

---

# **GPyOpt Documentation**

**GPyOpt Authors**

**Mar 27, 2018**



---

## Contents:

---

<b>1</b>	<b>GPyOpt.acquisitions package</b>	<b>1</b>
1.1	Submodules	1
1.2	GPyOpt.acquisitions.EI module	1
1.3	GPyOpt.acquisitions.EI_mcmc module	2
1.4	GPyOpt.acquisitions.LCB module	2
1.5	GPyOpt.acquisitions.LCB_mcmc module	3
1.6	GPyOpt.acquisitions.LP module	3
1.7	GPyOpt.acquisitions.MPI module	4
1.8	GPyOpt.acquisitions.MPI_mcmc module	4
1.9	GPyOpt.acquisitions.base module	5
1.10	Module contents	5
<b>2</b>	<b>GPyOpt.core package</b>	<b>7</b>
2.1	Subpackages	7
2.2	Submodules	14
2.3	GPyOpt.core.bo module	14
2.4	GPyOpt.core.errors module	16
2.5	Module contents	16
<b>3</b>	<b>GPyOpt.experiment_design package</b>	<b>17</b>
3.1	Submodules	17
3.2	GPyOpt.experiment_design.base module	17
3.3	GPyOpt.experiment_design.grid_design module	17
3.4	GPyOpt.experiment_design.latin_design module	18
3.5	GPyOpt.experiment_design.random_design module	18
3.6	GPyOpt.experiment_design.sobol_design module	18
3.7	Module contents	18
<b>4</b>	<b>GPyOpt.interface package</b>	<b>19</b>
4.1	Submodules	19
4.2	GPyOpt.interface.config_parser module	19
4.3	GPyOpt.interface.driver module	19
4.4	GPyOpt.interface.func_loader module	19
4.5	GPyOpt.interface.output module	20
4.6	Module contents	20
<b>5</b>	<b>GPyOpt.methods package</b>	<b>21</b>

5.1	Submodules	22
5.2	GPyOpt.methods.bayesian_optimization module	22
5.3	GPyOpt.methods.modular_bayesian_optimization module	24
5.4	Module contents	25
<b>6</b>	<b>GPyOpt.models package</b>	<b>27</b>
6.1	Submodules	27
6.2	GPyOpt.models.base module	27
6.3	GPyOpt.models.gpmodel module	27
6.4	GPyOpt.models.input_warped_gpmodel module	30
6.5	GPyOpt.models.rfmodel module	30
6.6	GPyOpt.models.warpedgpmodel module	31
6.7	Module contents	31
<b>7</b>	<b>GPyOpt.objective_examples package</b>	<b>33</b>
7.1	Submodules	33
7.2	GPyOpt.objective_examples.experiments1d module	33
7.3	GPyOpt.objective_examples.experiments2d module	33
7.4	GPyOpt.objective_examples.experimentsNd module	35
7.5	Module contents	36
<b>8</b>	<b>GPyOpt.optimization package</b>	<b>37</b>
8.1	Submodules	37
8.2	GPyOpt.optimization.acquisition_optimizer module	37
8.3	GPyOpt.optimization.anchor_points_generator module	38
8.4	GPyOpt.optimization.optimizer module	38
8.5	Module contents	40
<b>9</b>	<b>GPyOpt.plotting package</b>	<b>41</b>
9.1	Submodules	41
9.2	GPyOpt.plotting.plots_bo module	41
9.3	Module contents	41
<b>10</b>	<b>GPyOpt.util package</b>	<b>43</b>
10.1	Submodules	43
10.2	GPyOpt.util.arguments_manager module	43
10.3	GPyOpt.util.duplicate_manager module	43
10.4	GPyOpt.util.general module	44
10.5	GPyOpt.util.io module	45
10.6	GPyOpt.util.stats module	45
10.7	Module contents	45
<b>11</b>	<b>Indices and tables</b>	<b>47</b>
	<b>Python Module Index</b>	<b>49</b>

## 1.1 Submodules

## 1.2 GPyOpt.acquisitions.EI module

```
class GPyOpt.acquisitions.EI.AcquisitionEI (model, space, optimizer=None,  
                                           cost_withGradients=None, jitter=0.01)
```

Bases: *GPyOpt.acquisitions.base.AcquisitionBase*

Expected improvement acquisition function

### Parameters

- **model** – GPyOpt class of model
- **space** – GPyOpt class of domain
- **optimizer** – optimizer of the acquisition. Should be a GPyOpt optimizer
- **cost\_withGradients** – function
- **jitter** – positive value to make the acquisition more explorative.

---

**Note:** allows to compute the Improvement per unit of cost

---

```
analytical_gradient_prediction = True
```

```
static fromConfig (space, optimizer, cost_withGradients, config)
```

## 1.3 GPyOpt.acquisitions.EI\_mcmc module

**class** GPyOpt.acquisitions.EI\_mcmc.**AcquisitionEI\_MCMC** (*model, space, optimizer=None, cost\_withGradients=None, jitter=0.01*)

Bases: *GPyOpt.acquisitions.EI.AcquisitionEI*

Integrated Expected improvement acquisition function

### Parameters

- **model** – GPyOpt class of model
- **space** – GPyOpt class of domain
- **optimizer** – optimizer of the acquisition. Should be a GPyOpt optimizer
- **cost\_withGradients** – function
- **jitter** – positive value to make the acquisition more explorative

---

**Note:** allows to compute the Improvement per unit of cost

---

**analytical\_gradient\_prediction = True**

## 1.4 GPyOpt.acquisitions.LCB module

**class** GPyOpt.acquisitions.LCB.**AcquisitionLCB** (*model, space, optimizer=None, cost\_withGradients=None, exploration\_weight=2*)

Bases: *GPyOpt.acquisitions.base.AcquisitionBase*

GP-Lower Confidence Bound acquisition function

### Parameters

- **model** – GPyOpt class of model
- **space** – GPyOpt class of domain
- **optimizer** – optimizer of the acquisition. Should be a GPyOpt optimizer
- **cost\_withGradients** – function
- **jitter** – positive value to make the acquisition more explorative

---

**Note:** does not allow to be used with cost

---

**analytical\_gradient\_prediction = True**

## 1.5 GPyOpt.acquisitions.LCB\_mcmc module

**class** GPyOpt.acquisitions.LCB\_mcmc.**AcquisitionLCB\_MCMC** (*model*, *space*, *optimizer=None*, *cost\_withGradients=None*, *exploration\_weight=2*)

Bases: *GPyOpt.acquisitions.LCB.AcquisitionLCB*

Integrated GP-Lower Confidence Bound acquisition function

### Parameters

- **model** – GPyOpt class of model
- **space** – GPyOpt class of domain
- **optimizer** – optimizer of the acquisition. Should be a GPyOpt optimizer
- **cost\_withGradients** – function
- **exploration\_weight** – positive parameter to control exploration / exploitation

---

**Note:** allows to compute the Improvement per unit of cost

---

**analytical\_gradient\_prediction = True**

## 1.6 GPyOpt.acquisitions.LP module

**class** GPyOpt.acquisitions.LP.**AcquisitionLP** (*model*, *space*, *optimizer*, *acquisition*, *transform='none'*)

Bases: *GPyOpt.acquisitions.base.AcquisitionBase*

Class for Local Penalization acquisition. Used for batch design. :param model: model of the class GPyOpt :param space: design space of the class GPyOpt. :param optimizer: optimizer of the class GPyOpt. :param acquisition: acquisition function of the class GPyOpt :param transform: transformation applied to the acquisition (default, none).

---

**Note:** irrespective of the transformation applied the penalized acquisition is always mapped again to the log space.

---

This way gradients can be computed additively and are more stable.

**acquisition\_function** (*x*)

Returns the value of the acquisition function at x.

**acquisition\_function\_withGradients** (*x*)

Returns the acquisition function and its gradient at x.

**analytical\_gradient\_prediction = True**

**d\_acquisition\_function** (*x*)

Returns the gradient of the acquisition function at x.

**update\_batches** (*X\_batch*, *L*, *Min*)

Updates the batches internally and pre-computes the

## 1.7 GPyOpt.acquisitions.MPI module

```
class GPyOpt.acquisitions.MPI.AcquisitionMPI (model, space, optimizer=None,  
                                              cost_withGradients=None, jitter=0.01)
```

Bases: *GPyOpt.acquisitions.base.AcquisitionBase*

Maximum probability of improvement acquisition function

### Parameters

- **model** – GPyOpt class of model
- **space** – GPyOpt class of domain
- **optimizer** – optimizer of the acquisition. Should be a GPyOpt optimizer
- **cost\_withGradients** – function
- **jitter** – positive value to make the acquisition more explorative

---

**Note:** allows to compute the Improvement per unit of cost

---

```
analytical_gradient_prediction = True
```

```
static fromConfig (space, optimizer, cost_withGradients, config)
```

## 1.8 GPyOpt.acquisitions.MPI\_mcmc module

```
class GPyOpt.acquisitions.MPI_mcmc.AcquisitionMPI_MCMC (model, space, op-  
                                                         imizer=None,  
                                                         cost_withGradients=None,  
                                                         jitter=0.01)
```

Bases: *GPyOpt.acquisitions.MPI.AcquisitionMPI*

Integrated Maximum Probability of Improvement acquisition function

### Parameters

- **model** – GPyOpt class of model
- **space** – GPyOpt class of domain
- **optimizer** – optimizer of the acquisition. Should be a GPyOpt optimizer
- **cost\_withGradients** – function
- **jitter** – positive value to make the acquisition more explorative

---

**Note:** allows to compute the Improvement per unit of cost

---

```
analytical_gradient_prediction = True
```



## 1.9 GPyOpt.acquisitions.base module

**class** GPyOpt.acquisitions.base.**AcquisitionBase** (*model*, *space*, *optimizer*,  
*cost\_withGradients=None*)

Bases: object

Base class for acquisition functions in Bayesian Optimization

### Parameters

- **model** – GPyOpt class of model
- **space** – GPyOpt class of domain
- **optimizer** – optimizer of the acquisition. Should be a GPyOpt optimizer

**acquisition\_function** (*x*)

Takes an acquisition and weights it so the domain and cost are taken into account.

**acquisition\_function\_withGradients** (*x*)

Takes an acquisition and its gradient and weights it so the domain and cost are taken into account.

**analytical\_gradient\_prediction = False**

**static fromDict** (*space*, *optimizer*, *cost\_withGradients*, *config*)

**optimize** (*duplicate\_manager=None*)

Optimizes the acquisition function (uses a flag from the model to use gradients or not).

## 1.10 Module contents

GPyOpt.acquisitions.**select\_acquisition** (*name*)

Acquisition selector



## 2.1 Subpackages

### 2.1.1 GPyOpt.core.evaluators package

#### Submodules

##### GPyOpt.core.evaluators.base module

**class** GPyOpt.core.evaluators.base.**EvaluatorBase** (*acquisition, batch\_size, \*\*kwargs*)  
Bases: object

Base class for the evaluator of the function. This class handles both sequential and batch evaluators.

**compute\_batch** (*duplicate\_manager=None, context\_manager=None*)

**class** GPyOpt.core.evaluators.base.**SamplingBasedBatchEvaluator** (*acquisition, batch\_size, \*\*kwargs*)  
Bases: *GPyOpt.core.evaluators.base.EvaluatorBase*

This class handles specific types of batch evaluators, based on the sampling of anchor points (examples are random and Thompson sampling).

**compute\_batch** (*duplicate\_manager=None, context\_manager=None*)

**compute\_batch\_without\_duplicate\_logic** (*context\_manager=None*)

**get\_anchor\_points** (*duplicate\_manager=None, context\_manager=None*)

**initialize\_batch** (*duplicate\_manager=None, context\_manager=None*)

**optimize\_anchor\_point** (*a, duplicate\_manager=None, context\_manager=None*)

**zip\_and\_tuple** (*x*)

convenient helper :param x: input configuration in the model space :return: zipped x as a tuple

### GPyOpt.core.evaluators.batch\_local\_penalization module

**class** GPyOpt.core.evaluators.batch\_local\_penalization.**LocalPenalization** (*acquisition*,  
*batch\_size*)

Bases: *GPyOpt.core.evaluators.base.EvaluatorBase*

Class for the batch method on ‘Batch Bayesian optimization via local penalization’ (Gonzalez et al., 2016).

#### Parameters

- **acquisition** – acquisition function to be used to compute the batch.
- **size** (*batch*) – the number of elements in the batch.

**compute\_batch** (*duplicate\_manager=None*, *context\_manager=None*)

Computes the elements of the batch sequentially by penalizing the acquisition.

GPyOpt.core.evaluators.batch\_local\_penalization.**estimate\_L** (*model*, *bounds*, *store-*  
*history=True*)

Estimate the Lipschitz constant of  $f$  by taking maximizing the norm of the expectation of the gradient of  $f$ .

### GPyOpt.core.evaluators.batch\_random module

**class** GPyOpt.core.evaluators.batch\_random.**RandomBatch** (*acquisition*, *batch\_size*)

Bases: *GPyOpt.core.evaluators.base.SamplingBasedBatchEvaluator*

Class for a random batch method. The first element of the batch is selected by optimizing the acquisition in a standard way. The remaining elements are selected uniformly random in the domain of the objective.

#### Parameters

- **acquisition** – acquisition function to be used to compute the batch.
- **size** (*batch*) – the number of elements in the batch.

**compute\_batch\_without\_duplicate\_logic** (*context\_manager=None*)

**get\_anchor\_points** (*duplicate\_manager=None*, *context\_manager=None*)

**initialize\_batch** (*duplicate\_manager=None*, *context\_manager=None*)

**optimize\_anchor\_point** (*a*, *duplicate\_manager=None*, *context\_manager=None*)

### GPyOpt.core.evaluators.batch\_thompson module

**class** GPyOpt.core.evaluators.batch\_thompson.**ThompsonBatch** (*acquisition*,  
*batch\_size*)

Bases: *GPyOpt.core.evaluators.base.SamplingBasedBatchEvaluator*

Class for a Thompson batch method. Elements are selected iteratively using the current acquisition function but exploring the models by using Thompson sampling

#### Parameters

- **acquisition** – acquisition function to be used to compute the batch.
- **size** (*batch*) – the number of elements in the batch.

**compute\_batch\_without\_duplicate\_logic** (*context\_manager=None*)

**get\_anchor\_points** (*duplicate\_manager=None*, *context\_manager=None*)

**initialize\_batch** (*duplicate\_manager=None*, *context\_manager=None*)

`optimize_anchor_point` (*a*, *duplicate\_manager=None*, *context\_manager=None*)

## GPyOpt.core.evaluators.sequential module

**class** GPyOpt.core.evaluators.sequential.**Sequential** (*acquisition*, *batch\_size=1*)

Bases: *GPyOpt.core.evaluators.base.EvaluatorBase*

Class for standard Sequential Bayesian optimization methods.

### Parameters

- **acquisition** – acquisition function to be used to compute the batch.
- **size** (*batch*) – it is 1 by default since this class is only used for sequential methods.

**compute\_batch** (*duplicate\_manager=None*, *context\_manager=None*)

Selects the new location to evaluate the objective.

## Module contents

GPyOpt.core.evaluators.**select\_evaluator** (*name*)

## 2.1.2 GPyOpt.core.task package

### Submodules

#### GPyOpt.core.task.cost module

**class** GPyOpt.core.task.cost.**CostModel** (*cost\_withGradients*)

Bases: *object*

Class to handle the cost of evaluating the function.

param *cost\_withGradients*: function that returns the cost of evaluating the function and its gradient. By default no cost is used. Options are:

- *cost\_withGradients* is some pre-defined cost function. Should return numpy array as outputs.
- *cost\_withGradients* = 'evaluation\_time'.

---

**Note:** if *cost\_withGradients* = 'evaluation time' the evaluation time of the function is used to model a GP whose

---

mean is used as cost.

**update\_cost\_model** (*x*, *cost\_x*)

Updates the GP used to handle the cost.

param *x*: input of the GP for the cost model. param *x\_cost*: values of the time cost at the input locations.

GPyOpt.core.task.cost.**constant\_cost\_withGradients** (*x*)

Constant cost function used by default: *cost*=1, *d\_cost*=0.

## GPyOpt.core.task.objective module

**class** GPyOpt.core.task.objective.Objective

Bases: object

General class to handle the objective function internally.

**evaluate** (*x*)

**class** GPyOpt.core.task.objective.SingleObjective (*func*, *num\_cores=1*, *ob-*  
*jective\_name='no\_name'*,  
*batch\_type='synchronous'*,  
*space=None*)

Bases: *GPyOpt.core.task.objective.Objective*

Class to handle problems with one single objective function.

param *func*: objective function. param *batch\_size*: size of the batches (default, 1) param *num\_cores*: number of cores to use in the process of evaluating the objective (default, 1). param *objective\_name*: name of the objective function. param *batch\_type*: Type of batch used. Only 'synchronous' evaluations are possible at the moment. param *space*: Not in use.

---

**Note:** the objective function should take 2-dimensional numpy arrays as input and outputs. Each row should

---

contain a location (in the case of the inputs) or a function evaluation (in the case of the outputs).

**evaluate** (*x*)

Performs the evaluation of the objective at *x*.

## GPyOpt.core.task.space module

**class** GPyOpt.core.task.space.Design\_space (*space*, *constraints=None*,  
*store\_noncontinuous=False*)

Bases: object

Class to handle the input domain of the function. The format of a input domain, possibly with restrictions: The domain is defined as a list of dictionaries contains a list of attributes, e.g.:

- Arm bandit

*space*=[{'name': 'var\_1', 'type': 'bandit', 'domain': [(-1,1),(1,0),(0,1)]}, {'name': 'var\_2', 'type': 'bandit', 'domain': [(-1,4),(0,0),(1,2)]}]

- Continuous domain

*space*=[ {'name': 'var\_1', 'type': 'continuous', 'domain':(-1,1), 'dimensionality':1}, {'name': 'var\_2', 'type': 'continuous', 'domain':(-3,1), 'dimensionality':2}, {'name': 'var\_3', 'type': 'bandit', 'domain': [(-1,1),(1,0),(0,1)], 'dimensionality':2}, {'name': 'var\_4', 'type': 'bandit', 'domain': [(-1,4),(0,0),(1,2)]}, {'name': 'var\_5', 'type': 'discrete', 'domain': (0,1,2,3)}]

- Discrete domain

*space*=[ {'name': 'var\_3', 'type': 'discrete', 'domain': (0,1,2,3)}] {'name': 'var\_3', 'type': 'discrete', 'domain': (-10,10)}

- Mixed domain

```
space=[{'name': 'var_1', 'type': 'continuous', 'domain':(-1,1), 'dimensionality':1}, {'name': 'var_4',
      'type': 'continuous', 'domain':(-3,1), 'dimensionality':2}, {'name': 'var_3', 'type': 'discrete', 'domain':
      (0,1,2,3)}]
```

Restrictions can be added to the problem. Each restriction is of the form  $c(x) \leq 0$  where  $c(x)$  is a function of the input variables previously defined in the space. Restrictions should be written as a list of dictionaries. For instance, this is an example of an space coupled with a constraint

```
space=[ {'name': 'var_1', 'type': 'continuous', 'domain':(-1,1), 'dimensionality':2}] constraints = [ {'name':
'const_1', 'constraint': 'x[:,0]**2 + x[:,1]**2 - 1' }]
```

If no constraints are provided the hypercube determined by the bounds constraints are used.

Note about the internal representation of the variables: for variables in which the dimensionality has been specified in the domain, a subindex is internally assigned. For instance if the variable is called 'var1' and has dimensionality 3, the first three positions in the internal representation of the domain will be occupied by variables 'var1\_1', 'var1\_2' and 'var1\_3'. If no dimensionality is added, the internal naming remains the same. For instance, in the example above 'var3' should be fixed its original name.

param space: list of dictionaries as indicated above. param constraints: list of dictionaries as indicated above (default, none)

**find\_variable** (*variable\_name*)

**static fromConfig** (*constraints*)

**get\_bandit** ()

Extracts the arms of the bandit if any.

**get\_bounds** ()

Extracts the bounds of all the inputs of the domain of the *model*

**get\_continuous\_bounds** ()

Extracts the bounds of the continuous variables.

**get\_continuous\_dims** ()

Returns the dimension of the continuous components of the domain.

**get\_continuous\_space** ()

Extracts the list of dictionaries with continuous components

**get\_discrete\_dims** ()

Returns the dimension of the discrete components of the domain.

**get\_discrete\_grid** ()

Computes a Numpy array with the grid of points that results after crossing the possible outputs of the discrete variables

**get\_discrete\_space** ()

Extracts the list of dictionaries with continuous components

**get\_subspace** (*dims*)

Extracts subspace from the reference of a list of variables in the inputs of the model.

**has\_constraints** ()

Checks if the problem has constraints. Note that the coordinates of the constraints are defined in terms of the model inputs and not in terms of the objective inputs. This means that if bandit or discrete variables are in place, the restrictions should reflect this fact (TODO: implement the mapping of constraints defined on the objective to constraints defined on the model).

**has\_continuous** ()

Returns *true* if the space contains at least one continuous variable, and *false* otherwise

**indicator\_constraints** (*x*)

Returns array of ones and zeros indicating if *x* is within the constraints

**input\_dim** ()

Extracts the input dimension of the domain.

**model\_to\_objective** (*x\_model*)

This function serves as interface between model input vectors and objective input vectors

**objective\_to\_model** (*x\_objective*)

This function serves as interface between objective input vectors and model input vectors

**round\_optimum** (*x*)

Rounds some value *x* to a feasible value in the design space. *x* is expected to be a vector or an array with a single row

**supported\_types** = ['continuous', 'discrete', 'bandit', 'categorical']

**unzip\_inputs** (*X*)

**zip\_inputs** (*X*)

`GPyOpt.core.task.space.bounds_to_space` (*bounds*)

Takes as input a list of tuples with bounds, and create a dictionary to be processed by the class `Design_space`. This function is used to keep the compatibility with previous versions of GPyOpt in which only bounded continuous optimization was possible (and the optimization domain passed as a list of tuples).

## GPyOpt.core.task.variables module

**class** `GPyOpt.core.task.variables.BanditVariable` (*name, domain, dimensionality=None*)

Bases: `GPyOpt.core.task.variables.Variable`

**expand** ()

Builds a list of single dimensional variables representing current variable.

Examples: For single dimensional variable, it is returned as is discrete of (0,2,4) -> discrete of (0,2,4) For multi dimensional variable, a list of variables is returned, each representing a single dimension continuous  $\{0 \leq x \leq 1, 2 \leq y \leq 3\}$  -> continuous  $\{0 \leq x \leq 1\}$ , continuous  $\{2 \leq y \leq 3\}$

**get\_bounds** ()

Returns a list of tuples representing bounds of the variable

**get\_possible\_values** ()

Returns a list of possible variable values

**is\_bandit** ()

**model\_to\_objective** (*x\_model*)

Translates model input to objective input with respect to current variable

**objective\_to\_model** (*x\_objective*)

Translates objective input to model input with respect to current variable

**round** (*value\_array*)

Rounds a bandit variable by selecting the closest point in the domain Closest here is defined by euclidian distance Assumes an 1d array of the same length as the single variable value

**class** `GPyOpt.core.task.variables.CategoricalVariable` (*name, domain, dimensionality=1*)

Bases: `GPyOpt.core.task.variables.Variable`



**expand()**  
Builds a list of single dimensional variables representing current variable.

Examples: For single dimensional variable, it is returned as is discrete of (0,2,4) -> discrete of (0,2,4) For multi dimensional variable, a list of variables is returned, each representing a single dimension continuous  $\{0 \leq x \leq 1, 2 \leq y \leq 3\}$  -> continuous  $\{0 \leq x \leq 1\}$ , continuous  $\{2 \leq y \leq 3\}$

**get\_bounds()**  
Returns a list of tuples representing bounds of the variable

**get\_possible\_values()**  
Returns a list of possible variable values

**is\_bandit()**

**model\_to\_objective** (*x\_model*, *index\_in\_model*)  
Translates model input to objective input with respect to current variable

**objective\_to\_model** (*x\_objective*)  
Translates objective input to model input with respect to current variable

**round** (*value\_array*)  
Rounds a categorical variable by setting to one the max of the given vector and to zero the rest of the entries. Assumes an 1x[number of categories] array (due to one-hot encoding) as an input

**class** GPyOpt.core.task.variables.**ContinuousVariable** (*name*, *domain*, *dimensionality=1*)  
Bases: *GPyOpt.core.task.variables.Variable*

**get\_bounds()**  
Returns a list of tuples representing bounds of the variable

**get\_possible\_values()**  
Returns a list of possible variable values

**is\_bandit()**

**is\_continuous()**

**round** (*value\_array*)  
If value falls within bounds, just return it otherwise return min or max, whichever is closer to the value  
Assumes an 1d array with a single element as an input.

**class** GPyOpt.core.task.variables.**DiscreteVariable** (*name*, *domain*, *dimensionality=1*)  
Bases: *GPyOpt.core.task.variables.Variable*

**get\_bounds()**  
Returns a list of tuples representing bounds of the variable

**get\_possible\_values()**  
Returns a list of possible variable values

**is\_bandit()**

**round** (*value\_array*)  
Rounds a discrete variable by selecting the closest point in the domain Assumes an 1d array with a single element as an input.

**class** GPyOpt.core.task.variables.**Variable** (*name*, *var\_type*, *domain*, *dimensionality*)  
Bases: object

**expand()**  
Builds a list of single dimensional variables representing current variable.

Examples: For single dimensional variable, it is returned as is discrete of (0,2,4) -> discrete of (0,2,4) For multi dimensional variable, a list of variables is returned, each representing a single dimension continuous  $\{0 \leq x \leq 1, 2 \leq y \leq 3\}$  -> continuous  $\{0 \leq x \leq 1\}$ , continuous  $\{2 \leq y \leq 3\}$

**get\_bounds()**

Returns a list of tuples representing bounds of the variable

**get\_possible\_values()**

Returns a list of possible variable values

**is\_continuous()**

**model\_to\_objective**(*x\_model*, *index\_in\_model*)

Translates model input to objective input with respect to current variable

**objective\_to\_model**(*x\_objective*)

Translates objective input to model input with respect to current variable

**round**(*value\_array*)

Rounds the given value to the variable's domain. Value is assumed to be in a 1x[variable dimensionality] numpy array

**set\_index\_in\_model**(*index*)

Allows to set the index of this variable in the model space

**set\_index\_in\_objective**(*index*)

Allows to set the index of this variable in the objective space

**GPyOpt.core.task.variables.create\_variable**(*descriptor*)

Creates a variable from a dictionary descriptor

## Module contents

## 2.2 Submodules

## 2.3 GPyOpt.core.bo module

**class** GPyOpt.core.bo.**BO**(*model*, *space*, *objective*, *acquisition*, *evaluator*, *X\_init*, *Y\_init*=None, *cost*=None, *normalize\_Y*=True, *model\_update\_interval*=1, *de\_duplication*=False)

Bases: object

Runner of Bayesian optimization loop. This class wraps the optimization loop around the different handlers. :param model: GPyOpt model class. :param space: GPyOpt space class. :param objective: GPyOpt objective class. :param acquisition: GPyOpt acquisition class. :param evaluator: GPyOpt evaluator class. :param X\_init: 2d numpy array containing the initial inputs (one per row) of the model. :param Y\_init: 2d numpy array containing the initial outputs (one per row) of the model. :param cost: GPyOpt cost class (default, none). :param normalize\_Y: whether to normalize the outputs before performing any optimization (default, True). :param model\_update\_interval: interval of collected observations after which the model is updated (default, 1). :param de\_duplication: GPyOpt DuplicateManager class. Avoids re-evaluating the objective at previous, pending or infeasible locations (default, False).

**evaluate\_objective**()

Evaluates the objective

**get\_evaluations**()

**plot\_acquisition**(*filename*=None)

**Plots the model and the acquisition function.** if `self.input_dim = 1`: Plots data, mean and variance in one plot and the acquisition function in another plot if `self.input_dim = 2`: as before but it separates the mean and variance of the model in two different plots

**Parameters** `filename` – name of the file where the plot is saved

**plot\_convergence** (*filename=None*)

**Makes twp plots to evaluate the convergence of the model:** plot 1: Iterations vs. distance between consecutive selected `x`'s plot 2: Iterations vs. the mean of the current model in the selected sample.

**Parameters** `filename` – name of the file where the plot is saved

**run\_optimization** (*max\_iter=0, max\_time=inf, eps=1e-08, context=None, verbosity=False, save\_models\_parameters=True, report\_file=None, evaluations\_file=None, models\_file=None*)

Runs Bayesian Optimization for a number '`max_iter`' of iterations (after the initial exploration data)

**Parameters**

- **max\_iter** – exploration horizon, or number of acquisitions. If nothing is provided optimizes the current acquisition.
- **max\_time** – maximum exploration horizon in seconds.
- **eps** – minimum distance between two consecutive `x`'s to keep running the model.
- **verbosity** – flag to print the optimization results after each iteration (default, False).
- **report\_file** – filename of the file where the results of the optimization are saved (default, None).
- **context** – fixes specified variables to a particular context (values) for the optimization run (default, None).

**save\_evaluations** (*evaluations\_file=None*)

Saves a report with the results of the iterations of the optimization

**Parameters** `evaluations_file` – name of the file in which the results are saved.

**save\_models** (*models\_file*)

Saves a report with the results of the iterations of the optimization

**Parameters** `models_file` – name of the file or a file buffer, in which the results are saved.

**save\_report** (*report\_file=None*)

Saves a report with the main results of the optimization.

**Parameters** `report_file` – name of the file in which the results of the optimization are saved.

**suggest\_next\_locations** (*context=None, pending\_X=None, ignored\_X=None*)

Run a single optimization step and return the next locations to evaluate the objective. Number of suggested locations equals to `batch_size`.

**Parameters**

- **context** – fixes specified variables to a particular context (values) for the optimization run (default, None).
- **pending\_X** – matrix of input configurations that are in a pending state (i.e., do not have an evaluation yet) (default, None).

- **ignored\_X** – matrix of input configurations that the user black-lists, i.e., those configurations will not be suggested again (default, None).

## 2.4 GPyOpt.core.errors module

**exception** GPyOpt.core.errors.FullyExploredOptimizationDomainError  
Bases: exceptions.Exception

**exception** GPyOpt.core.errors.InvalidConfigError  
Bases: exceptions.Exception

**exception** GPyOpt.core.errors.InvalidVariableNameError  
Bases: exceptions.Exception

## 2.5 Module contents

---

## GPyOpt.experiment\_design package

---

### 3.1 Submodules

### 3.2 GPyOpt.experiment\_design.base module

**class** GPyOpt.experiment\_design.base.**ExperimentDesign**(*space*)  
Bases: object  
Base class for all experiment designs  
**get\_samples**(*init\_points\_count*)

### 3.3 GPyOpt.experiment\_design.grid\_design module

**class** GPyOpt.experiment\_design.grid\_design.**GridDesign**(*space*)  
Bases: *GPyOpt.experiment\_design.base.ExperimentDesign*  
Grid experiment design. Uses random design for non-continuous variables, and square grid for continuous ones  
**get\_samples**(*init\_points\_count*)  
This method may return less points than requested. The total number of generated points is the smallest closest integer of  $n^d$  to the selected amount of points.  
GPyOpt.experiment\_design.grid\_design.**iroot**(*k, n*)  
GPyOpt.experiment\_design.grid\_design.**multigrid**(*bounds, points\_count*)  
Generates a multidimensional lattice :param bounds: box constraints :param points\_count: number of points per dimension.

### 3.4 GPyOpt.experiment\_design.latin\_design module

**class** GPyOpt.experiment\_design.latin\_design.LatinDesign(*space*)

Bases: *GPyOpt.experiment\_design.base.ExperimentDesign*

Latin experiment design. Uses random design for non-continuous variables, and latin hypercube for continuous ones

**get\_samples**(*init\_points\_count*)

### 3.5 GPyOpt.experiment\_design.random\_design module

**class** GPyOpt.experiment\_design.random\_design.RandomDesign(*space*)

Bases: *GPyOpt.experiment\_design.base.ExperimentDesign*

Random experiment design. Random values for all variables within the given bounds.

**fill\_noncontinuous\_variables**(*samples*)

Fill sample values to non-continuous variables in place

**get\_samples**(*init\_points\_count*)

**get\_samples\_with\_constraints**(*init\_points\_count*)

Draw random samples and only save those that satisfy constraints Finish when required number of samples is generated

**get\_samples\_without\_constraints**(*init\_points\_count*)

GPyOpt.experiment\_design.random\_design.samples\_multidimensional\_uniform(*bounds*,  
points\_count)

Generates a multidimensional grid uniformly distributed. :param bounds: tuple defining the box constraints.  
:points\_count: number of data points to generate.

### 3.6 GPyOpt.experiment\_design.sobol\_design module

**class** GPyOpt.experiment\_design.sobol\_design.SobolDesign(*space*)

Bases: *GPyOpt.experiment\_design.base.ExperimentDesign*

Sobol experiment design. Uses random design for non-continuous variables, and Sobol sequence for continuous ones

**get\_samples**(*init\_points\_count*)

### 3.7 Module contents

GPyOpt.experiment\_design.initial\_design(*design\_name*, *space*, *init\_points\_count*)

### 4.1 Submodules

### 4.2 GPyOpt.interface.config\_parser module

`GPyOpt.interface.config_parser.parser(input_file_path='config.json')`  
Parser for the .json file containing the configuration of the method.

`GPyOpt.interface.config_parser.update_config(config_new, config_default)`  
Updates the loaded method configuration with default values.

### 4.3 GPyOpt.interface.driver module

**class** `GPyOpt.interface.driver.BODriver(config=None, obj_func=None, outputEng=None)`  
Bases: `object`

The class for driving the Bayesian optimization according to the configuration.

**run()**  
Runs the optimization using the previously loaded elements.

### 4.4 GPyOpt.interface.func\_loader module

`GPyOpt.interface.func_loader.load_objective(config)`  
Loads the objective function from a .json file.

## 4.5 GPyOpt.interface.output module

```
class GPyOpt.interface.output.DataSaver (config, outpath=None, prjname="", name="")
    Bases: object

    close()

    save_data (iters, times, offsets, X, Y, bo)

class GPyOpt.interface.output.Logger (config, outpath, prjname="", name="")
    Bases: GPyOpt.interface.output.DataSaver

    close()

    save_data (iters, times, offsets, X, Y, bo)

class GPyOpt.interface.output.OutputEng (config)
    Bases: object

    append_iter (iters, elapsed_time, X, Y, bo, final=False)

    close()

class GPyOpt.interface.output.Report (config, outpath, prjname="", name="")
    Bases: GPyOpt.interface.output.DataSaver

    save_data (iters, times, offsets, X, Y, bo)
```

## 4.6 Module contents





## CHAPTER 5

---

GPyOpt.methods package

---

## 5.1 Submodules

## 5.2 GPyOpt.methods.bayesian\_optimization module

```
class GPyOpt.methods.bayesian_optimization.BayesianOptimization(f,          do-  
    main=None,  
    con-  
    straints=None,  
    cost_withGradients=None,  
    model_type='GP',  
    X=None,  
    Y=None,  ini-  
    tial_design_numdata=5,  
    ini-  
    tial_design_type='random',  
    acquisi-  
    tion_type='EI',  
    normal-  
    ize_Y=True,  
    ex-  
    act_feval=False,  
    acquisi-  
    tion_optimizer_type='lbfgs',  
    model_update_interval=1,  
    evalua-  
    tor_type='sequential',  
    batch_size=1,  
    num_cores=1,  
    ver-  
    bosity=False,  
    ver-  
    bosity_model=False,  
    maxi-  
    mize=False,  
    de_duplication=False,  
    **kwargs)
```

Main class to initialize a Bayesian Optimization method. :param f: function to optimize. It should take 2-dimensional numpy arrays as input and return 2-dimensional outputs (one evaluation per row). :param domain: list of dictionaries containing the description of the inputs variables (See GPyOpt.core.space.Design\_space class for details). :param constraints: list of dictionaries containing the description of the problem constraints (See GPyOpt.core.space.Design\_space class for details). :cost\_withGradients: cost function of the objective. The input can be:

- a function that returns the cost and the derivatives and any set of points in the domain.
- 'evaluation\_time': a Gaussian process (mean) is used to handle the evaluation cost.

**Model\_type** type of model to use as surrogate: - 'GP', standard Gaussian process. - 'GP\_MCMC', Gaussian process with prior in the hyper-parameters. - 'sparseGP', sparse Gaussian process. - 'warperdGP', warped Gaussian process. - 'InputWarpedGP', input warped Gaussian process - 'RF', random forest (scikit-learn).

#### Parameters

- **X** – 2d numpy array containing the initial inputs (one per row) of the model.
- **Y** – 2d numpy array containing the initial outputs (one per row) of the model.
- **normalize\_Y** – whether to normalize the outputs before performing any optimization (default, True).
- **model\_update\_interval** – interval of collected observations after which the model is updated (default, 1).
- **evaluator\_type** – determines the way the objective is evaluated (all methods are equivalent if the batch size is one) - 'sequential', sequential evaluations. - 'random': synchronous batch that selects the first element as in a sequential policy and the rest randomly. - 'local\_penalization': batch method proposed in (Gonzalez et al. 2016). - 'thompson\_sampling': batch method using Thompson sampling.
- **batch\_size** – size of the batch in which the objective is evaluated (default, 1).
- **num\_cores** – number of cores used to evaluate the objective (default, 1).
- **verbosity** – prints the models and other options during the optimization (default, False).
- **maximize** – when True -f maximization of f is done by minimizing -f (default, False).
- **\*\*kwargs** – extra parameters. Can be used to tune the current optimization setup or to use deprecated options in this package release.

**Initial\_design\_numdata** number of initial points that are collected jointly before start running the optimization.

**Initial\_design\_type** type of initial design: - 'random', to collect points in random locations. - 'latin', to collect points in a Latin hypercube (discrete variables are sampled randomly.)

**Acquisition\_type** type of acquisition function to use. - 'EI', expected improvement. - 'EI\_MCMC', integrated expected improvement (requires GP\_MCMC model). - 'MPI', maximum probability of improvement. - 'MPI\_MCMC', maximum probability of improvement (requires GP\_MCMC model). - 'LCB', GP-Lower confidence bound. - 'LCB\_MCMC', integrated GP-Lower confidence bound (requires GP\_MCMC model).

**Exact\_feval** whether the outputs are exact (default, False).

**Acquisition\_optimizer\_type** type of acquisition function to use. - 'lbfgs': L-BFGS. - 'DIRECT': Dividing Rectangles. - 'CMA': covariance matrix adaptation.

---

**Note:** The parameters `bounds`, `kernel`, `numdata_initial_design`, `type_initial_design`, `model_optimize_interval`, `acquisition`, `acquisition_par`, `model_optimize_restarts`, `sparseGP`, `num_inducing` and `normalize` can still be used but will be deprecated in the next version.

---

## 5.3 GPyOpt.methods.modular\_bayesian\_optimization module

```
class GPyOpt.methods.modular_bayesian_optimization.ModularBayesianOptimization(model,  
                                                                           space,  
                                                                           ob-  
                                                                           jec-  
                                                                           tive,  
                                                                           ac-  
                                                                           qui-  
                                                                           si-  
                                                                           tion,  
                                                                           eval-  
                                                                           u-  
                                                                           a-  
                                                                           tor,  
                                                                           X_init,  
                                                                           Y_init=None,  
                                                                           cost=None,  
                                                                           nor-  
                                                                           mal-  
                                                                           ize_Y=True,  
                                                                           model_update_in-  
                                                                           te_duplication=
```

Bases: `GPyOpt.core.bo.BO`

Modular Bayesian optimization. This class wraps the optimization loop around the different handlers.

### Parameters

- **model** – GPyOpt model class.
- **space** – GPyOpt space class.
- **objective** – GPyOpt objective class.
- **acquisition** – GPyOpt acquisition class.
- **evaluator** – GPyOpt evaluator class.
- **X\_init** – 2d numpy array containing the initial inputs (one per row) of the model.
- **Y\_init** – 2d numpy array containing the initial outputs (one per row) of the model.
- **cost** – GPyOpt cost class (default, none).
- **normalize\_Y** – whether to normalize the outputs before performing any optimization (default, True).
- **model\_update\_interval** – interval of collected observations after which the model is updated (default, 1).
- **de\_duplication** – instantiated `de_duplication` GPyOpt class.

## 5.4 Module contents



## 6.1 Submodules

## 6.2 GPyOpt.models.base module

```
class GPyOpt.models.base.BOModel
    Bases: object

    The abstract Model for Bayesian Optimization

    MCMC_sampler = False

    analytical_gradient_prediction = False

    get_fmin()
        Get the minimum of the current model.

    predict(X)
        Get the predicted mean and std at X.

    predict_withGradients(X)
        Get the gradients of the predicted mean and variance at X.

    updateModel(X_all, Y_all, X_new, Y_new)
        Augment the dataset of the model
```

## 6.3 GPyOpt.models.gpmodel module

```
class GPyOpt.models.gpmodel.GPModel(kernel=None, noise_var=None, exact_feval=False,
                                     optimizer='bfgs', max_iters=1000, optimize_restarts=5,
                                     sparse=False, num_inducing=10, verbose=True,
                                     ARD=False)

    Bases: GPyOpt.models.base.BOModel
```

General class for handling a Gaussian Process in GPyOpt.

#### Parameters

- **kernel** – GPy kernel to use in the GP model.
- **noise\_var** – value of the noise variance if known.
- **exact\_feval** – whether noiseless evaluations are available. IMPORTANT to make the optimization work well in noiseless scenarios (default, False).
- **optimizer** – optimizer of the model. Check GPy for details.
- **max\_iters** – maximum number of iterations used to optimize the parameters of the model.
- **optimize\_restarts** – number of restarts in the optimization.
- **sparse** – whether to use a sparse GP (default, False). This is useful when many observations are available.
- **num\_inducing** – number of inducing points if a sparse GP is used.
- **verbose** – print out the model messages (default, False).
- **ARD** – whether ARD is used in the kernel (default, False).

---

**Note:** This model does Maximum likelihood estimation of the hyper-parameters.

---

**analytical\_gradient\_prediction = True**

**copy()**

Makes a safe copy of the model.

**static fromConfig()**

**get\_fmin()**

Returns the location where the posterior mean is takes its minimal value.

**get\_model\_parameters()**

Returns a 2D numpy array with the parameters of the model

**get\_model\_parameters\_names()**

Returns a list with the names of the parameters of the model

**predict(X)**

Predictions with the model. Returns posterior means and standard deviations at X. Note that this is different in GPy where the variances are given.

**predict\_withGradients(X)**

Returns the mean, standard deviation, mean gradient and standard deviation gradient at X.

**updateModel(X\_all, Y\_all, X\_new, Y\_new)**

Updates the model with new observations.

```
class GPyOpt.models.gpmodel.GPModel_MCMC(kernel=None, noise_var=None, exact_feval=False, n_samples=10, n_burnin=100, subsample_interval=10, step_size=0.1, leapfrog_steps=20, verbose=False)
```

Bases: *GPyOpt.models.base.BOModel*

General class for handling a Gaussian Process in GPyOpt.

#### Parameters



- **kernel** – GPy kernel to use in the GP model.
- **noise\_var** – value of the noise variance if known.
- **exact\_feval** – whether noiseless evaluations are available. IMPORTANT to make the optimization work well in noiseless scenarios (default, False).
- **n\_samples** – number of MCMC samples.
- **n\_burnin** – number of samples not used.
- **subsample\_interval** – sub-sample interval in the MCMC.
- **step\_size** – step-size in the MCMC.
- **leapfrog\_steps** – ??
- **verbose** – print out the model messages (default, False).

---

**Note:** This model does MCMC over the hyperparameters.

---

**MCMC\_sampler = True**

**analytical\_gradient\_prediction = True**

**copy()**

Makes a safe copy of the model.

**get\_fmin()**

Returns the location where the posterior mean is takes its minimal value.

**get\_model\_parameters()**

Returns a 2D numpy array with the parameters of the model

**get\_model\_parameters\_names()**

Returns a list with the names of the parameters of the model

**predict(X)**

Predictions with the model for all the MCMC samples. Returns posterior means and standard deviations at X. Note that this is different in GPy where the variances are given.

**predict\_withGradients(X)**

Returns the mean, standard deviation, mean gradient and standard deviation gradient at X for all the MCMC samples.

**updateModel(X\_all, Y\_all, X\_new, Y\_new)**

Updates the model with new observations.

## 6.4 GPyOpt.models.input\_warped\_gpmodel module

```
class GPyOpt.models.input_warped_gpmodel.InputWarpedGPModel(space,  
                                                             warping_function=None,  
                                                             kernel=None,  
                                                             noise_var=None,  
                                                             exact_feval=False,  
                                                             optimizer='bfgs',  
                                                             max_iters=1000,  
                                                             optimize_restarts=5,  
                                                             verbose=False,  
                                                             ARD=False)
```

Bases: *GPyOpt.models.gpmodel.GPModel*

Bayesian Optimization with Input Warped GP using Kumar Warping

The Kumar warping only applies to the numerical variables: continuous and discrete

**space** [object] Instance of Design\_space defined in GPyOpt.core.task.space

**warping\_function** [object, optional] Warping function defined in GPy.util.input\_warping\_functions.py. Default is Kumar warping

**kernel** [object, optional] An instance of kernel function defined in GPy.kern. Default is Matern 52

**noise\_var** [float, optional] Value of the noise variance if known

**exact\_feval** [bool, optional] Whether noiseless evaluations are available. IMPORTANT to make the optimization work well in noiseless scenarios, Default is False

**optimizer** [string, optional] Optimizer of the model. Check GPy for details. Default to bfgs

**max\_iter** [int, optional] Maximum number of iterations used to optimize the parameters of the model. Default is 1000

**optimize\_restarts** [int, optional] Number of restarts in the optimization. Default is 5

**verbose** [bool, optional] Whether to print out the model messages. Default is False

**analytical\_gradient\_prediction** = False

## 6.5 GPyOpt.models.rfmodel module

```
class GPyOpt.models.rfmodel.RFModel(bootstrap=True, criterion='mse', max_depth=None,  
                                     max_features='auto', max_leaf_nodes=None,  
                                     min_samples_leaf=1, min_samples_split=2,  
                                     min_weight_fraction_leaf=0.0, n_estimators=500,  
                                     n_jobs=1, oob_score=False, random_state=None,  
                                     verbose=0, warm_start=False)
```

Bases: *GPyOpt.models.base.BOModel*

General class for handling a Random Forest in GPyOpt.

---

**Note:** The model has been wrapper 'as it is' from Scikit-learn. Check

---

<http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html> for further details.

```

analytical_gradient_prediction = False

get_fmin()
    Get the minimum of the current model.

predict(X)
    Predictions with the model. Returns posterior means and standard deviations at X.

updateModel(X_all, Y_all, X_new, Y_new)
    Updates the model with new observations.

```

## 6.6 GPyOpt.models.warpedgpmodel module

```

class GPyOpt.models.warpedgpmodel.WarpedGPModel(kernel=None, noise_var=None, ex-
                                                    act_feval=False, optimizer='bfgs',
                                                    max_iters=1000, optimize_restarts=5,
                                                    warping_function=None, warp-
                                                    ing_terms=3, verbose=False)

    Bases: GPyOpt.models.base.BOModel

    analytical_gradient_prediction = False

    get_fmin()
        Get the minimum of the current model.

    predict(X)
        Get the predicted mean and std at X.

    updateModel(X_all, Y_all, X_new, Y_new)
        Augment the dataset of the model

```

## 6.7 Module contents

```

GPyOpt.models.select_model(name)

```



## 7.1 Submodules

## 7.2 GPyOpt.objective\_examples.experiments1d module

**class** GPyOpt.objective\_examples.experiments1d.**forrester** (*sd=None*)

Bases: *GPyOpt.objective\_examples.experiments1d.function1d*

Forrester function.

**Parameters** **sd** – standard deviation, to generate noisy evaluations of the function.

$f(X)$

**class** GPyOpt.objective\_examples.experiments1d.**function1d**

This is a benchmark of unidimensional functions interesting to optimize. :param bounds: the box constraints to define the domain in which the function is optimized.

**plot** (*bounds=None*)

## 7.3 GPyOpt.objective\_examples.experiments2d module

**class** GPyOpt.objective\_examples.experiments2d.**beale** (*bounds=None, sd=None*)

Bases: *GPyOpt.objective\_examples.experiments2d.function2d*

Cosines function

**Parameters**

- **bounds** – the box constraints to define the domain in which the function is optimized.
- **sd** – standard deviation, to generate noisy evaluations of the function.

$f(X)$

```
class GPyOpt.objective_examples.experiments2d.branin (bounds=None, a=None,  
                                                    b=None, c=None, r=None,  
                                                    s=None, t=None, sd=None)
```

Bases: *GPyOpt.objective\_examples.experiments2d.function2d*

Branin function

#### Parameters

- **bounds** – the box constraints to define the domain in which the function is optimized.
- **sd** – standard deviation, to generate noisy evaluations of the function.

**f** (*X*)

```
class GPyOpt.objective_examples.experiments2d.cosines (bounds=None, sd=None)
```

Bases: *GPyOpt.objective\_examples.experiments2d.function2d*

Cosines function

#### Parameters

- **bounds** – the box constraints to define the domain in which the function is optimized.
- **sd** – standard deviation, to generate noisy evaluations of the function.

**f** (*X*)

```
class GPyOpt.objective_examples.experiments2d.dropwave (bounds=None, sd=None)
```

Bases: *GPyOpt.objective\_examples.experiments2d.function2d*

Cosines function

#### Parameters

- **bounds** – the box constraints to define the domain in which the function is optimized.
- **sd** – standard deviation, to generate noisy evaluations of the function.

**f** (*X*)

```
class GPyOpt.objective_examples.experiments2d.eggholder (bounds=None, sd=None)
```

**f** (*X*)

```
class GPyOpt.objective_examples.experiments2d.function2d
```

This is a benchmark of bi-dimensional functions interesting to optimize.

**plot** ()

```
class GPyOpt.objective_examples.experiments2d.goldstein (bounds=None, sd=None)
```

Bases: *GPyOpt.objective\_examples.experiments2d.function2d*

Goldstein function

#### Parameters

- **bounds** – the box constraints to define the domain in which the function is optimized.
- **sd** – standard deviation, to generate noisy evaluations of the function.

**f** (*X*)

```
class GPyOpt.objective_examples.experiments2d.mccormick (bounds=None, sd=None)
```

Bases: *GPyOpt.objective\_examples.experiments2d.function2d*

Mccormick function

**Parameters**

- **bounds** – the box constraints to define the domain in which the function is optimized.
- **sd** – standard deviation, to generate noisy evaluations of the function.

 $f(x)$ 

**class** GPyOpt.objective\_examples.experiments2d.**powers** (*bounds=None, sd=None*)  
 Bases: *GPyOpt.objective\_examples.experiments2d.function2d*

Powers function

**Parameters**

- **bounds** – the box constraints to define the domain in which the function is optimized.
- **sd** – standard deviation, to generate noisy evaluations of the function.

 $f(x)$ 

**class** GPyOpt.objective\_examples.experiments2d.**rosenbrock** (*bounds=None, sd=None*)  
 Bases: *GPyOpt.objective\_examples.experiments2d.function2d*

Cosines function

**Parameters**

- **bounds** – the box constraints to define the domain in which the function is optimized.
- **sd** – standard deviation, to generate noisy evaluations of the function.

 $f(X)$ 

**class** GPyOpt.objective\_examples.experiments2d.**sixhumpcamel** (*bounds=None, sd=None*)  
 Bases: *GPyOpt.objective\_examples.experiments2d.function2d*

Six hump camel function

**Parameters**

- **bounds** – the box constraints to define the domain in which the function is optimized.
- **sd** – standard deviation, to generate noisy evaluations of the function.

 $f(x)$ 

## 7.4 GPyOpt.objective\_examples.experimentsNd module

**class** GPyOpt.objective\_examples.experimentsNd.**ackley** (*input\_dim, bounds=None, sd=None*)

Ackley function

**Parameters** **sd** – standard deviation, to generate noisy evaluations of the function.

 $f(X)$ 

**class** GPyOpt.objective\_examples.experimentsNd.**alpine1** (*input\_dim, bounds=None, sd=None*)

Alpine1 function

**Parameters**

- **bounds** – the box constraints to define the domain in which the function is optimized.

- **sd** – standard deviation, to generate noisy evaluations of the function.

$\mathbf{f}(X)$

**class** GPyOpt.objective\_examples.experimentsNd.**alpine2** (*input\_dim*, *bounds=None*,  
*sd=None*)

Alpine2 function

**Parameters**

- **bounds** – the box constraints to define the domain in which the function is optimized.
- **sd** – standard deviation, to generate noisy evaluations of the function.

$\mathbf{f}(X)$

**class** GPyOpt.objective\_examples.experimentsNd.**gSobol** (*a*, *bounds=None*, *sd=None*)  
gSolbol function

**Parameters**

- **a** – one-dimensional array containing the coefficients of the function.
- **sd** – standard deviation, to generate noisy evaluations of the function.

$\mathbf{f}(X)$

## 7.5 Module contents



## 8.1 Submodules

## 8.2 GPyOpt.optimization.acquisition\_optimizer module

```
class GPyOpt.optimization.acquisition_optimizer.AcquisitionOptimizer (space,  
                                                                    opti-  
                                                                    mizer='lbfgs',  
                                                                    **kwargs)
```

Bases: object

General class for acquisition optimizers defined in domains with mix of discrete, continuous, bandit variables

### Parameters

- **space** – design space class from GPyOpt.
- **optimizer** – optimizer to use. Can be selected among: - 'lbfgs': L-BFGS. - 'DIRECT': Dividing Rectangles. - 'CMA': covariance matrix adaptation.

```
optimize (f=None, df=None, f_df=None, duplicate_manager=None)
```

Optimizes the input function.

### Parameters

- **f** – function to optimize.
- **df** – gradient of the function to optimize.
- **f\_df** – returns both the function to optimize and its gradient.

```
class GPyOpt.optimization.acquisition_optimizer.ContextManager (space,      con-  
                                                                    text=None)
```

Bases: object

class to handle the context variable in the optimizer :param space: design space class from GPyOpt. :param context: dictionary of variables and their context values

## 8.3 GPyOpt.optimization.anchor\_points\_generator module

```
class GPyOpt.optimization.anchor_points_generator.AnchorPointsGenerator (space,  
                                                                    de-  
                                                                    sign_type,  
                                                                    num_samples)  
  
    Bases: object  
  
    get (num_anchor=5, duplicate_manager=None, unique=False, context_manager=None)  
  
    get_anchor_point_scores (X)  
  
class GPyOpt.optimization.anchor_points_generator.ObjectiveAnchorPointsGenerator (space,  
                                                                    de-  
                                                                    sign_type,  
                                                                    ob-  
                                                                    jec-  
                                                                    tive,  
                                                                    num_samples)  
  
    Bases: GPyOpt.optimization.anchor_points_generator.AnchorPointsGenerator  
  
    get_anchor_point_scores (X)  
  
class GPyOpt.optimization.anchor_points_generator.RandomAnchorPointsGenerator (space,  
                                                                    de-  
                                                                    sign_type,  
                                                                    num_samples=100)  
  
    Bases: GPyOpt.optimization.anchor_points_generator.AnchorPointsGenerator  
  
    get_anchor_point_scores (X)  
  
class GPyOpt.optimization.anchor_points_generator.ThompsonSamplingAnchorPointsGenerator (spa  
                                                                    de-  
                                                                    sign  
                                                                    mo  
                                                                    num)  
  
    Bases: GPyOpt.optimization.anchor_points_generator.AnchorPointsGenerator  
  
    get_anchor_point_scores (X)
```

## 8.4 GPyOpt.optimization.optimizer module

```
class GPyOpt.optimization.optimizer.OptCma (bounds, maxiter=1000)  
    Bases: GPyOpt.optimization.optimizer.Optimizer  
  
    Wrapper the Covariance Matrix Adaptation Evolutionary strategy (CMA-ES) optimization method. It works  
    generating an stochastic search based on multivariate Gaussian samples. Only requires f and the box constraints  
    to work.  
  
    optimize (x0, f=None, df=None, f_df=None)  
  
        Parameters  


- x0 – initial point for a local optimizer.
- f – function to optimize.
- df – gradient of the function to optimize.
- f_df – returns both the function to optimize and its gradient.

```

**class** GPyOpt.optimization.optimizer.**OptDirect** (*bounds, maxiter=1000*)

Bases: *GPyOpt.optimization.optimizer.Optimizer*

Wrapper for DIRECT optimization method. It works partitioning iteratively the domain of the function. Only requires *f* and the box constraints to work.

**optimize** (*x0, f=None, df=None, f\_df=None*)

#### Parameters

- **x0** – initial point for a local optimizer.
- **f** – function to optimize.
- **df** – gradient of the function to optimize.
- **f\_df** – returns both the function to optimize and its gradient.

**class** GPyOpt.optimization.optimizer.**OptLbfgs** (*bounds, maxiter=1000*)

Bases: *GPyOpt.optimization.optimizer.Optimizer*

Wrapper for l-bfgs-b to use the true or the approximate gradients.

**optimize** (*x0, f=None, df=None, f\_df=None*)

#### Parameters

- **x0** – initial point for a local optimizer.
- **f** – function to optimize.
- **df** – gradient of the function to optimize.
- **f\_df** – returns both the function to optimize and its gradient.

**class** GPyOpt.optimization.optimizer.**OptimizationWithContext** (*x0, f, df=None, f\_df=None, context\_manager=None*)

Bases: *object*

**df\_nc** (*x*)

Wrapper of the derivative of *f*: takes an input *x* with size of the not fixed dimensions expands it and evaluates the gradient of the entire function.

**f\_df\_nc** (*x*)

Wrapper of the derivative of *f*: takes an input *x* with size of the not fixed dimensions expands it and evaluates the gradient of the entire function.

**f\_nc** (*x*)

Wrapper of *f*: takes an input *x* with size of the noncontext dimensions expands it and evaluates the entire function.

**class** GPyOpt.optimization.optimizer.**Optimizer** (*bounds*)

Bases: *object*

Class for a general acquisition optimizer.

**Parameters** **bounds** – list of tuple with bounds of the optimizer

**optimize** (*x0, f=None, df=None, f\_df=None*)

#### Parameters

- **x0** – initial point for a local optimizer.
- **f** – function to optimize.
- **df** – gradient of the function to optimize.

- **f\_df** – returns both the function to optimize and its gradient.

`GPyOpt.optimization.optimizer.apply_optimizer(optimizer, x0, f=None, df=None, f_df=None, duplicate_manager=None, context_manager=None, space=None)`

#### Parameters

- **x0** – initial point for a local optimizer (x0 can be defined with or without the context included).
- **f** – function to optimize.
- **df** – gradient of the function to optimize.
- **f\_df** – returns both the function to optimize and its gradient.
- **duplicate\_manager** – logic to check for duplicate (always operates in the full space, context included)
- **context\_manager** – If provided, x0 (and the optimizer) operates in the space without the context
- **space** – GPyOpt class design space.

`GPyOpt.optimization.optimizer.choose_optimizer(optimizer_name, bounds)`

Selects the type of local optimizer

## 8.5 Module contents

### 9.1 Submodules

### 9.2 GPyOpt.plotting.plots\_bo module

`GPyOpt.plotting.plots_bo.plot_acquisition`(*bounds*, *input\_dim*, *model*, *Xdata*, *Ydata*,  
*acquisition\_function*, *suggested\_sample*, *filename=None*)

Plots of the model and the acquisition function in 1D and 2D examples.

`GPyOpt.plotting.plots_bo.plot_convergence`(*Xdata*, *best\_Y*, *filename=None*)

Plots to evaluate the convergence of standard Bayesian optimization algorithms

### 9.3 Module contents



### 10.1 Submodules

### 10.2 GPyOpt.util.arguments\_manager module

**class** GPyOpt.util.arguments\_manager.**ArgumentsManager** (*kwargs*)

Bases: object

Class to handle extra configurations in the definition of the BayesianOptimization class

**acquisition\_creator** (*acquisition\_type, model, space, acquisition\_optimizer, cost\_withGradients*)

Acquisition chooser from the available options. Extra parameters can be passed via **\*\*kwargs**.

**evaluator\_creator** (*evaluator\_type, acquisition, batch\_size, model\_type, model, space, acquisition\_optimizer*)

Acquisition chooser from the available options. Guide the optimization through sequential or parallel evaluations of the objective.

**model\_creator** (*model\_type, exact\_feval, space*)

Model chooser from the available options. Extra parameters can be passed via **\*\*kwargs**.

### 10.3 GPyOpt.util.duplicate\_manager module

**class** GPyOpt.util.duplicate\_manager.**DuplicateManager** (*space, zipped\_X, pending\_zipped\_X=None, ignored\_zipped\_X=None*)

Bases: object

Class to manage potential duplicates in the suggested candidates.

#### Parameters

- **space** – object managing all the logic related the domain of the optimization

- **zipped\_X** – matrix of evaluated configurations
- **pending\_zipped\_X** – matrix of configurations in the pending state
- **ignored\_zipped\_X** – matrix of configurations that the user desires to ignore (e.g., because they may have led to failures)

**is\_unzipped\_x\_duplicate** (*unzipped\_x*)  
param: unzipped\_x: configuration assumed to be unzipped

**is\_zipped\_x\_duplicate** (*zipped\_x*)  
param: zipped\_x: configuration assumed to be zipped

## 10.4 GPyOpt.util.general module

**GPyOpt.util.general.best\_guess** (*f, X*)  
Gets the best current guess from a vector. :param f: function to evaluate. :param X: locations.

**GPyOpt.util.general.best\_value** (*Y, sign=1*)  
Returns a vector whose components *i* are the minimum (default) or maximum of *Y*[*i*]

**GPyOpt.util.general.compute\_integrated\_acquisition** (*acquisition, x*)  
Used to compute the acquisition function when samples of the hyper-parameters have been generated (used in GP\_MCMC model).

### Parameters

- **acquisition** – acquisition function with GpyOpt model type GP\_MCMC.
- **x** – location where the acquisition is evaluated.

**GPyOpt.util.general.compute\_integrated\_acquisition\_withGradients** (*acquisition, x*)  
Used to compute the acquisition function with gradients when samples of the hyper-parameters have been generated (used in GP\_MCMC model).

### Parameters

- **acquisition** – acquisition function with GpyOpt model type GP\_MCMC.
- **x** – location where the acquisition is evaluated.

**GPyOpt.util.general.evaluate\_function** (*f, X*)  
Returns the evaluation of a function *f* and the time per evaluation

**GPyOpt.util.general.get\_d\_moments** (*model, x*)  
Gradients with respect to *x* of the moments (mean and sdev.) of the GP :param model: GPy model. :param x: location where the gradients are evaluated.

**GPyOpt.util.general.get\_moments** (*model, x*)  
Moments (mean and sdev.) of a GP model at *x*

**GPyOpt.util.general.get\_quantiles** (*acquisition\_par, fmin, m, s*)  
Quantiles of the Gaussian distribution useful to determine the acquisition function values :param acquisition\_par: parameter of the acquisition function :param fmin: current minimum. :param m: vector of means. :param s: vector of standard deviations.

**GPyOpt.util.general.merge\_values** (*values1, values2*)  
Merges two numpy arrays by calculating all possible combinations of rows

**GPyOpt.util.general.normalize** (*Y, normalization\_type='stats'*)  
Normalize the vector *Y* using statistics or its range.



**Parameters**

- **Y** – Row or column vector that you want to normalize.
- **normalization\_type** – String specifying the kind of normalization

to use. Options are ‘stats’ to use mean and standard deviation, or ‘maxmin’ to use the range of function values.  
:return Y\_normalized: The normalized vector.

`GPyOpt.util.general.reshape(x, input_dim)`  
Reshapes x into a matrix with input\_dim columns

`GPyOpt.util.general.samples_multidimensional_uniform(bounds, num_data)`  
Generates a multidimensional grid uniformly distributed. :param bounds: tuple defining the box constraints.  
:num\_data: number of data points to generate.

`GPyOpt.util.general.spawn(f)`  
Function for parallel evaluation of the acquisition function

`GPyOpt.util.general.values_to_array(input_values)`  
Transforms a values of int, float and tuples to a column vector numpy array

## 10.5 GPyOpt.util.io module

`GPyOpt.util.io.gen_datestr()`  
Returns a string with the yy/mm/dd and hh/mm/ss

## 10.6 GPyOpt.util.stats module

## 10.7 Module contents



# CHAPTER 11

---

## Indices and tables

---

- `genindex`
- `modindex`
- `search`



### g

GPpyOpt.acquisitions, 5  
GPpyOpt.acquisitions.base, 5  
GPpyOpt.acquisitions.EI, 1  
GPpyOpt.acquisitions.EI\_mcmc, 2  
GPpyOpt.acquisitions.LCB, 2  
GPpyOpt.acquisitions.LCB\_mcmc, 3  
GPpyOpt.acquisitions.LP, 3  
GPpyOpt.acquisitions.MPI, 4  
GPpyOpt.acquisitions.MPI\_mcmc, 4  
GPpyOpt.core, 16  
GPpyOpt.core.bo, 14  
GPpyOpt.core.errors, 16  
GPpyOpt.core.evaluators, 9  
GPpyOpt.core.evaluators.base, 7  
GPpyOpt.core.evaluators.batch\_local\_penalization, 8  
GPpyOpt.core.evaluators.batch\_random, 8  
GPpyOpt.core.evaluators.batch\_thompson, 8  
GPpyOpt.core.evaluators.sequential, 9  
GPpyOpt.core.task, 14  
GPpyOpt.core.task.cost, 9  
GPpyOpt.core.task.objective, 10  
GPpyOpt.core.task.space, 10  
GPpyOpt.core.task.variables, 12  
GPpyOpt.experiment\_design, 18  
GPpyOpt.experiment\_design.base, 17  
GPpyOpt.experiment\_design.grid\_design, 17  
GPpyOpt.experiment\_design.latin\_design, 18  
GPpyOpt.experiment\_design.random\_design, 18  
GPpyOpt.experiment\_design.sobol\_design, 18  
GPpyOpt.interface, 20  
GPpyOpt.interface.config\_parser, 19  
GPpyOpt.interface.driver, 19  
GPpyOpt.interface.func\_loader, 19  
GPpyOpt.interface.output, 20  
GPpyOpt.methods, 25  
GPpyOpt.methods.bayesian\_optimization, 22  
GPpyOpt.methods.modular\_bayesian\_optimization, 24  
GPpyOpt.models, 31  
GPpyOpt.models.base, 27  
GPpyOpt.models.gpmodel, 27  
GPpyOpt.models.input\_warped\_gpmodel, 30  
GPpyOpt.models.rfmodel, 30  
GPpyOpt.models.warpedgpmodel, 31  
GPpyOpt.objective\_examples, 36  
GPpyOpt.objective\_examples.experiments1d, 33  
GPpyOpt.objective\_examples.experiments2d, 33  
GPpyOpt.objective\_examples.experimentsNd, 35  
GPpyOpt.optimization, 40  
GPpyOpt.optimization.acquisition\_optimizer, 37  
GPpyOpt.optimization.anchor\_points\_generator, 38  
GPpyOpt.optimization.optimizer, 38  
GPpyOpt.plotting, 41  
GPpyOpt.plotting.plots\_bo, 41  
GPpyOpt.util, 45  
GPpyOpt.util.arguments\_manager, 43  
GPpyOpt.util duplicate\_manager, 43  
GPpyOpt.util.general, 44  
GPpyOpt.util.io, 45  
GPpyOpt.util.stats, 45



## A

ackley	(class in GPy- Opt.objective_examples.experimentsNd), 35	analytical_gradient_prediction (GPy- Opt.acquisitions.base.AcquisitionBase at- tribute), 5
acquisition_creator()	(GPy- Opt.util.arguments_manager.ArgumentsManager method), 43	analytical_gradient_prediction (GPy- Opt.acquisitions.EI.AcquisitionEI attribute), 1
acquisition_function()	(GPy- Opt.acquisitions.base.AcquisitionBase method), 5	analytical_gradient_prediction (GPy- Opt.acquisitions.EI_mcmc.AcquisitionEI_MCMC attribute), 2
acquisition_function()	(GPy- Opt.acquisitions.LP.AcquisitionLP method), 3	analytical_gradient_prediction (GPy- Opt.acquisitions.LCB.AcquisitionLCB at- tribute), 2
acquisition_function_withGradients()	(GPy- Opt.acquisitions.base.AcquisitionBase method), 5	analytical_gradient_prediction (GPy- Opt.acquisitions.LCB_mcmc.AcquisitionLCB_MCMC attribute), 3
acquisition_function_withGradients()	(GPy- Opt.acquisitions.LP.AcquisitionLP method), 3	analytical_gradient_prediction (GPy- Opt.acquisitions.LP.AcquisitionLP attribute), 3
AcquisitionBase (class in GPyOpt.acquisitions.base), 5		analytical_gradient_prediction (GPy- Opt.acquisitions.MPI.AcquisitionMPI at- tribute), 4
AcquisitionEI (class in GPyOpt.acquisitions.EI), 1		analytical_gradient_prediction (GPy- Opt.acquisitions.MPI_mcmc.AcquisitionMPI_MCMC attribute), 4
AcquisitionEI_MCMC (class in GPy- Opt.acquisitions.EI_mcmc), 2		analytical_gradient_prediction (GPy- Opt.models.base.BOModel attribute), 27
AcquisitionLCB (class in GPyOpt.acquisitions.LCB), 2		analytical_gradient_prediction (GPy- Opt.models.gpmodel.GPModel attribute), 28
AcquisitionLCB_MCMC (class in GPy- Opt.acquisitions.LCB_mcmc), 3		analytical_gradient_prediction (GPy- Opt.models.gpmodel.GPModel_MCMC attribute), 29
AcquisitionLP (class in GPyOpt.acquisitions.LP), 3		analytical_gradient_prediction (GPy- Opt.models.input_warped_gpmodel.InputWarpedGPModel attribute), 30
AcquisitionMPI (class in GPyOpt.acquisitions.MPI), 4		analytical_gradient_prediction (GPy- Opt.models.rfmodel.RFModel attribute), 30
AcquisitionMPI_MCMC (class in GPy- Opt.acquisitions.MPI_mcmc), 4		analytical_gradient_prediction (GPy- Opt.models.warpedgpmodel.WarpedGPModel
AcquisitionOptimizer (class in GPy- Opt.optimization.acquisition_optimizer), 37		
alpine1 (class in GPy- Opt.objective_examples.experimentsNd), 35		
alpine2 (class in GPy- Opt.objective_examples.experimentsNd), 36		

- attribute), 31
- AnchorPointsGenerator (class in GPy-  
Opt.optimization.anchor\_points\_generator),  
38
- append\_iter() (GPyOpt.interface.output.OutputEng  
method), 20
- apply\_optimizer() (in module GPy-  
Opt.optimization.optimizer), 40
- ArgumentsManager (class in GPy-  
Opt.util.arguments\_manager), 43
- ## B
- BanditVariable (class in GPyOpt.core.task.variables), 12
- BayesianOptimization (class in GPy-  
Opt.methods.bayesian\_optimization), 22
- beale (class in GPy-  
Opt.objective\_examples.experiments2d),  
33
- best\_guess() (in module GPyOpt.util.general), 44
- best\_value() (in module GPyOpt.util.general), 44
- BO (class in GPyOpt.core.bo), 14
- BODriver (class in GPyOpt.interface.driver), 19
- BOModel (class in GPyOpt.models.base), 27
- bounds\_to\_space() (in module GPyOpt.core.task.space),  
12
- branin (class in GPy-  
Opt.objective\_examples.experiments2d),  
33
- ## C
- CategoricalVariable (class in GPy-  
Opt.core.task.variables), 12
- choose\_optimizer() (in module GPy-  
Opt.optimization.optimizer), 40
- close() (GPyOpt.interface.output.DataSaver method), 20
- close() (GPyOpt.interface.output.Logger method), 20
- close() (GPyOpt.interface.output.OutputEng method), 20
- compute\_batch() (GPy-  
Opt.core.evaluators.base.EvaluatorBase  
method), 7
- compute\_batch() (GPy-  
Opt.core.evaluators.base.SamplingBasedBatchEvaluator  
method), 7
- compute\_batch() (GPy-  
Opt.core.evaluators.batch\_local\_penalization.LocalPenalization  
method), 8
- compute\_batch() (GPy-  
Opt.core.evaluators.sequential.Sequential  
method), 9
- compute\_batch\_without\_duplicate\_logic() (GPy-  
Opt.core.evaluators.base.SamplingBasedBatchEvaluator  
method), 7
- compute\_batch\_without\_duplicate\_logic() (GPy-  
Opt.core.evaluators.batch\_random.RandomBatch  
method), 8
- compute\_batch\_without\_duplicate\_logic() (GPy-  
Opt.core.evaluators.batch\_thompson.ThompsonBatch  
method), 8
- compute\_integrated\_acquisition() (in module GPy-  
Opt.util.general), 44
- compute\_integrated\_acquisition\_withGradients() (in  
module GPyOpt.util.general), 44
- constant\_cost\_withGradients() (in module GPy-  
Opt.core.task.cost), 9
- ContextManager (class in GPy-  
Opt.optimization.acquisition\_optimizer),  
37
- ContinuousVariable (class in GPy-  
Opt.core.task.variables), 13
- copy() (GPyOpt.models.gpmodel.GPModel method), 28
- copy() (GPyOpt.models.gpmodel.GPModel\_MCMC  
method), 29
- cosines (class in GPy-  
Opt.objective\_examples.experiments2d),  
34
- CostModel (class in GPyOpt.core.task.cost), 9
- create\_variable() (in module GPyOpt.core.task.variables),  
14
- ## D
- d\_acquisition\_function() (GPy-  
Opt.acquisitions.LP.AcquisitionLP method),  
3
- DataSaver (class in GPyOpt.interface.output), 20
- Design\_space (class in GPyOpt.core.task.space), 10
- df\_nc() (GPyOpt.optimization.optimizer.OptimizationWithContext  
method), 39
- DiscreteVariable (class in GPyOpt.core.task.variables),  
13
- dropwave (class in GPy-  
Opt.objective\_examples.experiments2d),  
34
- DuplicateManager (class in GPy-  
Opt.util.duplicate\_manager), 43
- ## E
- eggholder (class in GPy-  
Opt.objective\_examples.experiments2d),  
14
- estimate\_L() (in module GPy-  
Opt.core.evaluators.batch\_local\_penalization),  
8
- evaluate() (GPyOpt.core.task.objective.Objective  
method), 10
- evaluate() (GPyOpt.core.task.objective.SingleObjective  
method), 10
- evaluate\_function() (in module GPyOpt.util.general), 44
- evaluate\_objective() (GPyOpt.core.bo.BO method), 14



evaluator\_creator() (GPy-  
Opt.util.arguments\_manager.ArgumentsManager  
method), 43

EvaluatorBase (class in GPyOpt.core.evaluators.base), 7

expand() (GPyOpt.core.task.variables.BanditVariable  
method), 12

expand() (GPyOpt.core.task.variables.CategoricalVariable  
method), 12

expand() (GPyOpt.core.task.variables.Variable method),  
13

ExperimentDesign (class in GPy-  
Opt.experiment\_design.base), 17

## F

f() (GPyOpt.objective\_examples.experiments1d.forrester  
method), 33

f() (GPyOpt.objective\_examples.experiments2d.beale  
method), 33

f() (GPyOpt.objective\_examples.experiments2d.branin  
method), 34

f() (GPyOpt.objective\_examples.experiments2d.cosines  
method), 34

f() (GPyOpt.objective\_examples.experiments2d.dropwave  
method), 34

f() (GPyOpt.objective\_examples.experiments2d.eggholder  
method), 34

f() (GPyOpt.objective\_examples.experiments2d.goldstein  
method), 34

f() (GPyOpt.objective\_examples.experiments2d.mccormick  
method), 35

f() (GPyOpt.objective\_examples.experiments2d.powers  
method), 35

f() (GPyOpt.objective\_examples.experiments2d.rosenbrock  
method), 35

f() (GPyOpt.objective\_examples.experiments2d.sixhumpcamel  
method), 35

f() (GPyOpt.objective\_examples.experimentsNd.ackley  
method), 35

f() (GPyOpt.objective\_examples.experimentsNd.alpine1  
method), 36

f() (GPyOpt.objective\_examples.experimentsNd.alpine2  
method), 36

f() (GPyOpt.objective\_examples.experimentsNd.gSobol  
method), 36

f\_df\_nc() (GPyOpt.optimization.optimizer.OptimizationWithContext  
method), 39

f\_nc() (GPyOpt.optimization.optimizer.OptimizationWithContext  
method), 39

fill\_noncontinuous\_variables() (GPy-  
Opt.experiment\_design.random\_design.RandomDesign  
method), 18

find\_variable() (GPyOpt.core.task.space.Design\_space  
method), 11

forrester (class in GPy-  
Opt.objective\_examples.experiments1d),  
33

fromConfig() (GPyOpt.acquisitions.EI.AcquisitionEI  
static method), 1

fromConfig() (GPyOpt.acquisitions.MPI.AcquisitionMPI  
static method), 4

fromConfig() (GPyOpt.core.task.space.Design\_space  
static method), 11

fromConfig() (GPyOpt.models.gpmodel.GPModel static  
method), 28

fromDict() (GPyOpt.acquisitions.base.AcquisitionBase  
static method), 5

FullyExploredOptimizationDomainError, 16

function1d (class in GPy-  
Opt.objective\_examples.experiments1d),  
33

function2d (class in GPy-  
Opt.objective\_examples.experiments2d),  
34

## G

gen\_datestr() (in module GPyOpt.util.io), 45

get() (GPyOpt.optimization.anchor\_points\_generator.AnchorPointsGenerator  
method), 38

get\_anchor\_point\_scores() (GPy-  
Opt.optimization.anchor\_points\_generator.AnchorPointsGenerator  
method), 38

get\_anchor\_point\_scores() (GPy-  
Opt.optimization.anchor\_points\_generator.ObjectiveAnchorPointsGenerator  
method), 38

get\_anchor\_point\_scores() (GPy-  
Opt.optimization.anchor\_points\_generator.RandomAnchorPointsGenerator  
method), 38

get\_anchor\_point\_scores() (GPy-  
Opt.optimization.anchor\_points\_generator.ThompsonSamplingAnchorPointsGenerator  
method), 38

get\_anchor\_points() (GPy-  
Opt.core.evaluators.base.SamplingBasedBatchEvaluator  
method), 7

get\_anchor\_points() (GPy-  
Opt.core.evaluators.batch\_random.RandomBatchEvaluator  
method), 8

get\_anchor\_points() (GPy-  
Opt.core.evaluators.batch\_thompson.ThompsonBatchEvaluator  
method), 8

get\_bandit() (GPyOpt.core.task.space.Design\_space  
method), 11

get\_bounds() (GPyOpt.core.task.space.Design\_space  
method), 11

get\_bounds() (GPyOpt.core.task.variables.BanditVariable  
method), 12

get\_bounds() (GPyOpt.core.task.variables.CategoricalVariable  
method), 13

<code>get_bounds()</code> (GPYOPT.core.task.variables.ContinuousVariable method), 13	<code>get_possible_values()</code> (GPYOPT.core.task.variables.ContinuousVariable method), 13
<code>get_bounds()</code> (GPYOPT.core.task.variables.DiscreteVariable method), 13	<code>get_possible_values()</code> (GPYOPT.core.task.variables.DiscreteVariable method), 13
<code>get_bounds()</code> (GPYOPT.core.task.variables.Variable method), 14	<code>get_possible_values()</code> (GPYOPT.core.task.variables.Variable method), 14
<code>get_continuous_bounds()</code> (GPYOPT.core.task.space.Design_space method), 11	<code>get_quantiles()</code> (in module GPYOPT.util.general), 44
<code>get_continuous_dims()</code> (GPYOPT.core.task.space.Design_space method), 11	<code>get_samples()</code> (GPYOPT.experiment_design.base.ExperimentDesign method), 17
<code>get_continuous_space()</code> (GPYOPT.core.task.space.Design_space method), 11	<code>get_samples()</code> (GPYOPT.experiment_design.grid_design.GridDesign method), 17
<code>get_d_moments()</code> (in module GPYOPT.util.general), 44	<code>get_samples()</code> (GPYOPT.experiment_design.latin_design.LatinDesign method), 18
<code>get_discrete_dims()</code> (GPYOPT.core.task.space.Design_space method), 11	<code>get_samples()</code> (GPYOPT.experiment_design.random_design.RandomDesign method), 18
<code>get_discrete_grid()</code> (GPYOPT.core.task.space.Design_space method), 11	<code>get_samples()</code> (GPYOPT.experiment_design.sobol_design.SobolDesign method), 18
<code>get_discrete_space()</code> (GPYOPT.core.task.space.Design_space method), 11	<code>get_samples_with_constraints()</code> (GPYOPT.experiment_design.random_design.RandomDesign method), 18
<code>get_evaluations()</code> (GPYOPT.core.bo.BO method), 14	<code>get_samples_without_constraints()</code> (GPYOPT.experiment_design.random_design.RandomDesign method), 18
<code>get_fmin()</code> (GPYOPT.models.base.BOModel method), 27	<code>get_subspace()</code> (GPYOPT.core.task.space.Design_space method), 11
<code>get_fmin()</code> (GPYOPT.models.gpmodel.GPModel method), 28	<code>goldstein</code> (class in GPYOPT.objective_examples.experiments2d), 34
<code>get_fmin()</code> (GPYOPT.models.gpmodel.GPModel_MCMC method), 29	<code>GPModel</code> (class in GPYOPT.models.gpmodel), 27
<code>get_fmin()</code> (GPYOPT.models.rfmodel.RFModel method), 31	<code>GPModel_MCMC</code> (class in GPYOPT.models.gpmodel), 28
<code>get_fmin()</code> (GPYOPT.models.warpedgpmodel.WarpedGPModel method), 31	<code>GPYOPT.acquisitions</code> (module), 5
<code>get_model_parameters()</code> (GPYOPT.models.gpmodel.GPModel method), 28	<code>GPYOPT.acquisitions.base</code> (module), 5
<code>get_model_parameters()</code> (GPYOPT.models.gpmodel.GPModel_MCMC method), 29	<code>GPYOPT.acquisitions.EI</code> (module), 1
<code>get_model_parameters_names()</code> (GPYOPT.models.gpmodel.GPModel method), 28	<code>GPYOPT.acquisitions.EI_mcmc</code> (module), 2
<code>get_model_parameters_names()</code> (GPYOPT.models.gpmodel.GPModel_MCMC method), 29	<code>GPYOPT.acquisitions.LCB</code> (module), 2
<code>get_moments()</code> (in module GPYOPT.util.general), 44	<code>GPYOPT.acquisitions.LCB_mcmc</code> (module), 3
<code>get_possible_values()</code> (GPYOPT.core.task.variables.BanditVariable method), 12	<code>GPYOPT.acquisitions.LP</code> (module), 3
<code>get_possible_values()</code> (GPYOPT.core.task.variables.CategoricalVariable method), 13	<code>GPYOPT.acquisitions.MPI</code> (module), 4
	<code>GPYOPT.acquisitions.MPI_mcmc</code> (module), 4
	<code>GPYOPT.core</code> (module), 16
	<code>GPYOPT.core.bo</code> (module), 14
	<code>GPYOPT.core.errors</code> (module), 16
	<code>GPYOPT.core.evaluators</code> (module), 9
	<code>GPYOPT.core.evaluators.base</code> (module), 7
	<code>GPYOPT.core.evaluators.batch_local_penalization</code> (module), 8
	<code>GPYOPT.core.evaluators.batch_random</code> (module), 8
	<code>GPYOPT.core.evaluators.batch_thompson</code> (module), 8
	<code>GPYOPT.core.evaluators.sequential</code> (module), 9
	<code>GPYOPT.core.task</code> (module), 14

GPyOpt.core.task.cost (module), 9  
 GPyOpt.core.task.objective (module), 10  
 GPyOpt.core.task.space (module), 10  
 GPyOpt.core.task.variables (module), 12  
 GPyOpt.experiment\_design (module), 18  
 GPyOpt.experiment\_design.base (module), 17  
 GPyOpt.experiment\_design.grid\_design (module), 17  
 GPyOpt.experiment\_design.latin\_design (module), 18  
 GPyOpt.experiment\_design.random\_design (module), 18  
 GPyOpt.experiment\_design.sobol\_design (module), 18  
 GPyOpt.interface (module), 20  
 GPyOpt.interface.config\_parser (module), 19  
 GPyOpt.interface.driver (module), 19  
 GPyOpt.interface.func\_loader (module), 19  
 GPyOpt.interface.output (module), 20  
 GPyOpt.methods (module), 25  
 GPyOpt.methods.bayesian\_optimization (module), 22  
 GPyOpt.methods.modular\_bayesian\_optimization (module), 24  
 GPyOpt.models (module), 31  
 GPyOpt.models.base (module), 27  
 GPyOpt.models.gpmodel (module), 27  
 GPyOpt.models.input\_warped\_gpmodel (module), 30  
 GPyOpt.models.rfmodel (module), 30  
 GPyOpt.models.warpedgpmodel (module), 31  
 GPyOpt.objective\_examples (module), 36  
 GPyOpt.objective\_examples.experiments1d (module), 33  
 GPyOpt.objective\_examples.experiments2d (module), 33  
 GPyOpt.objective\_examples.experimentsNd (module), 35  
 GPyOpt.optimization (module), 40  
 GPyOpt.optimization.acquisition\_optimizer (module), 37  
 GPyOpt.optimization.anchor\_points\_generator (module), 38  
 GPyOpt.optimization.optimizer (module), 38  
 GPyOpt.plotting (module), 41  
 GPyOpt.plotting.plots\_bo (module), 41  
 GPyOpt.util (module), 45  
 GPyOpt.util.arguments\_manager (module), 43  
 GPyOpt.util.duplicate\_manager (module), 43  
 GPyOpt.util.general (module), 44  
 GPyOpt.util.io (module), 45  
 GPyOpt.util.stats (module), 45  
 GridDesign (class in GPyOpt.experiment\_design.grid\_design), 17  
 gSobol (class in GPyOpt.objective\_examples.experimentsNd), 36

## H

has\_constraints() (GPyOpt.core.task.space.Design\_space method), 11  
 has\_continuous() (GPyOpt.core.task.space.Design\_space method), 11

## I

indicator\_constraints() (GPyOpt.core.task.space.Design\_space method), 11  
 initial\_design() (in module GPyOpt.experiment\_design), 18  
 initialize\_batch() (GPyOpt.core.evaluators.base.SamplingBasedBatchEvaluator method), 7  
 initialize\_batch() (GPyOpt.core.evaluators.batch\_random.RandomBatch method), 8  
 initialize\_batch() (GPyOpt.core.evaluators.batch\_thompson.ThompsonBatch method), 8  
 input\_dim() (GPyOpt.core.task.space.Design\_space method), 12  
 InputWarpedGPModel (class in GPyOpt.models.input\_warped\_gpmodel), 30  
 InvalidConfigError, 16  
 InvalidVariableNameError, 16  
 iroot() (in module GPyOpt.experiment\_design.grid\_design), 17  
 is\_bandit() (GPyOpt.core.task.variables.BanditVariable method), 12  
 is\_bandit() (GPyOpt.core.task.variables.CategoricalVariable method), 13  
 is\_bandit() (GPyOpt.core.task.variables.ContinuousVariable method), 13  
 is\_bandit() (GPyOpt.core.task.variables.DiscreteVariable method), 13  
 is\_continuous() (GPyOpt.core.task.variables.ContinuousVariable method), 13  
 is\_continuous() (GPyOpt.core.task.variables.Variable method), 14  
 is\_unzipped\_x\_duplicate() (GPyOpt.util.duplicate\_manager.DuplicateManager method), 44  
 is\_zipped\_x\_duplicate() (GPyOpt.util.duplicate\_manager.DuplicateManager method), 44

## L

LatinDesign (class in GPyOpt.experiment\_design.latin\_design), 18  
 load\_objective() (in module GPyOpt.interface.func\_loader), 19  
 LocalPenalization (class in GPyOpt.core.evaluators.batch\_local\_penalization), 8  
 Logger (class in GPyOpt.interface.output), 20

## M

mccormick (class in GPy-

- Opt.objective\_examples.experiments2d), 34
- MCMC\_sampler (GPyOpt.models.base.BOModel attribute), 27
- MCMC\_sampler (GPyOpt.models.gpmodel.GPModel\_MCMC attribute), 29
- merge\_values() (in module GPyOpt.util.general), 44
- model\_creator() (GPyOpt.util.arguments\_manager.ArgumentsManager method), 43
- model\_to\_objective() (GPyOpt.core.task.space.Design\_space method), 12
- model\_to\_objective() (GPyOpt.core.task.variables.BanditVariable method), 12
- model\_to\_objective() (GPyOpt.core.task.variables.CategoricalVariable method), 13
- model\_to\_objective() (GPyOpt.core.task.variables.Variable method), 14
- ModularBayesianOptimization (class in GPyOpt.methods.modular\_bayesian\_optimization), 24
- multigrid() (in module GPyOpt.experiment\_design.grid\_design), 17
- ## N
- normalize() (in module GPyOpt.util.general), 44
- ## O
- Objective (class in GPyOpt.core.task.objective), 10
- objective\_to\_model() (GPyOpt.core.task.space.Design\_space method), 12
- objective\_to\_model() (GPyOpt.core.task.variables.BanditVariable method), 12
- objective\_to\_model() (GPyOpt.core.task.variables.CategoricalVariable method), 13
- objective\_to\_model() (GPyOpt.core.task.variables.Variable method), 14
- ObjectiveAnchorPointsGenerator (class in GPyOpt.optimization.anchor\_points\_generator), 38
- OptCma (class in GPyOpt.optimization.optimizer), 38
- OptDirect (class in GPyOpt.optimization.optimizer), 38
- OptimizationWithContext (class in GPyOpt.optimization.optimizer), 39
- optimize() (GPyOpt.acquisitions.base.AcquisitionBase method), 5
- optimize() (GPyOpt.optimization.acquisition\_optimizer.AcquisitionOptimizer method), 37
- optimize() (GPyOpt.optimization.optimizer.OptCma method), 38
- optimize() (GPyOpt.optimization.optimizer.OptDirect method), 39
- optimize() (GPyOpt.optimization.optimizer.Optimizer method), 39
- optimize() (GPyOpt.optimization.optimizer.OptLbfgs method), 39
- optimize\_anchor\_point() (GPyOpt.core.evaluators.base.SamplingBasedBatchEvaluator method), 7
- optimize\_anchor\_point() (GPyOpt.core.evaluators.batch\_random.RandomBatch method), 8
- optimize\_anchor\_point() (GPyOpt.core.evaluators.batch\_thompson.ThompsonBatch method), 8
- Optimizer (class in GPyOpt.optimization.optimizer), 39
- OptLbfgs (class in GPyOpt.optimization.optimizer), 39
- OutputEng (class in GPyOpt.interface.output), 20
- ## P
- parser() (in module GPyOpt.interface.config\_parser), 19
- plot() (GPyOpt.objective\_examples.experiments1d.function1d method), 33
- plot() (GPyOpt.objective\_examples.experiments2d.function2d method), 34
- plot\_acquisition() (GPyOpt.core.bo.BO method), 14
- plot\_acquisition() (in module GPyOpt.plotting.plots\_bo), 41
- plot\_convergence() (GPyOpt.core.bo.BO method), 15
- plot\_convergence() (in module GPyOpt.plotting.plots\_bo), 41
- powers (class in GPyOpt.objective\_examples.experiments2d), 35
- predict() (GPyOpt.models.base.BOModel method), 27
- predict() (GPyOpt.models.gpmodel.GPModel method), 28
- predict() (GPyOpt.models.gpmodel.GPModel\_MCMC method), 29
- predict() (GPyOpt.models.rfmodel.RFModel method), 31
- predict() (GPyOpt.models.warpedgpmodel.WarpedGPModel method), 31
- predict\_withGradients() (GPyOpt.models.base.BOModel method), 27
- predict\_withGradients() (GPyOpt.models.gpmodel.GPModel method), 28
- predict\_withGradients() (GPyOpt.models.gpmodel.GPModel\_MCMC method), 29

## R

RandomAnchorPointsGenerator (class in GPy-  
Opt.optimization.anchor\_points\_generator),  
38

RandomBatch (class in GPy-  
Opt.core.evaluators.batch\_random), 8

RandomDesign (class in GPy-  
Opt.experiment\_design.random\_design),  
18

Report (class in GPyOpt.interface.output), 20

reshape() (in module GPyOpt.util.general), 45

RFModel (class in GPyOpt.models.rfmodel), 30

rosenbrock (class in GPy-  
Opt.objective\_examples.experiments2d),  
35

round() (GPyOpt.core.task.variables.BanditVariable  
method), 12

round() (GPyOpt.core.task.variables.CategoricalVariable  
method), 13

round() (GPyOpt.core.task.variables.ContinuousVariable  
method), 13

round() (GPyOpt.core.task.variables.DiscreteVariable  
method), 13

round() (GPyOpt.core.task.variables.Variable method),  
14

round\_optimum() (GPy-  
Opt.core.task.space.Design\_space method),  
12

run() (GPyOpt.interface.driver.BODriver method), 19

run\_optimization() (GPyOpt.core.bo.BO method), 15

## S

samples\_multidimensional\_uniform() (in module GPy-  
Opt.experiment\_design.random\_design), 18

samples\_multidimensional\_uniform() (in module GPy-  
Opt.util.general), 45

SamplingBasedBatchEvaluator (class in GPy-  
Opt.core.evaluators.base), 7

save\_data() (GPyOpt.interface.output.DataSaver  
method), 20

save\_data() (GPyOpt.interface.output.Logger method),  
20

save\_data() (GPyOpt.interface.output.Report method), 20

save\_evaluations() (GPyOpt.core.bo.BO method), 15

save\_models() (GPyOpt.core.bo.BO method), 15

save\_report() (GPyOpt.core.bo.BO method), 15

select\_acquisition() (in module GPyOpt.acquisitions), 5

select\_evaluator() (in module GPyOpt.core.evaluators), 9

select\_model() (in module GPyOpt.models), 31

Sequential (class in GPyOpt.core.evaluators.sequential),  
9

set\_index\_in\_model() (GPy-  
Opt.core.task.variables.Variable method),  
14

set\_index\_in\_objective() (GPy-  
Opt.core.task.variables.Variable method),  
14

SingleObjective (class in GPyOpt.core.task.objective), 10

sixhumpcamel (class in GPy-  
Opt.objective\_examples.experiments2d),  
35

SobolDesign (class in GPy-  
Opt.experiment\_design.sobol\_design), 18

spawn() (in module GPyOpt.util.general), 45

suggest\_next\_locations() (GPyOpt.core.bo.BO method),  
15

supported\_types (GPyOpt.core.task.space.Design\_space  
attribute), 12

## T

ThompsonBatch (class in GPy-  
Opt.core.evaluators.batch\_thompson), 8

ThompsonSamplingAnchorPointsGenerator  
(class in GPy-  
Opt.optimization.anchor\_points\_generator),  
38

## U

unzip\_inputs() (GPyOpt.core.task.space.Design\_space  
method), 12

update\_batches() (GPy-  
Opt.acquisitions.LP.AcquisitionLP method),  
3

update\_config() (in module GPy-  
Opt.interface.config\_parser), 19

update\_cost\_model() (GPyOpt.core.task.cost.CostModel  
method), 9

updateModel() (GPyOpt.models.base.BOModel method),  
27

updateModel() (GPyOpt.models.gpmodel.GPModel  
method), 28

updateModel() (GPyOpt.models.gpmodel.GPModel\_MCMC  
method), 29

updateModel() (GPyOpt.models.rfmodel.RFModel  
method), 31

updateModel() (GPyOpt.models.warpedgpmodel.WarpedGPModel  
method), 31

## V

values\_to\_array() (in module GPyOpt.util.general), 45

Variable (class in GPyOpt.core.task.variables), 13

## W

WarpedGPModel (class in GPy-  
Opt.models.warpedgpmodel), 31

### Z

`zip_and_tuple()` (GPyOpt.core.evaluators.base.SamplingBasedBatchEvaluator  
method), [7](#)

`zip_inputs()` (GPyOpt.core.task.space.Design\_space  
method), [12](#)