# Parameters

The following classes of the `twinlab Parameter` function define parameters that can be use to further refine functionality for the respective twinLab function.

## Parameter classes

| | |
|---|---|
| `DesignParams`([sampling_method, seed]) | Parameter configuration to setup an initial experimental or simulations design structure. |
| `EstimatorParams`([detrend, covar_module, ...]) | Parameter configuration for the Gaussian Process emulator (estimator). |
| `ModelSelectionParams`([seed, ...]) | Parameter configuration for the Bayesian model selection process. |
| `TrainParams`([estimator, estimator_params, ...]) | Parameter configuration for training an emulator. |
| `ScoreParams`([metric, combined_score]) | Parameter configuration for scoring a trained emulator. |
| `BenchmarkParams`([type]) | Parameter configuration for benchmarking a trained emulator. |
| `PredictParams`([observation_noise]) | Parameter configuration for making predictions using a trained emulator. |
| `SampleParams`([seed, fidelity]) | Parameter configuration for sampling from a trained emulator. |
| `AcqFuncParams`(*args, **kwargs) | |
| `OptimiserParams`(*args, **kwargs) | |
| `RecommendParams`([weights, num_restarts, ...]) | Parameter configuration for recommending new points to sample using the Bayesian-optimisation routine. |

Skip to main content

| | |
|---|---|
| `CalibrateParams`([y_std_model, ...]) | Parameter configuration for inverting a trained emulator to estimate the input parameters that generated a given output. |

Previous

Next

# Utility Functions

The following functions are contained in the `twinlab Core` functionality

## API Key

| | |
|---|---|
| `set_api_key`(api_key[, verbose]) | Set the user API key for their twinLab cloud account. |
| `get_api_key`([verbose]) | Show the user API key for their twinLab cloud account. |

## Server URL

| | |
|---|---|
| `set_server_url`(url[, verbose]) | Set the server URL for twinLab. |
| `get_server_url`([verbose]) | Show the URL from which twinLab is currently being accessed. |

## User information

| | |
|---|---|
| `versions`([verbose]) | Get information about the twinLab version being used. |
| `user_information`([verbose]) | Get information about the twinLab user. |

## Datasets and Emulators

| | |
|---|---|
| `list_emulators`([verbose]) | List trained emulators that exist in the user's twinLab cloud account. |
| `list_datasets`([verbose]) | List datasets that have been uploaded to the user's twinLab cloud account. |

Skip to main content

| | |
|---|---|
| `list_example_datasets`([verbose]) | List example datasets that are available on the twinLab cloud. |
| `load_example_dataset`(dataset_id[, verbose]) | Load an example dataset from the twinLab cloud into the current session. |

## Helper functions

| | |
|---|---|
| `get_sample`(df, key) | Retrieve an individual sample from the multi-indexed dataframe returned by the `Emulator.sample()` method. |
| `join_samples`(df_one, df_two) | Join two dataframes that contain independent samples generated by the `Emulator.sample()` method. |
| `load_dataset`(filepath[, verbose]) | Load a dataset from a local file in `.csv` format into a pandas dataframe. |
| `load_params`(filepath[, verbose]) | Load a parameter set from a local file in `.json` format into a dictionary. |

Previous
twinlab.sampling.UniformRandom
Next
twinlab.set_api_key



Created using Sphinx 7.3.7.

Built with the PyData Sphinx Theme 0.15.2.

# Distributions

The following functions are contained in the ``twinlab Distribution` functions.

## Distributions

| | |
|---|---|
| `Prior`(name, distribution) | A prior probability distribution |
| `distributions` | |
| `distributions.Uniform`(min, max) | A one-dimensional continous uniform distribution between a minimum and maximum value. |

## Sampling

| | |
|---|---|
| `Sampling`() | |
| `sampling.LatinHypercube`([scramble, optimization]) | A sampling strategy that uses Latin Hypercube Sampling. |
| `sampling.UniformRandom`() | A sampling strategy that random-uniformly samples the space. |

“”

Previous
twinlab.CalibrateParams

Next
twinlab.Prior

# Documentation

Welcome to the twinLab python documentation!

Please see the exposed capabilities for creating models:

### Dataset

Explore how to import and manipulate your data

Explore →

### Emulators

Learn how to use twinLab to build probabalistic emulators

Explore →

### Parameters

See our parameter classes which enable hyper-users to fine tune their models.

Explore →

### Utilities

See the twinLab core and utility functions

Explore →

**See also**

To explore how twinLab is used in the wide explore our notebook Examples

***Deprecated since version V2.0.0:*** A previous version of twinLab (pre December 2023) is still available as Version 1 Documentation

Previous
API Key

Next
Dataset



Created using Sphinx 7.3.7.

Built with the PyData Sphinx Theme 0.15.2.

# Dataset

The following functions are contained in the `twinlab.Dataset` class.

## Constructor

| | |
|---|---|
| `Dataset`(id) | A twinLab dataset that can be used for training an emulator online. |

## Upload data

| | |
|---|---|
| `Dataset.upload`(df[, verbose]) | Upload a dataset to the twinLab cloud so that it can be queried and used for training. |

## Explore data

| | |
|---|---|
| `Dataset.view`([verbose]) | View (and download) a dataset that exists on the twinLab cloud. |
| `Dataset.summarise`([verbose]) | Show summary statistics for a dataset that exists on the twinLab cloud. |
| `Dataset.analyse_variance`(columns[, verbose]) | Return an analysis of the variance retained per dimension. |
| `Dataset.analyse_input_variance`(columns[, ...]) | ***Deprecated since version 2.5.0.*** |
| `Dataset.analyse_output_variance`(columns[, ...]) | ***Deprecated since version 2.5.0.*** |

Skip to main content

# Delete data

| | |
|---|---|
| `Dataset.delete`([verbose]) | Delete a dataset that was previously uploaded to the twinLab cloud. |

Previous
**Documentation**

Next
**twinlab.Dataset**



Created using Sphinx 7.3.7.

Built with the PyData Sphinx Theme 0.15.2.

# Utility Functions

The following functions are contained in the `twinlab Core` functionality

## API Key

| | |
|---|---|
| `set_api_key`(api_key[, verbose]) | Set the user API key for their twinLab cloud account. |
| `get_api_key`([verbose]) | Show the user API key for their twinLab cloud account. |

## Server URL

| | |
|---|---|
| `set_server_url`(url[, verbose]) | Set the server URL for twinLab. |
| `get_server_url`([verbose]) | Show the URL from which twinLab is currently being accessed. |

## User information

| | |
|---|---|
| `versions`([verbose]) | Get information about the twinLab version being used. |
| `user_information`([verbose]) | Get information about the twinLab user. |

## Datasets and Emulators

| | |
|---|---|
| `list_emulators`([verbose]) | List trained emulators that exist in the user's twinLab cloud account. |
| `list_datasets`([verbose]) | List datasets that have been uploaded to the user's twinLab cloud account. |

Skip to main content

| | |
|---|---|
| `list_example_datasets`([verbose]) | List example datasets that are available on the twinLab cloud. |
| `load_example_dataset`(dataset_id[, verbose]) | Load an example dataset from the twinLab cloud into the current session. |

## Helper functions

| | |
|---|---|
| `get_sample`(df, key) | Retrieve an individual sample from the multi-indexed dataframe returned by the `Emulator.sample()` method. |
| `join_samples`(df_one, df_two) | Join two dataframes that contain independent samples generated by the `Emulator.sample()` method. |
| `load_dataset`(filepath[, verbose]) | Load a dataset from a local file in `.csv` format into a pandas dataframe. |
| `load_params`(filepath[, verbose]) | Load a parameter set from a local file in `.json` format into a dictionary. |

Previous
twinlab.sampling.UniformRandom
Next
twinlab.set_api_key



Created using Sphinx 7.3.7.

Built with the PyData Sphinx Theme 0.15.2.

# Documentation

Welcome to the twinLab python documentation!

Please see the exposed capabilities for creating models:

### Dataset

Explore how to import and manipulate your data

Explore →

### Emulators

Learn how to use twinLab to build probabalistic emulators

Explore →

### Parameters

See our parameter classes which enable hyper-users to fine tune their models.

Explore →

### Utilities

See the twinLab core and utility functions

Explore →

**See also**

To explore how twinLab is used in the wide explore our notebook Examples

***Deprecated since version V2.0.0:*** A previous version of twinLab (pre December 2023) is still available as Version 1 Documentation

Previous
API Key

Next
Dataset



Created using Sphinx 7.3.7.

Built with the PyData Sphinx Theme 0.15.2.

# Dataset

The following functions are contained in the `twinlab.Dataset` class.

## Constructor

| | |
|---|---|
| `Dataset`(id) | A twinLab dataset that can be used for training an emulator online. |

## Upload data

| | |
|---|---|
| `Dataset.upload`(df[, verbose]) | Upload a dataset to the twinLab cloud so that it can be queried and used for training. |

## Explore data

| | |
|---|---|
| `Dataset.view`([verbose]) | View (and download) a dataset that exists on the twinLab cloud. |
| `Dataset.summarise`([verbose]) | Show summary statistics for a dataset that exists on the twinLab cloud. |
| `Dataset.analyse_variance`(columns[, verbose]) | Return an analysis of the variance retained per dimension. |
| `Dataset.analyse_input_variance`(columns[, ...]) | ***Deprecated since version 2.5.0.*** |
| `Dataset.analyse_output_variance`(columns[, ...]) | ***Deprecated since version 2.5.0.*** |

Skip to main content

# Delete data

| | |
|---|---|
| `Dataset.delete`([verbose]) | Delete a dataset that was previously uploaded to the twinLab cloud. |

Previous
**Documentation**

Next
**twinlab.Dataset**



Created using Sphinx 7.3.7.

Built with the PyData Sphinx Theme 0.15.2.

# Emulator

The following functions are contained in the `twinlab.Emulator` class.

## Constructor

| | |
|---|---|
| `Emulator`(id) | A trainable twinLab emulator. |

## Setting-up

| | |
|---|---|
| `Emulator.design`(priors, num_points[, ...]) | Generate an initial design space for an emulator. |

## Train

| | |
|---|---|
| `Emulator.train`(dataset, inputs, outputs[, ...]) | Train an emulator on the twinLab cloud. |
| `Emulator.status`(process_id[, verbose]) | Check the status of a training process on the twinLab cloud. |
| `Emulator.view`([verbose]) | View an emulator that exists on the twinLab cloud. |
| `Emulator.view_train_data`([verbose]) | View training data with which the emulator was trained in the twinLab cloud. |
| `Emulator.view_test_data`([verbose]) | View test data on which the emulator was tested in the twinLab cloud. |

## Explore

| | |
|---|---|
| `Emulator.summarise`([verbose]) | Get a summary of a trained emulator on the twinLab cloud. |

Skip to main content

| | |
|---|---|
| `Emulator.score`([params, verbose]) | Score the performance of a trained emulator. |
| `Emulator.benchmark`([params, verbose]) | Benchmark the performance of a trained emulator with a calibration curve. |

## Predict

| | |
|---|---|
| `Emulator.predict`(df[, params, wait, verbose]) | Make predictions using a trained emulator that exists on the twinLab cloud. |
| `Emulator.sample`(df, num_samples[, params, ...]) | Draw samples from a trained emulator that exists on the twinLab cloud. |

## Plot

| | |
|---|---|
| `Emulator.plot`(x_axis, y_axis[, x_fixed, ...]) | Plot the predictions from an emulator across a single dimension with one and two standard deviation bands. |
| `Emulator.heatmap`(x1_axis, x2_axis, y_axis[, ...]) | Plot a heatmap of the predictions from an emulator across two dimensions. |

## Improve

| | |
|---|---|
| `Emulator.recommend`(num_points, acq_func[, ...]) | Draw new recommended data points from a trained emulator that exists on the twinLab cloud. |
| `Emulator.learn`(dataset, inputs, outputs, ...) | Perform active learning to improve an emulator on the twinLab cloud. |
| `Emulator.calibrate`(df_obs, df_std[, params, ...]) | Solve an inverse problem using a trained emulator on the twinLab cloud. |

## Delete

| | |
|---|---|
| `Emulator.delete`([verbose]) | Delete emulator from the twinLab cloud. |

Skip to main content

# Documentation

Welcome to the twinLab python documentation!

Please see the exposed capabilities for creating models:

### Dataset

Explore how to import and manipulate your data

Explore →

### Emulators

Learn how to use twinLab to build probabalistic emulators

Explore →

### Parameters

See our parameter classes which enable hyper-users to fine tune their models.

Explore →

### Utilities

See the twinLab core and utility functions

Explore →

**See also**

To explore how twinLab is used in the wide explore our notebook Examples

***Deprecated since version V2.0.0:*** A previous version of twinLab (pre December 2023) is still available as Version 1 Documentation

Previous
API Key

Next
Dataset



Created using Sphinx 7.3.7.

Built with the PyData Sphinx Theme 0.15.2.

# Parameters

The following classes of the `twinlab Parameter` function define parameters that can be use to further refine functionality for the respective twinLab function.

## Parameter classes

| | |
|---|---|
| `DesignParams`([sampling_method, seed]) | Parameter configuration to setup an initial experimental or simulations design structure. |
| `EstimatorParams`([detrend, covar_module, ...]) | Parameter configuration for the Gaussian Process emulator (estimator). |
| `ModelSelectionParams`([seed, ...]) | Parameter configuration for the Bayesian model selection process. |
| `TrainParams`([estimator, estimator_params, ...]) | Parameter configuration for training an emulator. |
| `ScoreParams`([metric, combined_score]) | Parameter configuration for scoring a trained emulator. |
| `BenchmarkParams`([type]) | Parameter configuration for benchmarking a trained emulator. |
| `PredictParams`([observation_noise]) | Parameter configuration for making predictions using a trained emulator. |
| `SampleParams`([seed, fidelity]) | Parameter configuration for sampling from a trained emulator. |
| `AcqFuncParams`(*args, **kwargs) | |
| `OptimiserParams`(*args, **kwargs) | |
| `RecommendParams`([weights, num_restarts, ...]) | Parameter configuration for recommending new points to sample using the Bayesian-optimisation routine. |

Skip to main content

| | |
|---|---|
| `CalibrateParams`([y_std_model, ...]) | Parameter configuration for inverting a trained emulator to estimate the input parameters that generated a given output. |

Previous

Next

# Emulator

The following functions are contained in the `twinlab.Emulator` class.

## Constructor

| | |
|---|---|
| `Emulator`(id) | A trainable twinLab emulator. |

## Setting-up

| | |
|---|---|
| `Emulator.design`(priors, num_points[, ...]) | Generate an initial design space for an emulator. |

## Train

| | |
|---|---|
| `Emulator.train`(dataset, inputs, outputs[, ...]) | Train an emulator on the twinLab cloud. |
| `Emulator.status`(process_id[, verbose]) | Check the status of a training process on the twinLab cloud. |
| `Emulator.view`([verbose]) | View an emulator that exists on the twinLab cloud. |
| `Emulator.view_train_data`([verbose]) | View training data with which the emulator was trained in the twinLab cloud. |
| `Emulator.view_test_data`([verbose]) | View test data on which the emulator was tested in the twinLab cloud. |

## Explore

| | |
|---|---|
| `Emulator.summarise`([verbose]) | Get a summary of a trained emulator on the twinLab cloud. |

Skip to main content

| | |
|---|---|
| `Emulator.score`([params, verbose]) | Score the performance of a trained emulator. |
| `Emulator.benchmark`([params, verbose]) | Benchmark the performance of a trained emulator with a calibration curve. |

## Predict

| | |
|---|---|
| `Emulator.predict`(df[, params, wait, verbose]) | Make predictions using a trained emulator that exists on the twinLab cloud. |
| `Emulator.sample`(df, num_samples[, params, ...]) | Draw samples from a trained emulator that exists on the twinLab cloud. |

## Plot

| | |
|---|---|
| `Emulator.plot`(x_axis, y_axis[, x_fixed, ...]) | Plot the predictions from an emulator across a single dimension with one and two standard deviation bands. |
| `Emulator.heatmap`(x1_axis, x2_axis, y_axis[, ...]) | Plot a heatmap of the predictions from an emulator across two dimensions. |

## Improve

| | |
|---|---|
| `Emulator.recommend`(num_points, acq_func[, ...]) | Draw new recommended data points from a trained emulator that exists on the twinLab cloud. |
| `Emulator.learn`(dataset, inputs, outputs, ...) | Perform active learning to improve an emulator on the twinLab cloud. |
| `Emulator.calibrate`(df_obs, df_std[, params, ...]) | Solve an inverse problem using a trained emulator on the twinLab cloud. |

## Delete

| | |
|---|---|
| `Emulator.delete`([verbose]) | Delete emulator from the twinLab cloud. |

Skip to main content

# Version 1 Documentation

## Functionality

| | |
|---|---|
| `active_learn_campaign`(campaign_id, num_points) | Active learn campaign |
| `train_campaign`(filepath_or_params, campaign_id) | Train a campaign in the *twinLab* cloud. |
| `list_campaigns`([verbose, debug]) | List campaigns |
| `delete_campaign`(campaign_id[, verbose, debug]) | Delete campaign |
| `optimise_campaign`(campaign_id, num_points[, ...]) | Optimise campaign |
| `predict_campaign`(filepath_or_df, campaign_id) | Predict campaign |
| `sample_campaign`(filepath_or_df, campaign_id, ...) | Sample campaign |
| `score_campaign`(campaign_id[, ...]) | Quantify the performance of your trained model with a model score. |
| `get_calibration_curve_campaign`(campaign_id) | Quantify the performance of your trained model with a calibration curve. |

# twinlab.Sampling

*class* twinlab.**Sampling**

**__init__**()

### Methods

| | |
|---|---|
| **__init__**() | |
| **to_json**(sampling_params) | |

Previous
twinlab.distributions.Uniform

Next
twinlab.sampling.LatinHypercube

# twinlab.get_api_key

**twinlab.get_api_key**(*verbose=False*)

Show the user API key for their twinLab cloud account.

**Parameters:**

**verbose** (*bool, optional*) – Display information about the operation while running.

**Returns:**

User API key.

**Return type:**

str

### Example

```
tl.get_api_key()
```

```
'secret-12345'
```

Previous
twinlab.set_api_key

Next
twinlab.set_server_url

# twinlab.user_information

**twinlab.user_information**(*verbose=False*)

Get information about the twinLab user.

**Parameters:**

**verbose** (*bool, optional*) – Display information about the operation while running.

**Returns:**

User information.

**Return type:**

dict

### Example

```
user_info = tl.user_information()
print(user_info)
```

```
{'username': 'dodders@digilab.co.uk', 'credits': 850}
```

Previous
twinlab.versions

Next
twinlab.list_emulators

# twinlab.Emulator.plot

**Emulator.plot(*x_axis, y_axis, x_fixed={}, params=<twinlab.params.PredictParams object>, x_lim=None, n_points=100, label='Emulator', color='#009FE3', verbose=False*)**

Plot the predictions from an emulator across a single dimension with one and two standard deviation bands.

This will make a call to the emulator to predict across the specified dimension. Note that a multi-dimensional emulator will be sliced across the other dimensions. The matplotlib.pyplot object is returned, and can be further modified by the user.

**Parameters:**

- **x_axis** (*str*) – The name of the x-axis variable.
- **y_axis** (*str*) – The name of the y-axis variable.
- **x_fixed** (*dict, optional*) – A dictionary of fixed values for the other X variables. Note that all X variables of an emulator must either be specified as x_axis or appear as x_fixed keys. To pass through “None”. either leave x_fixed out or pass through an empty dictionary.
- **params** (`PredictParams`) – (PredictParams, optional). A parameter configuration that contains optional prediction parameters.
- **(tuple[float** (*x_lim*) – The limits of the x-axis. If not provided. the limits will be taken directly from the emulator.
- **float]** – The limits of the x-axis. If not provided. the limits will be taken directly from the emulator.
- **optional]** – The limits of the x-axis. If not provided. the limits will be taken directly from the emulator.
- **n_points** (*int, optional*) – The number of points to sample in the x-axis.
- **label** (*str, optional*) – The label for the line in the plot. defaults to “Emulator prediction”.
- **color** (*str, optional*) – The color of the plot. Defaults to digiLab blue. Can be any valid matplotlib color (https://matplotlib.org/stable/gallery/color/named_colors.html).
- **verbose** (*bool, optional*) – Display detailed information about the operation while running.

**Return type:**

`plot`

### Examples

```
emulator = tl.Emulator("emulator_id")
plt = emulator.plot("Time", "Temperature", x_fixed={"Latitude": 0, "Longitude": 30})
plt.show()
```

Previous
twinlab.Emulator.sample

Next
twinlab.Emulator.heatmap

# twinlab.Emulator.design

**Emulator.design(*priors*, *num_points*, *params=<twinlab.params.DesignParams object>*, *verbose=False*)**

Generate an initial design space for an emulator.

The method is used to generate an initial design for evaluating a set of experiments, emulator, or simulation. This is useful if data has not yet been collected and a user wants to generate an initial design space to train an emulator. Optimal space-filling methods can be used to generate an initial design space that are significantly better than either random or grid sampling. If data has already been acquired then an initial emulator can be trained using `Emulator.train()` and new sampling locations can be recommended using `Emulator.recommend()`.

**Parameters:**

- **priors** (*list[Prior]*) – A list of `Prior` objects that define the prior distributions for each input. These are independent one-dimensional probability distributions for each parameter.
- **num_points** (*int*) – The number of points to sample in designing the initial space.
- **params** (*twinlab.DesignParams*, *optional*) – A parameter configuration that contains all of the optional initial-design parameters.

**Return type:**

`DataFrame`

### Example

```
emulator = tl.Emulator("emulator_id")

my_priors = [
    tl.Prior("x1", tl.distributions.Uniform(0, 12)),
    tl.Prior("x2", tl.distributions.Uniform(0, 0.5)),
    tl.Prior("x3 ", tl.distributions.Uniform(0, 10)),
]

initial_design = emulator.design(my_priors, 10)
```

Created using Sphinx 7.3.7.

Built with the PyData Sphinx Theme 0.15.2.

# twinlab.set_server_url

**twinlab.set_server_url(*url*, *verbose*=*False*)**

Set the server URL for twinLab.

If this is not set the default URL is used: `https://twinlab.digilab.co.uk`. This default URL will be correct for most users and should not normally need to be changed. Setting this will override the URL set in the environment variable `TWINLAB_URL` in the `.env` file for the current session.

**Parameters:**

- **url** (*str*) – URL for the twinLab cloud.
- **verbose** (*bool, optional*) – Display information about the operation while running.

**Return type:**

`None`

**Example**

```
tl.set_server_url("https://twinlab.digilab.co.uk/stage")
```

Previous
twinlab.get_api_key

Next
twinlab.get_server_url

# twinlab.OptimiserParams

*class* twinlab.**OptimiserParams**(**args*, ***kwargs*)

**__init__**()

### Methods

| | |
|---|---|
| __init__() | |
| unpack_parameters() | |

Previous
twinlab.AcqFuncParams

Next
twinlab.RecommendParams

# twinlab.distributions.Uniform

***class*** **twinlab.distributions.Uniform(*min*, *max*)**

A one-dimensional continous uniform distribution between a minimum and maximum value.

**Parameters:**

- **min** (*float*) – The minimum value of the distribution.
- **max** (*float*) – The maximum value of the distribution.

**__init__(*min*, *max*)**

**Methods**

| | |
|---|---|
| `__init__` | (min, max) |
| `to_json` | () |

Previous
twinlab.distributions

Next
twinlab.Sampling

# twinlab.sampling.UniformRandom

*class* twinlab.sampling.**UniformRandom**

A sampling strategy that random-uniformly samples the space.

**__init__()**

### Methods

| |
|---|
| __init__() |
| to_json() |

Previous
twinlab.sampling.LatinHypercube

Next
Utility Functions

# twinlab.list_emulators

**twinlab.list_emulators(*verbose=False*)**

List trained emulators that exist in the user's twinLab cloud account.

These trained emulators can be used for a variety of inference operations (see methods of the Emulator class). Setting `verbose=True` will also show the state of currently training emulators, as well as those that have failed to train. Information about start time and current/final run time will also be shown if `verbose=True`.

**Parameters:**

**verbose** (*bool, optional*) – Display information about the emulators while running.

**Returns:**

Currently trained emulators.

**Return type:**

list

### Example

```
emulators = tl.list_emulators()
print(emulators)
```

```
['biscuits', 'gardening', 'new-emulator', 'my-emulator']
```

Previous
twinlab.user_information

Next
twinlab.list_datasets

# twinlab.Dataset.analyse_input_variance

**Dataset.analyse_input_variance(*columns*, *verbose*=*False*)**

**Deprecated since version 2.5.0:** This method is being deprecated. Please use the method `Dataset.analyse_variance()` to analyse input or output variance.

**Return type:**

DataFrame

Previous
twinlab.Dataset.analyse_variance

Ne
twinlab.Dataset.analyse_output_varianc

# twinlab.Dataset.summarise

**Dataset.summarise(*verbose=False*)**

Show summary statistics for a dataset that exists on the twinLab cloud.

**Parameters:**

**verbose** (*bool, optional*) – Display information about the operation while running.

**Returns:**

DataFrame with summary statistics.

**Return type:**

pandas.DataFrame

### Example

```
dataset = tl.Dataset("my_dataset")
dataset.summarise()
```

```
                x          y
count  10.000000  10.000000
mean    0.544199   0.029383
std     0.229352   0.748191
min     0.226851  -0.960764
25%     0.399865  -0.694614
50%     0.516123   0.087574
75%     0.693559   0.734513
max     0.980764   0.921553
```

Previous
twinlab.Dataset.view

Next
twinlab.Dataset.analyse_variance

# twinlab.Emulator.learn

**Emulator.learn(*dataset, inputs, outputs, num_loops, num_points_per_loop, acq_func, simulation, train_params=<twinlab.params.TrainParams object>, recommend_params=<twinlab.params.RecommendParams object>, verbose=True*)**

Perform active learning to improve an emulator on the twinLab cloud.

Active learning is a method that can identify and utilise the most informative data points to add to an emulator in order to reduce the number of measurements to be taken or simulations that are required. Using active learning can result in a more accurate model, trained with less data. The primary difference between this method and `Emulator.recommend` is that in this method, the emulator is trained, new data points are suggested, and then training occurs continuously in an active loop. This way, new data can be used to train and update an emulator until the desired level of accuracy is achieved. This can be done using either the `"optmise"` or `"explore"` acquisition functions. The emulator is therefore updated on the twinLab cloud with the objective of either finding the point of maximum output or reducing the overall uncertainty in the emulator. This method does not return anything to the user directly, but instead updates the `Dataset` and `Emulator` in the cloud.

**Parameters:**

- **dataset** (*Dataset*) – twinLab dataset object which contains the initial training data for the emulator.
- **inputs** (*list[str]*) – List of input column names in the training dataset.
- **outputs** (*list[str]*) – List of output column names in the training dataset.
- **num_loops** (*int*) – Number of loops to run of the learning process. This must be a positive integer. Note that in this method, the emulator is trained and then re-trained on new suggested data points, so setting `num_loops=1` here will mean that `Emulator.train` is run _twice_, and `Emulator.recommend` is run once.
- **num_points_per_loop** (*int*) – Number of points to sample in each loop.
- **acq_func** (*str*) – Specifies the acquisition function to be used when recommending new points: either `"explore"` or `"optimise"`.
- **simulation** (*Callable*) – A function that takes in a set of inputs and generates the outputs (for example, a simulator for the data generating process).
- **train_params** (*TrainParams, optional*) – A parameter configuration that contains optional training

running a learning loop.

- **recommend_params** (*RecommendParams, optional*) – A parameter configuration that contains optional recommendation parameters.
- **verbose** (*bool, optional*) – Display detailed information about the operation while running. If `True`, the requested candidate points will be printed to the screen while running. If `False` the emulator will be updated on the cloud while the method runs silently.

**Return type:**

`None`

### Examples

```
emulator = tl.Emulator("quickstart")
dataset = tl.Dataset("quickstart")
emulator.learn(
    dataset=dataset,
    inputs=["x"],
    outputs=["y"],
    num_loops=3,
    num_points_per_loop=5,
    acq_func="explore",
    simulation=my_simulator,
)
```

Previous
twinlab.Emulator.recommend

Next
twinlab.Emulator.calibrate

# twinlab.Emulator.view_train_data

**Emulator.view_train_data(*verbose=False*)**

View training data with which the emulator was trained in the twinLab cloud.

**Parameters:**

**verbose** (*bool, optional*) – Display information about the operation while running.

**Returns:**

Dataframe containing the training data on which the emulator was trained

**Return type:**

pandas.DataFrame

### Example

```
emulator = tl.Emulator("quickstart")
emulator.view_train_data()
```

```
          x         y
0  0.696469 -0.817374
1  0.286139  0.887656
2  0.226851  0.921553
3  0.551315 -0.326334
4  0.719469 -0.832518
5  0.423106  0.400669
6  0.980764 -0.164966
7  0.684830 -0.960764
```

Previous
twinlab.Emulator.view

Next
twinlab.Emulator.view_test_data

# twinlab.load_example_dataset

**twinlab.load_example_dataset(*dataset_id*, *verbose*=*False*)**

Load an example dataset from the twinLab cloud into the current session.

**Parameters:**

- **dataset_id** (*str*) – The name of the dataset to load.
- **verbose** (*bool, optional*) – Display information about the operation while running.

**Returns:**

The example dataset.

**Return type:**

pandas.DataFrame

### Example

```
tl.load_example_dataset("quickstart")
```

```
          x         y
0  0.696469 -0.817374
1  0.286139  0.887656
2  0.226851  0.921553
3  0.551315 -0.326334
4  0.719469 -0.832518
5  0.423106  0.400669
6  0.980764 -0.164966
7  0.684830 -0.960764
8  0.480932  0.340115
9  0.392118  0.845795
```

Created using Sphinx 7.3.7.

Built with the PyData Sphinx Theme 0.15.2.

# twinlab.set_api_key

**twinlab.set_api_key(*api_key*, *verbose=False*)**

Set the user API key for their twinLab cloud account.

Setting this will override the API key set in the environment variable `TWINLAB_API_KEY` in the .env file for the current session. This can be used instead of setting the API key in the `.env` file and negates the need for a `.env` file.

**Parameters:**

- **api_key** (*str*) – API key to use for access to the twinLab cloud.
- **verbose** (*bool, optional*) – Display information about the operation while running.

**Return type:**

`None`

### Example

```
tl.set_api_key("12345")
```

Previous
Utility Functions

Next
twinlab.get_api_key

# twinlab.PredictParams

***class*** **twinlab.PredictParams(*observation_noise=True*)**

Parameter configuration for making predictions using a trained emulator.

**Variables:**

**observation_noise** (*bool*, *optional*) – Whether or not to include the noise term in the standard deviation of the prediction. Setting this to `False` can be a good idea if the training data is noisy but the underlying trend of the trained model is smooth. In this case, the predictions would correspond to the underlying trend. Setting this to `True` can be a good idea to see the spread of possible predictions including the noise. This is useful to know the true uncertainty in the data-generation process, and to estimate the true span of values that might be observed next. The default value is `True`.

**__init__(*observation_noise=True*)**

### Methods

| | |
|---|---|
| `__init__`([observation_noise]) | |
| `unpack_parameters`() | |

Previous
twinlab.BenchmarkParams

Next
twinlab.SampleParams

# twinlab.Emulator.summarise

**Emulator.summarise(*verbose=False*)**

Get a summary of a trained emulator on the twinLab cloud.

This summary returns transformer diagnostics, with details about the input/output decomposition. It also returns the estimator diagnostics, detailing properties of the trained emulator. This information can help inform a user about the makeup of an emulator – for example, what kind of kernel was used.

**Parameters:**

**verbose** (*bool, optional*) – Display information about the operation while running.

**Return type:**

`dict`

### Example

```
emulator = tl.Emulator("quickstart")
emulator.summarise()
```

```
{
    'estimator_diagnostics': ...,
    'transformer_diagnostics': ...,
}
```

Previous
twinlab.Emulator.view_test_data

Next
twinlab.Emulator.score

# twinlab.list_example_datasets

**twinlab.list_example_datasets(*verbose=False*)**

List example datasets that are available on the twinLab cloud.

These datasets can be downloaded and used for training emulators and other operations and are used in many of the examples in the documentation. Datasets can be loaded using the `load_example_dataset` function.

**Parameters:**

**verbose** (*bool, optional*) – Display information about the operation while running.

**Returns:**

Example datasets available for loading.

**Return type:**

list

### Example

```
example_datasets = tl.list_example_datasets()
print(example_datasets)
```

```
['biscuits', 'gardening', 'quickstart', 'tritium-desorption']
```

Previous
twinlab.list_datasets

Next
twinlab.load_example_dataset

# twinlab.Dataset

***class* twinlab.Dataset(*id*)**

A twinLab dataset that can be used for training an emulator online.

**Parameters:**

**id** (*str*) – Name of the dataset.

### Example

```
dataset = tl.Dataset("my_dataset")
```

## __init__(*id*)

### Methods

| | |
|---|---|
| __init__(id) | |
| analyse_input_variance(columns[, verbose]) | ***Deprecated since version 2.5.0.*** |
| analyse_output_variance(columns[, verbose]) | ***Deprecated since version 2.5.0.*** |
| analyse_variance(columns[, verbose]) | Return an analysis of the variance retained per dimension. |
| delete([verbose]) | Delete a dataset that was previously uploaded to the twinLab cloud. |

| | |
|---|---|
| `summarise`([verbose]) | Show summary statistics for a dataset that exists on the twinLab cloud. |
| `upload`(df[, verbose]) | Upload a dataset to the twinLab cloud so that it can be queried and used for training. |
| `view`([verbose]) | View (and download) a dataset that exists on the twinLab cloud. |

Previous
**Dataset**

Next
**twinlab.Dataset.upload**

# twinlab.Emulator.train

**Emulator.train(*dataset*, *inputs*, *outputs*, *params=<twinlab.params.TrainParams object>*, *wait=True*, *verbose=False*)**

Train an emulator on the twinLab cloud.

This is the primary functionality of twinLab, where an emulator is trained on a dataset. The emulator learns trends in the dataset and then is able to make predictions on new data. These new data can be far away from the training data, and the emulator will interpolate between the training data points. The emulator can also be used to extrapolate beyond the training data, but this is less reliable. The emulator can be trained on a dataset with multiple inputs and outputs, and can be used to make predictions on new data with multiple inputs and outputs. The powerful algorithms in twinLab allow for the emulator to not only make predictions, but to also quantify the uncertainty in these predictions. This is extremely advantageous, because it allows for the reliability of the predictions to be quantified.

**Parameters:**

- **dataset** (*Dataset*) – The training and test data for the emulator. The ratio of train to test data can be set in `TrainParams`.
- **inputs** (*list[str]*) – A list of the input column names in the training dataset. These correspond to the independent variables in the dataset, which are often the parameters of a model. These are usually known as `X` (note that capital) values.
- **outputs** (*list[str]*) – A list of the output column names in the training dataset. These correspond to the dependent variables in the dataset, which are often the results of a model. These are usually known as `y` values.
- **params** (*TrainParams*, *optional*) – A training parameter configuration that contains all optional training parameters.
- **wait** (*bool*, *optional*) – If `True` wait for the job to complete, otherwise return the process ID and exit. Setting `wait=False` is useful for longer training jobs.
- **verbose** (*bool*, *optional*) – Display information about the operation while running.

**Return type:**

`Optional`[`str`]

**Returns:**

If `wait=True` the function will run until the emulator is trained on the cloud. If `wait=False` the function will return the process ID and exit. This is useful for longer training jobs. The training status can then be checked later using `Emulator.status()`.

### Example

```
df = pd.DataFrame({"X": [1, 2, 3, 4], "y": [1, 4, 9, 16]})
dataset = tl.Dataset("my_dataset")
dataset.upload(df)
emulator = tl.Emulator("my_emulator")
emulator.train(dataset, ["X"], ["y"])
```

Previous
twinlab.Emulator.design

Next
twinlab.Emulator.status

# twinlab.load_params

**twinlab.load_params(*filepath*, *verbose*=*False*)**

Load a parameter set from a local file in `.json` format into a dictionary.

**Parameters:**

- **filepath** (*str*) – Path to the dataset file, which should be in json format.
- **verbose** (*bool, optional*) – Display information while running.

**Returns:**

The parameter set loaded from the file.

**Return type:**

dict

### Example

```
params = tl.load_params("path/to/params.json", verbose=True)
```

Previous
twinlab.load_dataset

Next
Examples

# twinlab.Dataset.analyse_variance

**Dataset.analyse_variance(*columns*, *verbose*=*False*)**

Return an analysis of the variance retained per dimension.

**Parameters:**

- **columns** (*list*[*str*]) – List of columns to evaluate. This would usually be either the set of input or output columns.
- **verbose** (*bool*, *optional*) – Display information about the operation while running.

**Return type:**

DataFrame

**Returns:**

pandas.Dataframe containing the variance analysis.

### Example

```
dataset = tl.Dataset("quickstart")
dataset.analyse_variance(columns=["x", "y"]) # Usually the columns would be either inpu
```

```
   Number of Dimensions  Cumulative Variance
0                     0             0.000000
1                     1             0.925741
2                     2             1.000000
```

Previous
twinlab.Dataset.summarise

Next
twinlab.Dataset.analyse_input_variance

# twinlab.Dataset.upload

**Dataset.upload(*df*, *verbose*=*False*)**

Upload a dataset to the twinLab cloud so that it can be queried and used for training.

Please note the largest dataset that can be uploaded is currently 5GB.

When using twinLab emulators, note that training time scales cubically with the amount of data included. It may be worthwhile training with a smaller subset of data at first, to determine approximately how long it will take to train. Please get in touch with our experts for technical support to understand how to make best use of data.

**Parameters:**

- **df** (*pandas.DataFrame*) – A *pandas.DataFrame* containing the dataset to be uploaded.
- **verbose** (*bool, optional*) – Display information about the operation while running.

**Return type:**

None

### Example

```
dataset = tl.Dataset("my_dataset")
df = pd.DataFrame({"X": [1, 2, 3, 4], "y": [1, 4, 9, 16]})
dataset.upload(df)
```

Previous
twinlab.Dataset

Next
twinlab.Dataset.view

# twinlab.Emulator.recommend

**Emulator.recommend**(*num_points*, *acq_func*, *params=<twinlab.params.RecommendParams object>*, *wait=True*, *verbose=False*)

Draw new recommended data points from a trained emulator that exists on the twinLab cloud.

The recommend functionality of an emulator is used to suggest new data points to sample. These new data points can be chosen depending on a variety of different user objectives. Currently, the user can choose between `"optimise"` and `"explore"` acquisition functions. Choosing `"optimise"` will obtain suggested `"X"` values the evaluation of which (acquiring the corresponding `"y"`) will maximise the knowledge of the emulator about the location of the maximum. A classic use case for this would be a user trying to maximise the output of a model. For example, the maximum strenth of a pipe given a set of design parameters. Choosing `"explore"` will instead suggest `"X"` that reduce the overall uncertainty of the emulator across the entire input space. A classic use case for this would be a user trying to reduce overally uncertainty. For example, a user trying to reduce the uncertainty in the strength of a pipe across all design parameters. The number of requested data points can be specified by the user, and if this is greater than one then then recommendations are all suggested at once, and are designed to be the optmial set, as a group, to achieve the user outcome. twinLab optimises which specific acquisition function within the chosen category will be used, prioritising numerical stability based on the number of points requested.

The value of the acquisition function is also returned to the user. While this is of limited value in isolation, the trend of the acquisition function value over multiple iterations of `Recommend` can be used to understand the performance of the emulator. The `Emulator.learn` method can be used to improve the performance of an emulator iteratively.

**Parameters:**

- **num_points** (*int*) – The number of samples to draw for each row of the evaluation data.
- **acq_func** (*str*) – Specifies the acquisition function to be used when recommending new points. The acquisition function can be either `"explore"` or `"optimise"`.
- **params** (*RecommendParams*, *optional*) – A parameter configuration that contains all of the optional recommendation parameters.
- **wait** (*bool*, *optional*) – If `True` wait for the job to complete, otherwise return the process ID and

- **verbose** (*bool, optional*) – Display detailed information about the operation while running.

**Returns:**

By default, a tuple is returned containing the recommended samples and the acquisition function value. Instead, if `wait=False`, the process ID is returned. The results can then be retrieved later using `Emulator.get_process(<process_id>)`. Process IDs associated with an emulator can be found using `Emulator.list_processes()`.

**Return type:**

Tuple[pandas.DataFrame, float], str

### Example

```
emulator = tl.Emulator("quickstart")
emulator.recommend(5, "explore")
```

```
          x
0  0.852853
1  0.914091
2  0.804012
3  0.353463
4  0.595268

-0.00553509
```

### Example

```
emulator = tl.Emulator("quickstart")
emulator.recommend(3, "optimise")
```

```
          x
0  0.273920
1  0.306423
2  0.226851

0.046944751
```

Previous
twinlab.Emulator.heatmap

Next
twinlab.Emulator.learn

Skip to main content

Created using Sphinx 7.3.7.

Built with the PyData Sphinx Theme 0.15.2.

# twinlab.Emulator

***class* twinlab.Emulator(*id*)**

A trainable twinLab emulator.

An emulator is trainable model that learns the trends in a dataset. It is a machine-learning model in that it requires a dataset of inputs `X` and outputs `y` on which to be trained. In this way, it learns to mimic, or emulate, the dataset and can be used to make predictions on new data. Emulators are also often called models, surrogates, or digital twins.

**Variables:**

**id** (*str*) – The name for the emulator in the twinLab cloud. If an emulator that does not currently exist is specified, then a new emulator will be instantiated. Otherwise the corresponding emulator will be loaded from the cloud. Be sure to double check which emulators have been created using .. autofunction:: *~list_emulator*.

**__init__(*id*)**

### Methods

| | |
|---|---|
| `__init__`(id) | |
| `benchmark`([params, verbose]) | Benchmark the performance of a trained emulator with a calibration curve. |
| `calibrate`(df_obs, df_std[, params, wait, ...]) | Solve an inverse problem using a trained emulator on the twinLab cloud. |
| `delete`([verbose]) | Delete emulator from the twinLab cloud. |
| `design`(priors, num_points[, params, verbose]) | Generate an initial design space for an emulator. |
| `get_process`(process_id[, verbose]) | Get the results from a process associated with the emulator on the twinLab cloud. |

| | |
|---|---|
| `heatmap`(x1_axis, x2_axis, y_axis[, x_fixed, ...]) | Plot a heatmap of the predictions from an emulator across two dimensions. |
| `learn`(dataset, inputs, outputs, num_loops, ...) | Perform active learning to improve an emulator on the twinLab cloud. |
| `list_processes`([verbose]) | List all of the processes associated with a given emulator on the twinLab cloud. |
| `plot`(x_axis, y_axis[, x_fixed, params, ...]) | Plot the predictions from an emulator across a single dimension with one and two standard deviation bands. |
| `predict`(df[, params, wait, verbose]) | Make predictions using a trained emulator that exists on the twinLab cloud. |
| `recommend`(num_points, acq_func[, params, ...]) | Draw new recommended data points from a trained emulator that exists on the twinLab cloud. |
| `sample`(df, num_samples[, params, wait, verbose]) | Draw samples from a trained emulator that exists on the twinLab cloud. |
| `score`([params, verbose]) | Score the performance of a trained emulator. |
| `status`(process_id[, verbose]) | Check the status of a training process on the twinLab cloud. |
| `summarise`([verbose]) | Get a summary of a trained emulator on the twinLab cloud. |
| `train`(dataset, inputs, outputs[, params, ...]) | Train an emulator on the twinLab cloud. |
| `view`([verbose]) | View an emulator that exists on the twinLab cloud. |
| `view_test_data`([verbose]) | View test data on which the emulator was tested in the twinLab cloud. |
| `view_train_data`([verbose]) | View training data with which the emulator was trained in the twinLab cloud. |

Previous
Emulator

Next
twinlab.Emulator.design

Skip to main content

Created using Sphinx 7.3.7.

Built with the PyData Sphinx Theme 0.15.2.

# twinlab.load_dataset

**twinlab.load_dataset(*filepath*, *verbose*=*False*)**

Load a dataset from a local file in `.csv` format into a pandas dataframe.

**Parameters:**

- **filepath** (*str*) – Path to the dataset file, which should be in csv format.
- **verbose** (*bool, optional*) – Display information while running.

**Returns:**

The dataset loaded from the file.

**Return type:**

pd.DataFrame

### Example

```
df = tl.load_dataset("path/to/data.csv", verbose=True)
```

```
Dataset loaded:
     x         y
0  0.0  1.097485
1  1.0  0.835439
2  2.0  0.655124
```

Previous
twinlab.join_samples

Next
twinlab.load_params

# twinlab.DesignParams

***class*** twinlab.**DesignParams**(*sampling_method=<twinlab.sampling.LatinHypercube object>, seed=None*)

Parameter configuration to setup an initial experimental or simulations design structure.

**Variables:**

- **sampling_method** (*Sampling, optional*) –

  The sampling method to use for the initial design. Options are either:

  - `tl.LatinHypercube`: Populate the initial design space in a clever way such that each dimension, and projection of dimensions, are sampled evenly.
  - `tl.UniformRandom`: Randomly populate the input space, which is usually a bad idea.

- **seed** (*Union[int, None], optional*) – The seed used to initialise the random number generators for reproducibility. Setting this to an integer is good for creating reproducibile design configurations. The default is `None`, which means the seed is randomly generated each time.

**__init__**(*sampling_method=<twinlab.sampling.LatinHypercube object>, seed=None*)

### Methods

| | |
|---|---|
| `__init__`([sampling_method, seed]) | |

Previous
Parameters

Next
twinlab.EstimatorParams

# twinlab.distributions

## Classes

| | |
|---|---|
| `Distribution`() | |
| `DistributionMethods`(value) | An enumeration. |
| `Uniform`(min, max) | A one-dimensional continous uniform distribution between a minimum and maximum value. |

Previous
twinlab.Prior

Next
twinlab.distributions.Uniform

# twinlab.Dataset.analyse_output_variance

**Dataset.analyse_output_variance**(*columns*, *verbose=False*)

**Deprecated since version 2.5.0:** This method is being deprecated. Please use the method `Dataset.analyse_variance()` to analyse input or output variance.

**Return type:**

DataFrame

Previous
twinlab.Dataset.analyse_input_varianc

Next
twinlab.Dataset.delete

# twinlab.sampling.LatinHypercube

***class*** **twinlab.sampling.LatinHypercube(*scramble=True*, *optimization='random-cd'*)**

A sampling strategy that uses Latin Hypercube Sampling.

**Parameters:**

- **scramble** (*bool, optional*) – Whether to scramble the samples within sub-cubes. The default value is `True`.
- **optimization** (*str | None, optional*) –

  The optimization method to use for generating the samples. Options are:

  - `None`: No optimization is performed once the intial samples are generated.
  - `"random-cd"`: Randomly permute the columns of the matrix in order to lower the centred discrepancy of the generated samples.
  - `"lloyd"`: Perturb the samples using a modified Lloyd-Max algorithm. The process converges to equally spaced samples.

  The default is `"random-cd"`.

**__init__(*scramble=True*, *optimization='random-cd'*)**

### Methods

| | |
|---|---|
| `__init__`([scramble, optimization]) | |
| `to_json`() | |

Created using Sphinx 7.3.7.

Built with the PyData Sphinx Theme 0.15.2.

# twinlab.list_datasets

**twinlab.list_datasets(*verbose=False*)**

List datasets that have been uploaded to the user's twinLab cloud account.

These datasets can be used for training emulators and for other operations. New datasets can be uploaded using the `upload` method of the `Dataset` class.

**Parameters:**

**verbose** (*bool, optional*) – Display information about the operation while running.

**Returns:**

Datasets currently uploaded to the user's *twinLab* cloud account.

**Return type:**

list

### Example

```
datasets = tl.list_datasets()
print(datasets)
```

```
['biscuits', 'gardening', 'force-energy', 'combusion']
```

Previous
twinlab.list_emulators

Next
twinlab.list_example_datasets

# twinlab.EstimatorParams

***class*** **twinlab.EstimatorParams(*detrend=False*, *covar_module=None*, *estimator_type='single_task_gp'*)**

Parameter configuration for the Gaussian Process emulator (estimator).

**Variables:**

- **detrend** (*bool, optional*) – Should the linear trend in the data be removed (detrended) before training the emulator? The defaults is `False`.
- **covar_module** (*Union[str, None], optional*) –

  Specifies the functions that build up the kernel (covariance matrix) of the Gaussian Process. The default is `None`, which means the library will use a default kernel, which is a scaled Matern 5/2. This can be chosen from a list of possible kernels:

  - `"LIN"`: Linear.
  - `"M12"`: Matern 1/2. A standard kernel for modelling data with a smooth trend
  - `"M32"`: Matern 3/2. A standard kernel for modelling data with a smooth trend.
  - `"M52"`: Matern 5/2. A standard kernel for modelling data with a smooth trend.
  - `"PER"`: Periodic. Good for modelling data that has a periodic structure.
  - `"RBF"`: Radial Basis Function. A standard kernel for modelling data with a smooth trend. A good default choice that can model smooth functions.
  - `"RQF"`: Rational Quadratic Function.

  Kernels can also be composed by using combinations of the `"+"` (addative) and `"*"` (multiplicative) operators. For example, `covar_module = "(M52*PER)+RQF"` is valid.
- **estimator_type** (*str, optional*) –

  Specifies the type of Gaussian process to use for the emulator. The default is `"single_task_gp"`, but the value can be chosen from the following list:

  - `"single_task_gp"`: The standard Gaussian Process, which learns a mean, covariance, and noise level.
  - `"fixed_noise_gp"`: A Gaussian Process with fixed noise, which is specified by the user. Particularly useful for modelling noise-free simulated data where the noise can be set to zero manually.

Skip to main content

- `"heteroskedastic_gp"`: A Gaussian Process with fixed noise that is allowed to vary with the input. The noise is specified by the user, and is also learned by the Process.
- `"variational_gp"`: An approximate Gaussian Process that is more efficient to train with large datasets.
- `"mixed_single_task_gp"`: A Gaussian Process that works with a mix of continuous and categorical or discrete input data.
- `"multi_fidelity_gp"`: A Gaussian Process that works with input data that has multiple levels of fidelity. For example, combined data from both a high- and low-resolution simulation.
- `"fixed_noise_multi_fidelity_gp"`: A Gaussian Process that works with input data that has multiple levels of fidelity and fixed noise.

**`__init__`**(***detrend***=*False*, ***covar_module***=*None*, ***estimator_type***='*single_task_gp*')

### Methods

| | |
|---|---|
| **`__init__`**([detrend, covar_module, estimator_type]) | |
| **`unpack_parameters`**() | |

Previous
twinlab.DesignParams

Next
twinlab.ModelSelectionParams

# twinlab.Dataset.delete

**Dataset.delete(*verbose=False*)**

Delete a dataset that was previously uploaded to the twinLab cloud.

It can be useful to delete an emulator to keep a user's cloud account tidy, or if dataset has been set up incorrectly and no longer needs to be used.

**Parameters:**

**verbose** (*bool, optional*) – Display information about the operation while running.

**Return type:**

None

**Example**

```
dataset = tl.Dataset("quickstart")
dataset.delete()
```

Previous
twinlab.Dataset.analyse_output_variar

Next
Emulator

# twinlab.join_samples

**twinlab.join_samples(*df_one*, *df_two*)**

Join two dataframes that contain independent samples generated by the `Emulator.sample()` method.

The output from the `Emulator.sample()` method is a multi-indexed dataframe where the first level of the index is the parameter name and the second level is the sample number. This convenience method allows you to join two dataframes that contain independent samples generated by the `Emulator.sample()` method together into a single dataframe.

**Parameters:**

- **df_one** (*pd.DataFrame*) – The first multi-indexed dataframe to join.
- **df_two** (*pd.DataFrame*) – The second multi-indexed dataframe to join.

**Returns:**

The joined dataframe

**Return type:**

pd.DataFrame

### Example

```
df_y1 = emulator.sample(df_X, 1) # Create first set of samples
df_y2 = emulator.sample(df_X, 3) # Creates new independent samples
tl.join_samples(df_y1, df_y2)
```

| | y | | | |
|---|---|---|---|---|
| | 0 | 1 | 2 | 3 |
| 0 | 0.784193 | 1.308067 | 0.176582 | 0.875387 |
| 1 | 0.978259 | 1.039125 | 0.646922 | 0.887118 |
| 2 | 1.086855 | 0.942270 | 0.864730 | 0.934348 |

Created using Sphinx 7.3.7.

Built with the PyData Sphinx Theme 0.15.2.

# twinlab.Emulator.heatmap

**Emulator.heatmap**(*x1_axis*, *x2_axis*, *y_axis*, *x_fixed={}*, *mean_or_std='mean'*, *params=<twinlab.params.PredictParams object>*, *x1_lim=None*, *x2_lim=None*, *n_points=25*, *cmap=<matplotlib.colors.LinearSegmentedColormap object>*, *verbose=False*)

Plot a heatmap of the predictions from an emulator across two dimensions.

This will make a call to the emulator to predict across the specified dimensions. Note that a higher-than-two-dimensional emulator will be sliced across the other dimensions. The matplotlib.pyplot object is returned, and can be further modified by the user. The uncertainty of the emulator is not plotted here.

**Parameters:**

- **x1_axis** (*str*) – The name of the x1-axis variable (horizonal axis).
- **x2_axis** (*str*) – The name of the x2-axis variable (vertical axis).
- **y_axis** (*str*) – The name of the plotted variable (heatmap).
- **x_fixed** (*dict, optional*) – A dictionary of fixed values for the other `X` variables. Note that all `X` variables of an emulator must either be specified as `x1_axis`, `x2_axis` or appear as keys in `x_fixed`. Passing an empty dictionary (the default) will fix none of the variables.
- **mean_or_std** (*str, optional*) – A string determining whether to plot the mean (`"mean"`) or standard deviation (`"std"`) of the emulator. Defaults to `"mean"`.
- **params** (`PredictParams`) – (PredictParams, optional). A parameter configuration that contains optional prediction parameters.
- **x1_lim** (*tuple[float, float], optional*) – The limits of the x1-axis. If not provided, the limits will be taken directly from the emulator.
- **x2_lim** (*tuple[float, float], optional*) – The limits of the x2-axis. If not provided, the limits will be taken directly from the emulator.
- **n_points** (*int, optional*) – The number of points to sample in each dimension. The default is 25, which will create a 25x25 grid.
- **cmap** (*str, optional*) – The color of the plot. Defaults to a digiLab palette. Can be any valid matplotlib color (https://matplotlib.org/stable/users/explain/colors/colormaps.html).
- **verbose** (*bool, optional*) – Display detailed information about the operation while running.

Matplotlib plot object

**Return type:**

matplotlib.pyplot

### Examples

```
emulator = tl.Emulator("emulator_id") # A trained emulator
plt = emulator.heatmap("Latitude", "Longitude", "Rainfall", x_fixed={"Month": 6})
plt.show()
```

Previous
twinlab.Emulator.plot

Next
twinlab.Emulator.recommend

# twinlab.ScoreParams

***class* twinlab.ScoreParams(*metric='MSLL', combined_score=False*)**

Parameter configuration for scoring a trained emulator.

**Variables:**

- **metric** (*str, optional*) –

  Metric used for scoring the performance of an emulator. Can be one of:

  - `"MSLL"`: Mean Squared Log Loss, which compares the distribution of the emulator prediction to that of the test data. A score of zero is that of the most naive data-generating model, which predicts the mean and standard deviation of the training data. Lower (more negative) scores are better, while positive scores indicate serious problems.
  - `"MSE"`: Mean Squared Error, which only compares the mean emulator prediction to the test data.

  The default is `"MSLL"`.
- **combined_score** (*bool, optional*) – Determining whether to combined (average) the emulator score across output dimensions. If `False` a dataframe of scores will be returned, with the score for each output dimension, even if there is only a single emulator output dimension. If `True` a single number will be returned, which is the average score across all output dimensions. The default is `False`.

**__init__(*metric='MSLL', combined_score=False*)**

### Methods

| | |
|---|---|
| `__init__`([metric, combined_score]) | |
| `unpack_parameters`() | |

Created using Sphinx 7.3.7.

Built with the PyData Sphinx Theme 0.15.2.

# twinlab.get_sample

**twinlab.get_sample(*df*, *key*)**

Retrieve an individual sample from the multi-indexed dataframe returned by the `Emulator.sample()` method.

The output from the `Emulator.sample()` method is a multi-indexed dataframe where the first level of the index is the parameter name and the second level is the sample number. This convenience method allows you to isolate an individual sample from the dataframe by providing the sample number. The sample is returned as a standard dataframe.

**Parameters:**

- **df** (*pd.DataFrame*) – A multi-indexed dataframe returned by the Emulator.sample() method.
- **key** (*int*) – The integer key for the sample to retrieve.

**Returns:**

The individual sample as a standard (non-multi-indexed) dataframe.

**Return type:**

pd.DataFrame

### Example

```
df = emulator.sample(df_X, 5) # Generates independent samples
tl.get_sample(df, 1) # Isolate sample "1"
```

```
          y
0  1.097485
1  0.835439
2  0.655124
```

Previous
twinlab.load_example_dataset

Next
twinlab.join_samples

Skip to main content

# twinlab.Emulator.delete

**Emulator.delete(*verbose=False*)**

Delete emulator from the twinLab cloud.

It can be useful to delete an emulator to keep a cloud account tidy, or if an emulator is no longer necessary.

**Parameters:**

**verbose** (*bool, optional*) – Display detailed information about the operation while running.

**Return type:**

None

### Examples

```
emulator = tl.Emulator("quickstart")
emulator.delete()
```

Previous
twinlab.Emulator.calibrate

Next
Parameters

# twinlab.SampleParams

***class*** **twinlab.SampleParams**(***seed***=*None*, ***fidelity***=*None*)

Parameter configuration for sampling from a trained emulator.

**Variables:**

- **seed** (*Union[int, None], optional*) – Specifies the seed used by the random number generator to generate a set of samples. Setting this to an integer is useful for the reproducibility of results. The default value is `None`, which means the seed is randomly generated each time.
- **fidelity** (*Union[pandas.DataFrame, None], optional*) – Fidelity information to be provided if the model is a multi-fidelity model (`estimator_type="multi_fidelity_gp"` in `EstimatorParams`). This must be a single column *pandas.DataFrame* with the same sample order as the dataframe of X values used to draw samples. The default value is `None`, which is appropriate for most trained emulators.

**__init__**(***seed***=*None*, ***fidelity***=*None*)

### Methods

| | |
|---|---|
| `__init__`([seed, fidelity]) | |
| `unpack_parameters`() | |

Previous
twinlab.PredictParams

Next
twinlab.AcqFuncParams

# twinlab.ModelSelectionParams

***class*** **twinlab.ModelSelectionParams**(***seed***=*None*, ***evaluation_metric***='*MSLL*', ***val_ratio***=*0.2*, ***base_kernels***='*restricted*', ***depth***=*1*, ***beam***=*None*)

Parameter configuration for the Bayesian model selection process.

**Variables:**

- **seed** (*Union[int, None], optional*) – Specifies the seed for the random number generator for every trial of the model selection process. Setting to an integer is necessary for reproducible results. The default value is `None`, which means the seed is randomly generated each time.
- **evaluation_metric** (*str, optional*) –

  Specifies the evaluation metric used to score different configuration during the model selection process. Can be either:

  - `"BIC"`: Bayesian information criterion.
  - `"MSLL"`: Mean squared log loss.

  The default is `"MSLL"`.
- **val_ratio** (*float, optional*) – Specifies the percentage of random validation data allocated to to compute the `"BIC"` metric. The default is `0.2`.
- **base_kernels** (*Union[str, Set[str]], optional*) –

  Specifies the set of individual kernels to use for compositional kernel search. Can be:

  - `"all"`: The complete set of available kernels: `{"LIN", "M12", "M32", "M52", "PER", "RBF", "RQF"}`.
  - `"restricted"`: The restricted set of kernels: `{"LIN", "M32", "M52", "PER", "RBF"}`.
  - A set of strings corresponding to the individual kernels to use for kernel selection, for example `{"RBF", "PER"}`.

  The default is *“restricted”*.
- **depth** (*int, optional*) – Specifies the number of base kernels allowed to be combined in the compositional kernel search. For example, a `depth=3` search means the resulting kernel may be composed from up-to three base kernels, so examples of allowed kernel combinations would be `"(LIN+PER)*RBF"` or `"(M12*RBF)+RQF"`. The default value is `1`, which simply compares all kernel functions individually.

Skip to main content

- **beam** (*Union[int, None], optional*) – Specifies the beam width of the compositional kernel search algorithm. This uses a beam search algorithm to find the best kernel combination. This is a heuristic search algorithm that explores a graph by expanding the most promising nodes in a limited set. A `beam=1` search is exhaustive, which is algorithmically 'greedy'. `beam=None` instead performs a breadth-first search. `beam > 1` performs a beam search with the specified beam value. The default value is `None`.

**`__init__`**(***seed*=*None*, *evaluation_metric*='*MSLL*', *val_ratio*=*0.2*, *base_kernels*='*restricted*', *depth*=*1*, *beam*=*None*)

### Methods

| | |
|---|---|
| **`__init__`**([seed, evaluation_metric, ...]) | |
| **`unpack_parameters`**() | |

Previous
twinlab.EstimatorParams

Next
twinlab.TrainParams

# twinlab.RecommendParams

***class*** **twinlab.RecommendParams**(***weights***=*None*, ***num_restarts***=*5*, ***raw_samples***=*128*, ***bounds***=*None*, ***seed***=*None*)

Parameter configuration for recommending new points to sample using the Bayesian-optimisation routine.

**Variables:**

- **weights** (*Union[list[float], None], optional*) – A list of weighting values that are used to scalarise the objective function in the case of a multi-output model. The default value is `None`, which applies equal weight to each output dimension.
- **num_restarts** (*int, optional*) – The number of random restarts for optimisation. The default value is `5`.
- **raw_samples** (*int, optional*) – The number of samples for initialization. The default value is `128`.
- **bounds** (*Union[Tuple, None], optional*) – The bounds of the input space. If this is set to *None* then the bounds are inferred from the range of the training data. Otherwise, this must be a dictionary mapping column names to a tuple of lower and upper bounds. For example, `{"x0": (0, 1), "x1": (0, 2)}` to set boundaries on two input variables `x0` and `x1`.
- **seed** (*Union[int, None], optional*) – Specifies the seed used by the random number generator to start the optimiser to discover the recommendations. Setting this to an integer is good for reproducibility. The default value is `None`, which means the seed is randomly generated each time.

**__init__**(***weights***=*None*, ***num_restarts***=*5*, ***raw_samples***=*128*, ***bounds***=*None*, ***seed***=*None*)

### Methods

| | |
|---|---|
| `__init__`([weights, num_restarts, ...]) | |
| `unpack_parameters`() | |

Skip to main content

twinlab.OptimiserParams twinlab.CalibrateParams



Created using Sphinx 7.3.7.

Built with the PyData Sphinx Theme 0.15.2.

# twinlab.Emulator.benchmark

**Emulator.benchmark(*params=<twinlab.params.BenchmarkParams object>, verbose=False*)**

Benchmark the performance of a trained emulator with a calibration curve.

A test dataset must have been defined in order for this to produce a meaningful result. This means that `train_test_ratio` must be less than 1 when training the emulator. The calibration curve can be plotted to show how well the training data fits to the emulator, and is calculated differently depending on the *params* chosen. The returned dataframe contains 100 rows for each output column of the emulator. These can be plotted to ascertain the performance of the emulator.

**Parameters:**

- **params** (*BenchmarkParams, optional*) – A parameter-configuration object that contains optional parameters for benchmarking an emulator.
- **verbose** (*bool, optional*) – Display detailed information about the operation while running.

**Returns:**

Either a `pandas.DataFrame` containing the calibration curve for an emulator, or `None` if there is no test data.

**Return type:**

pandas.DataFrame, None

### Example

```
emulator = tl.Emulator("quickstart")
emulator.benchmark()
```

```
      y
0   0.0
1   0.0
2   0.0
3   0.0
4   0.0
..  ...
95  1.0
```

Skip to main content

```
98  1.0
99  1.0
```

Previous
twinlab.Emulator.score

Next
twinlab.Emulator.predict

# twinlab.Dataset.view

**Dataset.view(*verbose=False*)**

View (and download) a dataset that exists on the twinLab cloud.

**Parameters:**

**verbose** (*bool, optional*) – Display information about the operation while running.

**Return type:**

DataFrame

**Returns:**

pandas.Dataframe containing the requested dataset.

### Example

```
dataset = tl.Dataset("quickstart")
dataset.view()
```

```
          x         y
0  0.696469 -0.817374
1  0.286139  0.887656
2  0.226851  0.921553
3  0.551315 -0.326334
4  0.719469 -0.832518
5  0.423106  0.400669
6  0.980764 -0.164966
7  0.684830 -0.960764
8  0.480932  0.340115
9  0.392118  0.845795
```

Previous
twinlab.Dataset.upload

Next
twinlab.Dataset.summarise

# twinlab.Emulator.score

**Emulator.score(*params=<twinlab.params.ScoreParams object>, verbose=False*)**

Score the performance of a trained emulator.

Returns a score for a trained emulator that quantifies its performance on the test dataset. Note that a test dataset must have been defined in order for this to produce a result. This means that `train_test_ratio` in TrainParams must be less than `1` when training the emulator. If there is no test dataset then this will return `None`. The score can be calculated using different metrics, see the `ScoreParams` class for a full list and description of available metrics.

**Parameters:**

- **params** (*ScoreParams, optional*) – A parameters object that contains optional scoring parameters.
- **verbose** (*bool, optional*) – Display detailed information about the operation while running.

**Return type:**

`Union`[`float`, `DataFrame`, `None`]

**Returns:**

Either a `pandas.DataFrame` containing the emulator per output dimension (if `combined_score = False`), or a `float` containing the combined score of the emulator averaged acorss output dimensions (if `combined_score = True`), or `None` if there was no test data defined during training.

### Examples

Request the mean-standarised log loss (MSLL) averaged (combined) across all emulator output dimensions:

```
emulator = tl.Emulator("my_emulator")
params = tl.ScoreParams(metric="MSLL", combined_score=True)
emulator.score(params=params)
```

```
-4.07
```

Request the mean-squared error (MSE) for each output individually:

```
emulator = tl.Emulator("my_emulator")
params = tl.ScoreParams(metric="MSE", combined_score=False)
emulator.score(params=params)
```

```
pd.DataFrame({'y1': [1.8], 'y2': [0.9]})
```

Previous
twinlab.Emulator.summarise

Next
twinlab.Emulator.benchmark

# twinlab.Emulator.predict

**`Emulator.predict(df, params=<twinlab.params.PredictParams object>, wait=True, verbose=False)`**

Make predictions using a trained emulator that exists on the twinLab cloud.

This method makes predictions from a trained emulator on new data. This method is the workhorse of the twinLab suite, allowing users to make predictions based on their training data. The emulator can make predictions on data that are far away from the training data, and can interpolate reliably between the training data points. The emulator returns both a predicted mean and standard deviation for each output dimension. This allows a user to not only make predictions, but also to quantify the uncertainty on those predictions. For a Gaussian Process, the standard deviation is a measure of the uncertainty in the prediction, while the mean is the prediction itself. The emulator is 95% confident that the true value lies within two standard deviations of the mean.

**Parameters:**

- **df** (*pandas.DataFrame*) – The `X` values for which to make predictions.
- **params** (*PredictParams*) – A parameter-configuration that contains optional parameters for making predictions.
- **wait** (*bool, optional*) – If `True` wait for the job to complete, otherwise return the process ID and exit.
- **verbose** (*bool, optional*) – Display detailed information about the operation while running.

**Returns:**

By default a tuple containing the mean and standard deviation of the emulator prediction. Instead, if `wait=False`, the process ID is returned. The results can then be retrieved later using `Emulator.get_process(<process_id>)`. Process IDs associated with an emulator can be found using `Emulator.list_processes()`.

**Return type:**

Tuple[pandas.DataFrame, pandas.DataFrame], str

### Example

Skip to main content

```
df_mean, df_std = emulator.predict(df)
```

```
          y
0  0.845942
1  0.922921
2  0.846308
3  0.570473

          y
0  0.404200
1  0.180853
2  0.146619
3  0.147886
```

Previous
twinlab.Emulator.benchmark

Next
twinlab.Emulator.sample

# twinlab.Emulator.view

**Emulator.view(*verbose=False*)**

View an emulator that exists on the twinLab cloud.

This returns the parameter configuration of the emulator that is stored on the twinLab cloud. This allows a user to check the parameters that were used to train an emulator.

**Parameters:**

**verbose** (*bool, optional*) – Display information about the operation while running.

**Return type:**

dict

### Example

```
emulator = tl.Emulator("quickstart")
emulator.view()
```

```
{'dataset_id': 'quickstart',
 'decompose_inputs': False,
 'decompose_outputs': False,
 'estimator': 'gaussian_process_regression',
 'estimator_kwargs': {'detrend': False, 'estimator_type': 'single_task_gp'},
 'inputs': ['x'],
 'modal_handle': 'fc-6L9EsWZhOkc8xyHguPphh6',
 'model_id': 'quickstart',
 'model_selection': False,
 'model_selection_kwargs': {'base_kernels': 'restricted',
                            'depth': 1,
                            'evaluation_metric': 'MSLL',
                            'val_ratio': 0.2},
 'outputs': ['y'],
 'train_test_ratio': 0.8}
```

Created using Sphinx 7.3.7.

Built with the PyData Sphinx Theme 0.15.2.

# twinlab.TrainParams

***class* twinlab.TrainParams(*estimator='gaussian_process_regression', estimator_params=<twinlab.params.EstimatorParams object>, input_explained_variance=None, input_retained_dimensions=None, output_explained_variance=None, output_retained_dimensions=None, fidelity=None, dataset_std=None, train_test_ratio=1.0, model_selection=False, model_selection_params=<twinlab.params.ModelSelectionParams object>, seed=None*)**

Parameter configuration for training an emulator.

This includes parameters that pertain directly to the training of the model, such as the ratio of training to testing data, as well as parameters that pertain to the setup of the model such as the number of dimensions to retain after decomposition.

**Variables:**

- **estimator** (*str, optional*) – The type of estimator (emulator) to be trained. Currently only “gaussian_process_regression” is supported, which is the default value.
- **estimator_params** (*EstimatorParams, optional*) – The parameters for the Gaussian Process emulator.
- **input_retained_dimensions** (*Union[int, None], optional*) – The number of input dimensions to retain after applying dimensional reduction. Setting this to an integer cannot be done at the same time as specifying the `input_explained_variance`. The default value is `None`, which means that dimensional reduction is not applied to the input unless `input_explained_variance` is specified.
- **input_explained_variance** (*Union[float, None], optional*) – Specifies what fraction of the variance of the input data is retained after applying dimensional reduction. This must be a number between 0 and 1. This cannot be specified at the same time as `input_retained_dimensions`. The default value is `None`, which means that dimensional reduction is not applied to the input unless `input_retained_dimensions` is specified.
- **output_retained_dimensions** (*Union[int, None], optional*) – The number of output dimensions to retain after applying dimensional reduction. Setting this to an integer cannot be done at the same time as specifying the `output_explained_variance`. The default value is `None`, which means that dimensional reduction is not applied to the output unless `output_explained_variance` is

- **output_explained_variance** (*Union[float, None], optional*) – Specifies what fraction of the variance of the output data is retained after applying dimensional reduction. This must be a number between 0 and 1. This cannot be specified at the same time as `output_retained_dimensions`. The default value is `None`, which means that dimensional reduction is not applied to the output unless `output_retained_dimensions` is specified.
- **fidelity** (*Union[str, None], optional*) – Name of the column in the dataset corresponding to the fidelity parameter if a multi-fidelity model is being trained. The default value is `None`, whereby fidelity information is provided. Fidelity refers to the degree an emulator is able to reproduce the behaviour of the simulated data.
- **train_test_ratio** (*Union[float, None], optional*) – Specifies the fraction of training samples in the dataset. This must be a number beteen 0 and 1. The default value is 1, which means that all of the provided data is used for training. This is good to make the most out of a dataset, but means that it will not be possible to score or benchmark the performance of an emulator.
- **dataset_std** (*Union[Dataset, None], optional*) – A twinLab dataset object that contains the standard deviation of the training data. This is necessary when training a heteroskedastic or fixed noise Gaussian Process.
- **model_selection** (*bool, optional*) – Whether to run Bayesian model selection, a form of automatic machine learning. The default value is `False`, which simply trains the specified emulator, rather than iterating over them.
- **model_selection_params** (*ModelSelectionParams, optional*) – The parameters for model selection, if it is being used.
- **seed** (*Union[int, None], optional*) – The seed used to initialise the random number generators for reproducibility. Setting to an integer is necessary for reproducible results. The default value is `None`, which means the seed is randomly generated each time.

**`__init__`(*estimator='gaussian_process_regression', estimator_params=<twinlab.params.EstimatorParams object>, input_explained_variance=None, input_retained_dimensions=None, output_explained_variance=None, output_retained_dimensions=None, fidelity=None, dataset_std=None, train_test_ratio=1.0, model_selection=False, model_selection_params=<twinlab.params.ModelSelectionParams object>, seed=None*)**

### Methods

| | |
|---|---|
| `__init__`([estimator, estimator_params, ...]) | |

Skip to main content

Previous
**twinlab.ModelSelectionParams**

Next
**twinlab.ScoreParams**



Created using Sphinx 7.3.7.

Built with the PyData Sphinx Theme 0.15.2.

# twinlab.AcqFuncParams

*class* twinlab.**AcqFuncParams**(**args*, ***kwargs*)

**__init__**()

### Methods

| |
|---|
| __init__() |
| unpack_parameters() |

Previous
twinlab.SampleParams

Next
twinlab.OptimiserParams

# twinlab.versions

**twinlab.versions**(*verbose=False*)

Get information about the twinLab version being used.

**Parameters:**

**verbose** (*bool, optional*) – Display information about the operation while running.

**Returns:**

Version information.

**Return type:**

dict

### Example

```
version_info = tl.versions()
print(version_info)
```

```
{'cloud': '2.3.0', 'modal': '0.4.0', 'library': '1.7.0', 'image': 'twinlab'}
```

Previous
twinlab.get_server_url

Next
twinlab.user_information

# twinlab.Prior

***class*** **twinlab.Prior(*name*, *distribution*)**

A prior probability distribution

**Variables:**

- **name** (*str*) – This is the name given to the prior, usually corresponding to the parameter it represents.
- **distribution** (*Distribution*) – The one-dimensional probability distribution for the prior.

**__init__(*name*, *distribution*)**

### Methods

| | |
|---|---|
| `__init__`(name, distribution) | |
| `to_json`() | |

Previous
Distributions

Next
twinlab.distributions

# twinlab.Emulator.calibrate

**Emulator.calibrate(*df_obs, df_std, params=<twinlab.params.CalibrateParams object>, wait=True, verbose=False*)**

Solve an inverse problem using a trained emulator on the twinLab cloud.

A classic trained emulator can ingest `X` values and use these to predict corresponding `y` values. However, the emulator can also be used to solve an inverse problem, where the user has an observation of `y` and wants to find the corresponding `X`. Problems of this type are common in engineering and science, where the user has an observation of a system and wants to find the parameters that generated that observation. This operation can be numerically intensive, and the emulator can be used to solve this problem quickly and efficiently.

**Parameters:**

- **df_obs** (*pandas.DataFrame*) – A dataframe containing the single observation.
- **df_std** (*pandas.DataFrame*) – A dataframe containing the error on the single observation.
- **params** (*CalibrateParams, optional*) – A parameter configuration that contains all optional calibration parameters.
- **wait** (*bool, optional*) – If `True` wait for the job to complete, otherwise return the process ID and exit.
- **verbose** (*bool, optional*) – Display detailed information about the operation while running.

**Returns:**

By default, the solution to the inverse problem is either presented as a summary, or as the full set of points sampled from the posterior distribution. See the documentation for `CalibrateParams` for more information on the different options. Instead, if `wait=False`, the process ID is returned. The results can then be retrieved later using `Emulator.get_process(<process_id>)`. Process IDs associated with an emulator can be found using `Emulator.list_processes()`.

**Return type:**

pandas.DataFrame, str

## Example

Skip to main content

```
df_std = pd.DataFrame({'y': [0.01]})
emulator.calibrate(df_obs, df_std)
```

| | mean | sd | hdi_3% | hdi_97% | mcse_mean | mcse_sd | ess_bulk | ess_tail | r_hat |
|---|---|---|---|---|---|---|---|---|---|
| x | 0.496 | 0.013 | 0.471 | 0.521 | 0.0 | 0.0 | 2025.0 | 2538.0 | 1.0 |

Previous
twinlab.Emulator.learn

Next
twinlab.Emulator.delete

# twinlab.get_server_url

**twinlab.get_server_url(*verbose=False*)**

Show the URL from which twinLab is currently being accessed.

**Parameters:**

**verbose** (*bool, optional*) – Display information about the operation while running.

**Returns:**

Server URL.

**Return type:**

str

### Example

```
tl.get_server_url()
```

```
'https://twinlab.digilab.co.uk'
```

Previous
twinlab.set_server_url

Next
twinlab.versions

# twinlab.Emulator.sample

**Emulator.sample(*df*, *num_samples*, *params=<twinlab.params.SampleParams object>*, *wait=True*, *verbose=False*)**

Draw samples from a trained emulator that exists on the twinLab cloud.

A secondary functionality of the emulator is to draw sample predictions from the trained emulator. Rather than quantifying the uncertainty in the predictions, this method draws samples from the emulator. The collection of samples can be used to explore the distribution of the emulator predictions. Each sample is a possible prediction of the emulator, and therefore a prediction of a possible new observation from the data-generation process. The covariance in the emulator predictions can therefore be explored, which is particularly useful for functional Gaussian Processes.

If the output of the multi-indexed DataFrame needs to be manipulated then we provide the convenience functions:

> - `tl.get_sample`: Isolate an individual sample into a new `pandas.DataFrame`
> - `tl.join_samples`: Join together multiple sets of samples into a single `pandas.DataFrame`

**Parameters:**

- **df** (*pandas.DataFrame*) – The `X` values for which to draw samples.
- **num_samples** (*int*) – Number of samples to draw for each row of the evaluation data.
- **params** (*SampleParams, optional*) – A *SampleParams* object with sampling parameters.
- **wait** (*bool, optional*) – If `True` wait for the job to complete, otherwise return the process ID and exit.
- **verbose** (*bool, optional*) – Display detailed information about the operation while running.

**Returns:**

By default a multi-indexed DataFrame containing the `y` samples drawn from the emulator. Instead, if `wait=False` the process ID is returned. The results can then be retrieved later using `Emulator.get_process(<process_id>)`. Process IDs associated with an emulator can be found using `Emulator.list_processes()`.

pandas.DataFrame, str

### Example

```
emulator = tl.Emulator("quickstart")
df = pd.DataFrame({'x': [0.1, 0.2, 0.3, 0.4]})
emulator.sample(df, 3)
```

```
          y
          0         1         2
0  0.490081  1.336099  0.608441
1  0.829179  1.038671  0.807405
2  0.805102  0.773975  0.984713
3  0.605568  0.416630  0.713652
```

Previous
twinlab.Emulator.predict

Next
twinlab.Emulator.plot

# twinlab.CalibrateParams

***class*** **twinlab.CalibrateParams**(***y_std_model***=*False*, ***return_summary***=*True*, ***iterations***=*10000*, ***n_chains***=*2*, ***force_sequential***=*False*, ***seed***=*None*)

Parameter configuration for inverting a trained emulator to estimate the input parameters that generated a given output.

**Variables:**

- **y_std_model** (*Union[bool, pd.DataFrame], optional*) –

  Whether to include model noise covariance in the likelihood.

  - If `True` TODO …
  - If `False` TODO …
  - If a *pandas.DataFrame* is supplied, it must contain the same columns as the set of emulator outputs.

  The default value is `False`.
- **return_summary** (*bool, optional*) –

  Should the result of the inversion be presented as a summary or as the full solution?

  - If `True` then return a summary of the inverse solution.
  - If `False` return the entire solution in the form of the points sampled.

  The default value is `True`.
- **iterations** (*int, optional*) – The number of points to sample in each inversion chain. More points is better. The default value is `10,000`.
- **n_chains** (*int, optional*) – The number of independent chains to use for the inversion process. More is better, so that the solution derived between indepent chains can be compared and convergence can be checked. The default value is `2`.
- **force_sequential** (*bool, optional*) – TODO The default value is False.
- **seed** (*Union[int, None], optional*) – Specifies the seed used by the random number generator to start the inversion process. Setting the seed to an integer is good for reproducibility. The default value is `None`, which means the seed is randomly generated each time.

**__init__**(***y_std_model***=*False*, ***return_summary***=*True*, ***iterations***=*10000*,

### Methods

| | |
|---|---|
| `__init__` | ([y_std_model, return_summary, ...]) |
| `unpack_parameters` | () |

Previous
twinlab.RecommendParams

Next
Distributions

# twinlab.Emulator.status

**Emulator.status(*process_id*, *verbose=False*)**

Check the status of a training process on the twinLab cloud.

**Parameters:**

- **process_id** (*str*) – The process ID of the training process to check the status of.
- **verbose** (*bool, optional*) – Display information about the operation while running.

**Returns:**

A tuple containing the status code and the response body.

**Return type:**

Tuple[int, dict]

### Example

```
emulator = tl.Emulator("beb7f97f")
emulator.status()
```

```
{
    'process_status': 'Your job has finished and is on its way back to you.',
    'process_id': 'beb7f97f',
}
```

Previous
twinlab.Emulator.train

Next
twinlab.Emulator.view

# twinlab.BenchmarkParams

***class*** **twinlab.BenchmarkParams(*type*='*quantile*')**

Parameter configuration for benchmarking a trained emulator.

**Variables:**

**type** (*str, optional*) –

Specifies the type of emulator benchmark to be performed. Can be either:

- `"quantile"`: The calibration curve is calculated over Gaussian quantiles.
- `"interval"`: The calibration curve is calculated over Gaussian confidence intervals.

The default is `"quantile"`.

**__init__(*type*='*quantile*')**

### Methods

| | |
|---|---|
| `__init__`([type]) | |
| `unpack_parameters`() | |

Previous
twinlab.ScoreParams

Next
twinlab.PredictParams

# twinlab.Emulator.view_test_data

**Emulator.view_test_data(*verbose=False*)**

View test data on which the emulator was tested in the twinLab cloud.

**Parameters:**

**verbose** (*bool, optional*) – Display information about the operation while running.

**Returns:**

Dataframe containing the training data on which the emulator was tested

**Return type:**

pandas.DataFrame

### Example

```
emulator = tl.Emulator("quickstart")
emulator.view_test_data()
```

```
          x         y
0  0.480932  0.340115
1  0.392118  0.845795
```

Previous
twinlab.Emulator.view_train_data

Next
twinlab.Emulator.summarise