

# API Documentation

API Documentation

March 28, 2014

## Contents

<b>Contents</b>	<b>1</b>
<b>1 Package coinor.grumpy</b>	<b>2</b>
1.1 Modules . . . . .	2
1.2 Variables . . . . .	2
<b>2 Module coinor.grumpy.BB</b>	<b>3</b>
2.1 Functions . . . . .	3
2.2 Variables . . . . .	3
2.3 Class BBTree . . . . .	4
2.3.1 Methods . . . . .	4
2.3.2 Properties . . . . .	12
<b>3 Module coinor.grumpy.forecasting</b>	<b>13</b>
3.1 Variables . . . . .	13
3.2 Class ProgressMeasurement . . . . .	13
3.2.1 Methods . . . . .	13
3.2.2 Properties . . . . .	14
3.3 Class TimeForecast . . . . .	14
3.3.1 Methods . . . . .	14
3.3.2 Properties . . . . .	14
3.4 Class DoubleExponentialSmoothingForecaster . . . . .	15
3.4.1 Methods . . . . .	15
3.4.2 Properties . . . . .	15
3.5 Class ForecastingChainedSequences . . . . .	15
3.5.1 Methods . . . . .	16
3.5.2 Properties . . . . .	16
<b>4 Module coinor.grumpy.polyhedron2D</b>	<b>17</b>
4.1 Functions . . . . .	17
4.2 Variables . . . . .	17
4.3 Class Polyhedron2D . . . . .	17
4.3.1 Methods . . . . .	17

# 1 Package coinor.grumpy

## 1.1 Modules

- **BB** (Section 2, p. 3)
- **forecasting** (Section 3, p. 13)
- **polyhedron2D** (Section 4, p. 17)

## 1.2 Variables

Name	Description
BEST_ESTIMATE	<b>Value:</b> 'Best Estimate'
BEST_FIRST	<b>Value:</b> 'Best First'
BRANCH_STRATEGY	<b>Value:</b> None
DEPTH_FIRST	<b>Value:</b> 'Depth First'
DOT2TEX_INSTALLED	<b>Value:</b> True
DOT2TEX_TEMPLATE	<b>Value:</b> '\n\\documentclass[landscape]{article}\n\\usepackage[x11n...
ETREE_INSTALLED	<b>Value:</b> True
FIXED_BRANCHING	<b>Value:</b> 'Fixed Branching'
INFINITY	<b>Value:</b> 2147483647
LpMaximize	<b>Value:</b> -1
LpStatus	<b>Value:</b> {-3: 'Undefined', -2: 'Unbounded', -1: 'Infeasible', 0: '...
MOST_FRACTIONAL	<b>Value:</b> 'Most Fraction'
PIL_INSTALLED	<b>Value:</b> True
PIPE	<b>Value:</b> -1
PSEUDOCOST_BRANCHING	<b>Value:</b> 'Pseudocost Branching'
PYGAME_INSTALLED	<b>Value:</b> True
SEARCH_STRATEGY	<b>Value:</b> None
STDOUT	<b>Value:</b> -2
XDOT_INSTALLED	<b>Value:</b> True
__package__	<b>Value:</b> 'coinor.grumpy'

## 2 Module *coinor.grumpy.BB*

**Author:** Brady Hunsaker, Osman Ozaltin, Ted Ralphs, Aykut Bulut

### 2.1 Functions

**CreatePerlStyleBooleanFlag**(*parser, flag\_text, variable\_name, help\_text*)

Add two options to an `optparse.OptionParser`, one with a 'no' prefix. Two options are created. One has the `flag_text` and one has 'no' prepended to the `flag_text`. For example, `--foo` and `--nofoo`. This is similar to a common style in Perl.

Args:

`parser`: `optparse.OptionParser` object.  
`flag_text`: String text for the flag.  
`variable_name`: String name of the variable to store the flag results.  
`help_text`: String that describes the flag.

**parse\_options**()

Parse arguments and flags

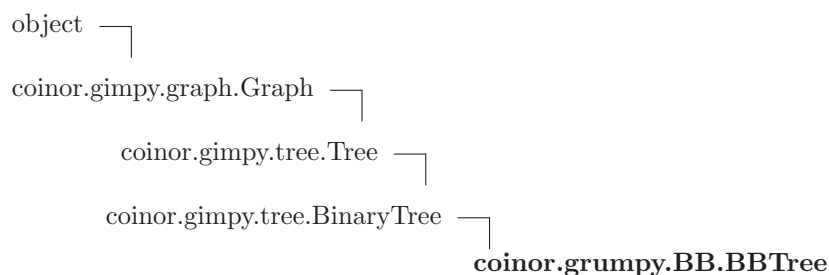
### 2.2 Variables

Name	Description
<code>--maintainer--</code>	This package is for visualizatiing branch-and-bound. It also contains a basicbranch-and-bound implementation primarily for classroom and educational use. Communication with solvers is through a grammar described in separate documentation. Solvers can interface to this class in a number of different ways and a number of different types of images may be created. Images at intervals that can be specified on the command line as well as after new incumbent solutions are found. Note that the generation of tree images takes significantly longer than other images because every node appears in the image. <b>Value:</b> 'Aykut Bulut (aykut@lehigh.edu)'
<code>PYGAME_INSTALLED</code>	<b>Value:</b> True
<code>DOT2TEX_INSTALLED</code>	<b>Value:</b> True
<code>PIL_INSTALLED</code>	<b>Value:</b> True
<code>XDOT_INSTALLED</code>	<b>Value:</b> True
<code>ETREE_INSTALLED</code>	<b>Value:</b> True
<code>BRANCH_STRATEGY</code>	<b>Value:</b> None
<code>SEARCH_STRATEGY</code>	<b>Value:</b> None
<code>MOST_FRACTIONAL</code>	<b>Value:</b> 'Most Fraction'

*continued on next page*

Name	Description
FIXED_BRANCHING	<b>Value:</b> 'Fixed Branching'
PSEUDOCOST_BRANCHING	<b>Value:</b> 'Pseudocost Branching'
DEPTH_FIRST	<b>Value:</b> 'Depth First'
BEST_FIRST	<b>Value:</b> 'Best First'
BEST_ESTIMATE	<b>Value:</b> 'Best Estimate'
INFINITY	<b>Value:</b> 2147483647
DOT2TEX_TEMPLATE	<b>Value:</b> '\n\\documentclass[landscape]{article}\n\\usepackage{x11n...'
--package--	<b>Value:</b> 'coinor.grumpy'

## 2.3 Class *BBTree*



Methods to process and visualize information about a b&b tree. It can process an output file (in a specific format, see BAK project in COIN-OR) of a solver that has three information. See `run.py` in examples directory for this use. Moreover it implements a branch and bound method that can solve binary programs (0-1 variables only) using PuLP as an LP solver. It provides different branching and searching strategies. See `test_strategies.py` in test directory.

This is the main class of GrUMPy. It inherits `BinaryTree` from GIMPy and keeps the entire branch-and-bound tree in self.

### 2.3.1 Methods

```
__init__(self, **attrs)
```

API: `__init__(self, **attrs)`

Description:

Class constructor.

Input:

`attrs`: Tree attributes in keyword arguments format. See `Graph` and `Tree` class for details.

Overrides: `object.__init__` `exitit`(inherited documentation)

```
process_file(self, file_name)
```

```
write_as_dynamic_gexf(self, filename, mode='Dot')
```

**set\_display\_mode**(*self*, *mode*)

API:

    set\_display\_mode(*self*, *value*)

Description:

    Sets display mode to *value*.

Input:

*value*: New display mode.

Post:

    Display mode attribute of *graph* is updated.

Overrides: *coinor.gimpy.graph.Graph.set\_display\_mode* *exitit*(inherited documentation)

**display**(*self*, *item*='all', *basename*='graph', *format*='png', *count*=None)

Displays/Saves images requested. BranchAndBound method calls this method to visualize the branch and bound tree.

Overrides: *coinor.gimpy.graph.Graph.display*

**display\_all**(*self*)

Assumes all the images have the same size.

**display\_image**(*self*, *gnuplot*)

**set\_label**(*self*, *label*)

**set\_logscaley**(*self*, *boolean*)

**set\_fathom**(*self*, *boolean*)

**set\_edge\_limit**(*self*, *limit*)

**set\_sample\_tree**(*self*, *number*)

**AddOrUpdateNode**(*self*, *id*, *parent\_id*, *branch\_direction*, *status*, *lp\_bound*, *integer\_infeasibility\_count*, *integer\_infeasibility\_sum*, *\*\*attrs*)

This method designed to update nodes (in BAK) but we use it for updating/adding arcs. This is because of the tree data structure the authors adopted in BAK. We can divide these attributes such that some will belong to the edge *parent\_id*->*id* and the others belong to the *id* node. The following shows whether the attribute belongs to edge or node. *branch direction* -> edge *status* -> node *lp\_bound* -> node *integer\_infeasibility\_count* -> node *integer\_infeasibility\_sum* -> node *parent\_id* -> node

**IsBetterThan**(*self*, *value1*, *value2*)

Returns True if *value1* is better than *value2* as an objective value.  
This depends on the optimization sense of the instance.

Args:

*value1*: Float.  
*value2*: Float.

Returns:

True if *value1* is better than *value2* as an objective value.

**IsBetterThanIncumbent**(*self*, *value*)

Returns True if the passed value is better than current incumbent.

Args:

*value*: Float to use for comparison.

Returns:

True if the passed value is better than the current incumbent.  
'Better' is determined by the sense of optimization.

**UpdateObjectiveValueLimits**(*self*, *value*)

Updates the min and max objective values if appropriate.

Args:

*value*: Float objective value.

**AddProgressMeasures**(*self*)**GetImageCounterString**(*self*)

Returns a string with the image counter.

**WriteHistogramScript**(*self*, *num\_bins*, *bin\_width*, *max\_bin\_count*, *lp\_bound*, *data\_filename*, *output\_file*)

Write a Gnuplot script file to generate a histogram image.

Args:

*num\_bins*: Integer number of bins for the histogram.  
*bin\_width*: Float width of the bins in terms of objective values.  
*max\_bin\_count*: Integer number of the highest bin count.  
*lp\_bound*: Float value of the current LP bound.  
*data\_filename*: String name of the file; used for display purposes.

---

**AdjustHistogramEndBins**(*self*, *objective\_list*, *num\_bins*, *bin\_width*, *bin\_counts*, *bin\_centers*, *bin\_widths*)

---

Adjusts the two end bins if necessary to make them narrower. The two end bins may need to be narrower than the other bins so that they do not go past the current incumbent value on one end and the current lp bound on the other. So that the histogram is still correct in areas, the height of these bins needs to be adjusted so that the area does not change.

Note that there is likely to be some bias toward taller bins on the ends since they always have a point at one end of their width. It may be more accurate visually to ignore or discount that one point when determining the bin height, but that is not currently done.

Args:

*objective\_list*: List of float objective values.  
*num\_bins*: Integer number of bins.  
*bin\_width*: Float standard width of bins in terms of objective values.  
*bin\_counts*: List of integer counts for each bin.  
*bin\_centers*: List of float coordinates for the center of each bin.  
*bin\_widths*: List of float widths for bins, allowing for individualized widths.

---

**GenerateHistogram**(*self*, *output\_file*=False)

---

Generate files necessary for a histogram image. Two files are necessary: a data file and a Gnuplot script file (which references the data file).

Args:

*time*: Float number of seconds since the start of optimization.

---

**GetImageObjectiveBounds**(*self*, *min\_value*, *max\_value*)

---

Return min and max bounds to be used for images. Images should use bounds that are slightly wider than observed objective values. Also, the special case of a single value must be handled.

Args:

*min\_value*: Float minimum objective value.  
*max\_value*: Float maximum objective value.

Returns:

A tuple of two float values (*lower\_bound*, *upper\_bound*).

**WriteScatterplotScript**(*self*, *data\_filename*, *output\_file*)

Write a Gnuplot script file to generate a scatterplot image.

Args:

*data\_filename*: String name of the file; used for display purposes.

**GenerateScatterplot**(*self*, *output\_file*=False)

Generate files necessary for a scatterplot image.

Two files are necessary: a data file and a Gnuplot script file (which references the data file).

Args:

*output\_file*: if not given the gnuplot image will not be written to disk but returned (to be displayed in pygame window)

**WriteIncumbentPathScript**(*self*, *data\_filename*)

Write a Gnuplot script file to generate an incumbent path image.

Args:

*data\_filename*: String name of the file; used for display purposes.

**WriteAllIncumbentPathsScript**(*self*)

Return a Gnuplot script string to generate an incumbent path image.

Args:

*data\_filenames*: List of string names of files.

**GenerateIncumbentPath**(*self*)

Generate files necessary for an incumbent scatterplot path image. Two files are necessary: a data file and a Gnuplot script file (which references the data file).

**GenerateAllIncumbentPaths**(*self*)

Generate file for a path image with all incumbent paths. Data files were previously generated for each incumbent. This re-uses those files.

**WriteTreeScript**(*self*, *additional\_lines*=None)

Write a Gnuplot script file to generate a tree image.

Args:

*additional\_lines*: String with additional lines to be added to the script file.



**GetTreeFixedHorizontalPositions(*self*)**

Returns horizontal positions for all nodes based on fixed positions.

Returns:

Dictionary of float horizontal positions, keyed by node id.

**GetTreeHorizontalPositions(*self*)**

Returns horizontal positions for all nodes.

Each node is given equal horizontal space.

Returns:

Dictionary of float horizontal positions, keyed by node id.

**WriteDataFileFromList(*self*, *filename*, *data\_list*)**

Write a list of string data to a file with one entry per line.

Args:

*filename*: String filename to open.

*data\_list*: List of string values to write.

**GenerateTreeImage(*self*, *fixed\_horizontal\_positions*=False)**

Generate files necessary for a tree image. Two files are necessary: a data file and a Gnuplot script file (which references the data file).

**ProcessLine(*self*, *line*)**

Process a line of the input file, generating images if appropriate.

Parses the line, updates internal data structures, and creates images if appropriate.

Args:

*line*: String input line to process.

**ProcessHeuristicLine(*self*, *remaining\_tokens*)**

Core processing for a line of type 'heuristic'.

Args:

*remaining\_tokens*: List of string tokens. These are those that remain after any common tokens are processed.

---

**ProcessIntegerLine**(*self, node\_id, parent\_id, branch\_direction, remaining\_tokens*)

---

Core processing for a line of type 'integer'.

Args:

node\_id: String node id.  
parent\_id: String node id of parent.  
branch\_direction: String of 'L' or 'R' indicating whether this node is the left or right child of its parent.  
remaining\_tokens: List of string tokens. These are those that remain after any common tokens are processed.

---

**ProcessFathomedLine**(*self, node\_id, parent\_id, branch\_direction, remaining\_tokens*)

---

Core processing for a line of type 'fathomed'.

Args:

node\_id: String node id.  
parent\_id: String node id of parent.  
branch\_direction: String of 'L' or 'R' indicating whether this node is the left or right child of its parent.  
remaining\_tokens: List of string tokens. These are those that remain after any common tokens are processed.

---

**ProcessPregnantLine**(*self, node\_id, parent\_id, branch\_direction, remaining\_tokens*)

---

Core processing for a line of type 'pregnant'.

Args:

node\_id: String node id.  
parent\_id: String node id of parent.  
branch\_direction: String of 'L' or 'R' indicating whether this node is the left or right child of its parent.  
remaining\_tokens: List of string tokens. These are those that remain after any common tokens are processed.

---

**ProcessBranchedLine**(*self, node\_id, parent\_id, branch\_direction, remaining\_tokens*)

---

Core processing for a line of type 'branched'.

Args:

node\_id: String node id.  
parent\_id: String node id of parent.  
branch\_direction: String of 'L' or 'R' indicating whether this node is the left or right child of its parent.  
remaining\_tokens: List of string tokens. These are those that remain after any common tokens are processed.

**ProcessInfeasibleLine**(*self*, *node\_id*, *parent\_id*, *branch\_direction*, *remaining\_tokens*)

Core processing for a line of type 'infeasible'.

Args:

*node\_id*: String node id.  
*parent\_id*: String node id of parent.  
*branch\_direction*: String of 'L' or 'R' indicating whether this node is the left or right child of its parent.  
*remaining\_tokens*: List of string tokens. These are those that remain after any common tokens are processed.

**ProcessCandidateLine**(*self*, *node\_id*, *parent\_id*, *branch\_direction*, *remaining\_tokens*)

Core processing for a line of type 'candidate'.

Args:

*node\_id*: String node id.  
*parent\_id*: String node id of parent.  
*branch\_direction*: String of 'L' or 'R' indicating whether this node is the left or right child of its parent.  
*remaining\_tokens*: List of string tokens. These are those that remain after any common tokens are processed.

**RunGnuplotOnAllFiles**(*self*)

Runs Gnuplot on all files in *self.gnuplot\_files*.

**CreateAnimatedImages**(*self*)

Create animated images based on the static images.

**GeneratePredictionImages**(*self*)

**GenerateForecastImages**(*self*)

**GenerateRandomMIP**(*self*, *numVars*=40, *numCons*=20, *density*=0.2, *maxObjCoeff*=10, *maxConsCoeff*=10, *tightness*=2, *rand\_seed*=2)

**BranchAndBound**(*self*, *CONSTRAINTS*, *VARIABLES*, *OBJ*, *MAT*, *RHS*, *branch\_strategy*='Most Fraction', *search\_strategy*='Depth First', *complete\_enumeration*=False, *display\_interval*=None)

*Inherited from coinor.gimpy.tree.BinaryTree*

*add\_left\_child()*, *add\_right\_child()*, *add\_root()*, *bfs()*, *del\_node()*, *dfs()*, *get\_left\_child()*, *get\_right\_child()*, *postordereval()*, *print\_nodes()*, *printexp()*, *traverse()*

*Inherited from coinor.gimpy.tree.Tree*

add\_child(), get\_children(), get\_parent()

***Inherited from coinor.gimpy.graph.Graph***

\_\_repr\_\_(), add\_edge(), add\_node(), augment\_cycle(), check\_edge(), create(), create\_cluster(), create\_residual\_graph(), cycle\_canceling(), del\_edge(), edge\_to\_string(), fifo\_label\_correcting(), find\_cycle\_capacity(), find\_feasible\_flow(), get\_approximate\_diameter(), get\_degree(), get\_diameter(), get\_edge\_attr(), get\_edge\_cost(), get\_edge\_list(), get\_edge\_num(), get\_in\_neighbors(), get\_layout(), get\_negative\_cycle(), get\_neighbors(), get\_node(), get\_node\_attr(), get\_node\_list(), get\_node\_num(), get\_out\_neighbors(), get\_simplex\_solution\_graph(), label\_components(), label\_correcting\_check\_cycle(), label\_correcting\_get\_cycle(), label\_strong\_component(), max\_flow(), max\_flow\_preflowpush(), min\_cost\_flow(), minimum\_spanning\_tree\_kruskal(), minimum\_spanning\_tree\_prim(), network\_simplex(), page\_rank(), print\_flow(), process\_edge\_dijkstra(), process\_edge\_flow(), process\_edge\_prim(), process\_edge\_search(), process\_node\_search(), random(), relabel(), search(), set\_edge\_attr(), set\_layout(), set\_node\_attr(), show\_flow(), simplex\_augment\_cycle(), simplex\_compute\_potentials(), simplex\_connect(), simplex\_determine\_leaving\_arc(), simplex\_find\_cycle(), simplex\_find\_tree(), simplex\_identify\_cycle(), simplex\_mark\_entering\_arc(), simplex\_mark\_leaving\_arc(), simplex\_mark\_st\_arcs(), simplex\_optimal(), simplex\_redraw(), simplex\_remove\_arc(), simplex\_search(), simplex\_select\_entering\_arc(), strong\_connect(), tarjan(), to\_string(), write()

***Inherited from object***

\_\_delattr\_\_(), \_\_format\_\_(), \_\_getattr\_\_(), \_\_hash\_\_(), \_\_new\_\_(), \_\_reduce\_\_(), \_\_reduce\_ex\_\_(), \_\_setattr\_\_(), \_\_sizeof\_\_(), \_\_str\_\_(), \_\_subclasshook\_\_()

### 2.3.2 Properties

Name	Description
<i>Inherited from object</i>	
__class__	

### 3 Module `coinor.grumpy.forecasting`

**Author:** Brady Hunsaker, Osman Ozaltin

#### 3.1 Variables

Name	Description
<code>--maintainer--</code>	<p>Double exponential smoothing forecasting in chained sequences.</p> <p>A single sequence is stored in a <code>DoubleExponentialSmoothingForecaster</code>, which also provides forecasts for time to completion based on the stored measures, which should be monotonically decreasing.</p> <p>Sequences are chained together in a <code>ForecastingChainedSequences</code> object. Such an object includes scale factors for each sequence. The scale factors will be applied to the measurements, usually for the purpose of making the chained sequence monotonically decreasing.</p> <p><b>Value:</b> 'Brady Hunsaker (bhunsaker@google.com)'</p>
<code>--package--</code>	<b>Value:</b> 'coinor.grumpy'

#### 3.2 Class `ProgressMeasurement`

object └─ `coinor.grumpy.forecasting.ProgressMeasurement`

Key data recording progress. Data members are public.

##### 3.2.1 Methods

```
--init__(self, time, value, active_node_count, node_count)
```

`x.__init__(...)` initializes `x`; see `help(type(x))` for signature

Overrides: `object.__init__` `exitit`(inherited documentation)

*Inherited from object*

`__delattr__()`, `__format__()`, `__getattr__()`, `__hash__()`, `__new__()`, `__reduce__()`, `__reduce_ex__()`,  
`__repr__()`, `__setattr__()`, `__sizeof__()`, `__str__()`, `__subclasshook__()`

### 3.2.2 Properties

Name	Description
<i>Inherited from object</i>	
<code>__class__</code>	

## 3.3 Class *TimeForecast*

object └─ **coinor.grumpy.forecasting.TimeForecast**

Time-stamped forecast of time remaining.

### 3.3.1 Methods

<b><code>__init__(self, time, forecast)</code></b>
<code>x.__init__(...)</code> initializes x; see <code>help(type(x))</code> for signature
Overrides: <code>object.__init__</code> extit(inherited documentation)

### *Inherited from object*

`__delattr__()`, `__format__()`, `__getattr__()`, `__hash__()`, `__new__()`, `__reduce__()`, `__reduce_ex__()`,  
`__repr__()`, `__setattr__()`, `__sizeof__()`, `__str__()`, `__subclasshook__()`

### 3.3.2 Properties

Name	Description
<i>Inherited from object</i>	
<code>__class__</code>	

### 3.4 Class `DoubleExponentialSmoothingForecaster`

object — `coinor.grumpy.forecasting.DoubleExponentialSmoothingForecaster`

Uses double exponential smoothing to forecast values.

#### 3.4.1 Methods

```
__init__(self, scale_factor, first_value, first_time)
```

`x.__init__(...)` initializes `x`; see `help(type(x))` for signature

Overrides: `object.__init__` `exitit`(inherited documentation)

```
AddMeasure(self, measurement)
```

```
ComputeForecast(self)
```

```
GetForecasts(self)
```

```
GetMeasures(self)
```

#### *Inherited from object*

```
__delattr__(), __format__(), __getattr__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(),
__repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()
```

#### 3.4.2 Properties

Name	Description
<i>Inherited from object</i>	
<code>__class__</code>	

### 3.5 Class `ForecastingChainedSequences`

object — `coinor.grumpy.forecasting.ForecastingChainedSequences`

### 3.5.1 Methods

**`__init__(self)`**`x.__init__(...)` initializes `x`; see `help(type(x))` for signatureOverrides: `object.__init__` `exitit`(inherited documentation)**`AddMeasure(self, time, value, active_node_count, node_count)`****`StartNewSequence(self, scale_factor)`**

Starts a new sequence of measures.

The scale factor should compare only to the previous sequence.

**`GetAllForecasts(self)`****`GetAllMeasures(self)`**

#### *Inherited from object*

`__delattr__()`, `__format__()`, `__getattr__()`, `__hash__()`, `__new__()`, `__reduce__()`, `__reduce_ex__()`,  
`__repr__()`, `__setattr__()`, `__sizeof__()`, `__str__()`, `__subclasshook__()`

### 3.5.2 Properties

Name	Description
<i>Inherited from object</i>	
<code>__class__</code>	



## 4 Module `coinor.grumpy.polyhedron2D`

### 4.1 Functions

```
add_line(ax, coeffs, level, plot_max=None, plot_min=None, color='blue',  
linestyle='solid')
```

### 4.2 Variables

Name	Description
<code>--package--</code>	<b>Value:</b> <code>'coinor.grumpy'</code>

### 4.3 Class `Polyhedron2D`

#### 4.3.1 Methods

```
--init--(self, points=None, rays=None, A=None, b=None)
```

```
make_integer_hull(self)
```

```
determine_hull_size(self)
```

```
determine_plot_size(self, padding=1)
```

```
draw(self, ax, color='blue', linestyle='solid')
```

## Index

- coinor (*package*)
  - coinor.grumpy (*package*), 2
  - coinor.grumpy.BB (*module*), 3–12
  - coinor.grumpy.forecasting (*module*), 13–16
  - coinor.grumpy.polyhedron2D (*module*), 17