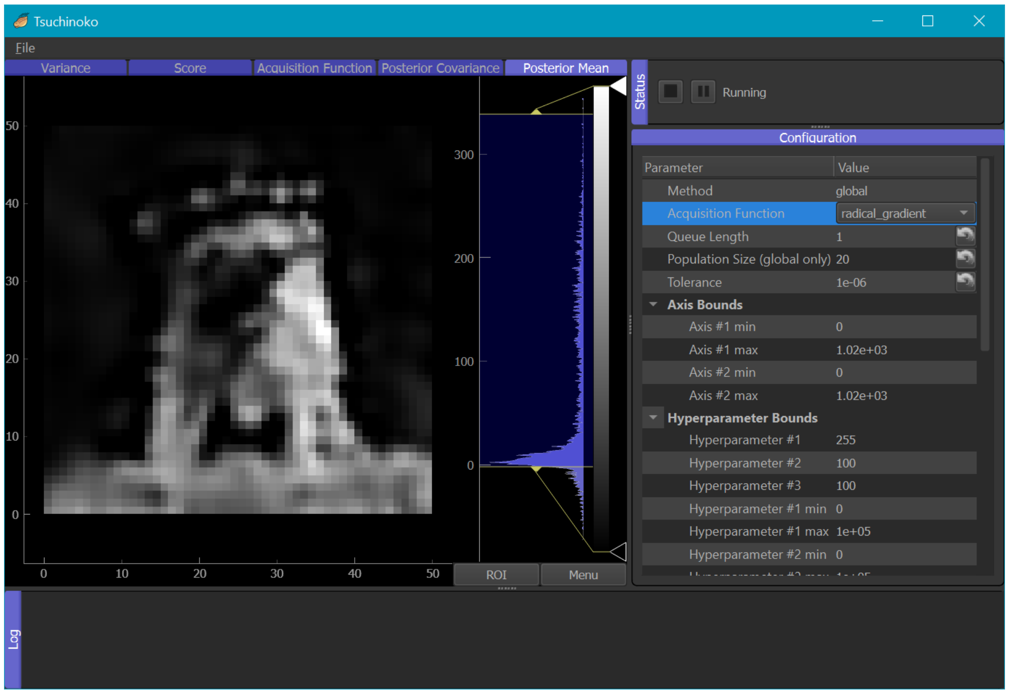


Running

Parameter	Value
Method	global
Acquisition Function	radical_gradient
Queue Length	1
Population Size (global only)	20
Tolerance	1e-06
Axis Bounds	
Axis #1 min	0
Axis #1 max	1.02e+03
Axis #2 min	0
Axis #2 max	1.02e+03
Hyperparameter Bounds	
Hyperparameter #1	255
Hyperparameter #2	100
Hyperparameter #3	100
Hyperparameter #1 min	0
Hyperparameter #1 max	1e+05
Hyperparameter #2 min	0

Now have fun!

- Experiment with different acquisition functions! Try switching between them while Tsuchinoko is running.
- Want to nudge Tsuchinoko in the right direction? Right click in the `Score` or `Posterior Mean` displays and select `Queue Measurement at Point`
- Save your work! You can save the current state of an experiment from File > Save..., or save an image of a display by right-clicking a display and selecting `Export...`.



Parameter	Value
Method	global
Acquisition Function	radical_gradient
Queue Length	1
Population Size (global only)	20
Tolerance	1e-06
Axis Bounds	
Axis #1 min	0
Axis #1 max	1.02e+03
Axis #2 min	0
Axis #2 max	1.02e+03
Hyperparameter Bounds	
Hyperparameter #1	255
Hyperparameter #2	100
Hyperparameter #3	100
Hyperparameter #1 min	0
Hyperparameter #1 max	1e+05
Hyperparameter #2 min	0

4.2.3 Next Steps

Now that you've seen Tsuchinoko in action, let's take a look at the server script that *describes the experiment*.

4.2.4 Installation Troubleshooting

Some environments may need extra steps to install Tsuchinoko. Solutions are provided for these unusual cases below.

libGL.so.1: cannot open shared object file: No such file or directory

You may be missing libgl on your system, for example if you are installing on a headless server. Look for a package that provides libgl in your package manager (i.e. for Ubuntu, install `libgl1-mesa-glx`)

4.3 Describing an experiment

A Tsuchinoko experiment is described in 2 parts:

- An **adaptive** engine which chooses positions to measure
- An **execution** engine which measures positions

These two elements are required for Tsuchinoko's Core service, which communicates with the client.

Let's look at an example experiment.

4.3.1 Example

For this section, we'll look at the server_demo.py script discussed in *Getting Started*. In this example, Tsuchinoko will adaptively sample data from the source image to create a reconstruction by simulated measurements.

Skipping the imports, first we'll load the source image data to be sampled from. In this case, our source data is an RGB JPEG image, so we'll average over the color dimension to form a luminosity map.

```
# Load data from a jpg image to be used as a luminosity map
image = np.flipud(np.asarray(Image.open(Path(__file__).parent/'sombrero_pug.jpg')))
luminosity = np.average(image, axis=2)
```

Next, we can build a function that does the 'sampling'. Since our simulation source here is discrete, it helps to interpolate between pixels so as to avoid sharpness across pixel edges. Each measurement must include:

- The measured position (in some cases it may not be exactly the target position)
- The measured value
- The variance associated with that measurement
- A `dict` with any additional measurement data not covered above

```
# Bilinear sampling will be used to effectively smooth pixel edges in source data
def bilinear_sample(pos):
    measured_value = ndimage.map_coordinates(luminosity, [[pos[1]], [pos[0]]],␣
↪order=1)[0]
    variance = 1
    return pos, measured_value, variance, {}
```

Note that the variance returned here is simply 1. In this example with simulated measurements, there's no empirical measure of variance.

A simple **execution** engine can be constructed from the above function.

```
execution = SimpleEngine(measure_func=bilinear_sample)
```

Let's also construct an **adaptive** engine. The featured adaptive engine in Tsuchinoko is gpCAM.

```
# Define a gpCAM adaptive engine with initial parameters
adaptive = GPCAMInProcessEngine(dimensionality=2,
                                parameter_bounds=[(0, image.shape[1]),
                                                  (0, image.shape[0])],
                                hyperparameters=[255, 100, 100],
                                hyperparameter_bounds=[(0, 1e5),
                                                       (0, 1e5),
                                                       (0, 1e5)])
```

Here the `GPCAMInProcessEngine` is constructed with a set dimensionality, the domain bounds (in this case the image shape), and some hyperparameter initial values and bounds (for more info on these see gpCAM's docs).

Next, let's construct the core. The ZMQCore is standard.

```
# Construct a core server
core = ZMQCore()
core.set_adaptive_engine(adaptive)
core.set_execution_engine(execution)
```

The `core.main()` in the last section starts the core's main loop, and is required for the core server to run.

```
if __name__ == '__main__':
    # Start the core server
    core.main()
```

Only when python runs this script with `python server_demo.py` will the above part be executed.

4.3.2 Expanding on this example

In the above example, `measure_func` (or `bilinear_sample`) is the critical piece from which you might start expanding. By providing your own `measure_func`, you could make 'measurements' any way you'd like.

For more advanced usages, you may even subclass the `Engine` classes to customize their functionality, or customize the `Core` to modify the experimental process.

Users of Bluesky should note that a `BlueskyInProcessEngine` may provide convenience.

4.4 Bluesky Integration

See `tsuchinoko.execution.bluesky_in_process.BlueskyInProcessEngine` and `examples/server_demo_bluesky.py` for more info.

(WIP)

4.5 API

4.5.1 Adaptive Engine

class `tsuchinoko.adaptive.`**`Engine`**

The Adaptive Engine base class. This component is generally to be responsible for determining future measurement targets.

abstract request_targets(*position: Tuple*) → Iterable[Tuple]

Determine new targets to be measured

Parameters

position: tuple
The current 'position' of the experiment in the target domain.

Returns

targets: array_like
The new targets to be measured.

abstract reset()

Called when an experiment stops, or is about to start. Returns the engine to a clean state.

abstract train()

Perform training. This can be short-circuited to only train on every N-th iteration, for example.

abstract update_measurements(*data: Data*)

Update internal variables with the provided new data

abstract update_metrics(*data: Data*)

Calculates various metrics to drive visualizations for the client. The data object is expected to be mutated to include these new values.

4.5.2 Execution Engine

class tsuchinoko.execution.Engine

The Execution Engine base class. This component is generally to be responsible for measuring targets.

abstract get_measurements() → List[Tuple]

Returns new measurements made since its last call. Generally measured values are accumulated by a background thread, then returned and cleared here.

Returns

position: tuple
The measured target position.

value: float
The value at that position.

variance: float
The variance associated with the measurement of that value.

metrics: dict
Any non-standard metric values to be used for visualization at the client.

abstract get_position() → Tuple

Returns the current position in the target domain.

Returns

current_position: tuple
The current position in the target domain

abstract update_targets(*targets: List[Tuple]*)

Updates the list of measurement targets. Generally new targets will take precedence, and will be stashed for measurement in the background.

Parameters

targets: list
A list of new measure target positions.

4.5.3 Core Service

class tsuchinoko.core.Core(*execution_engine:* Engine | *None* = *None*, *adaptive_engine:* Engine | *None* = *None*)

4.6 Bluesky-Adaptive Integration

Bluesky-Adaptive is Bluesky companion package for tightly integrated adaptive scans. Tsuchinoko may integrate with Bluesky-Adaptive by serving as the source of an `Agent`.

Running Tsuchinoko with Bluesky-Adaptive requires a `TsuchinokoAgent` between the Tsuchinoko server and the Bluesky RunEngine. The `TsuchinokoAgent` abstract base class provides a concise interface requiring the same components as a Bluesky-Adaptive `Agent`.

INDEX