fillplots Documentation

Release 0.0.3.dev1

Takafumi Arakaki

September 11, 2016

Contents

Links:

- Documentation (at Read the Docs)
  - Examples
- Repository (at GitHub)
- Issue tracker (at GitHub)
- PyPI
- Travis CI

`fillplots` is a library to plot regions and boundaries given systems of inequality. Here is a simple example to fill region like a piece of pie.

```
>>> from fillplots import plot_regions
>>> plot_regions([
...     [(lambda x: (1.0 - x ** 2) ** 0.5, True),
...      (lambda x: x,)]
... ], xlim=(0, 1), ylim=(0, 1))
<fillplots.api.Plotter object at 0x...>
```

See documentation and examples for more information.

You can install `fillplots` from PyPI:

```
pip install fillplots
```

CHAPTER 1

License

fillplots is licensed under the terms of the BSD 2-Clause License. See the COPYING for more information.

CHAPTER 2

More resources

2.1 fillplots API

class `fillplots.`**`Plotter`**(*regions*, *config='default'*, *xlim=(-10, 10)*, *ylim=(-10, 10)*)

Plotter to fill regions with colors.

Parameters

- **`regions`** (*`list`*) – Each value of list must be a "region specifier" (see below).
- **`config`** (*`str or Config`*) – `'default'`, `'none'` or a *`Config`* object.
- **`xlim`** (*`(int, int)`*) – Limit for x-axis. Default is `(-10, 10)`.
- **`ylim`** (*`(int, int)`*) – Limit for y-axis. Default is `(-10, 10)`.

"Region specifier" is a list of "inequality specifiers". Each "inequality specifier" is a tuple of `(data[, less[, domain]])`.

data is either a callable or a number. If it is a callable, this defines y of boundary as a function of x. If it is a number it is a vertical boundary, inidicated by that number.

less is a bool. True means < (less than) and False means > (larger than). Default is False.

domain is a pair `(min, max)` which indicates defined region of the function. It can be None, which means that this inequality is defined for any real number.

An example of specifying "*x^2 > 0* and *x + 5 > 0*" is:

```
>>> plotter = Plotter(
...     [  # Regions:
...         [  # Inequalities:
...             (lambda x: x ** 2,),  # <-- Boundary data
...             (lambda x: x + 5,)
...         ],
...     ])
```

Internal representation

You can access internal representation as follows. First, `regions` attribute of this class holds a list of *`BaseRegion`* instances:

```
>>> plotter.regions
[<fillplots.regions... object at 0x...>]
```

Each region object has *`inequalities`* attribute, which holds a list of *`BaseInequality`* instances:

```
>>> plotter.regions[0].inequalities
[<fillplots.inequalities... object at 0x...>,
 <fillplots.inequalities... object at 0x...>]
```

Finally, each inequality object has *boundary* attribute to hold an instance of *BaseBoundary*.

```
>>> plotter.regions[0].inequalities[0].boundary
<fillplots.boundaries... object at 0x...>
```

Configuration interface

Each of these "layer" has an instance of *Config* object whose attributes can be modified.

```
>>> plotter.regions[0].inequalities[0].boundary.config
<fillplots.core.Config object at 0x...>
>>> plotter.regions[0].inequalities[0].config
<fillplots.core.Config object at 0x...>
>>> plotter.regions[0].config
<fillplots.core.Config object at 0x...>
>>> plotter.config
<fillplots.core.Config object at 0x...>
```

Modifying upstream configuration propagates to downstream configuration. Let's consider the following configuration:

```
>>> plotter.config.line_args['ls'] = 'dotted'
>>> plotter.config.line_args['lw'] = 8
>>> plotter.regions[0].inequalities[0].config.line_args['ls'] = 'dashed'
```

The boundary object of the 0th inequality mixes the configurations of the inequality and the root plotter object.

```
>>> plotter.regions[0].inequalities[0].boundary.config.line_args['ls']
'dashed'
>>> plotter.regions[0].inequalities[0].boundary.config.line_args['lw']
8
```

The configuration for the 0th inequality does not effect to the 1st inequality and its down stream. So, the configuration of the root plotter object is used:

```
>>> plotter.regions[0].inequalities[1].boundary.config.line_args['ls']
'dotted'
```

config
An instance of *Config*.

You can't set this attribute (i.e., `self.config = config` raises an error). Use `config._set_base` instead.

ax
`matplotlib.axes.Axes` instance used by this plotter.

plot()
Plot regions and boundaries.

plot_boundaries()
Plot boundaries.

plot_positive_direction()
Plot direction that makes LHS of the inequality positive.

plot_regions()
Plot regions.

fillplots.**plot_regions**(*regions*, **args*, ***kwds*)
Create *Plotter* object and call plot function of it.

All arguments are passed to *Plotter*.

Return type *Plotter*

fillplots.**boundary**(*function_or_number*, *domain=None*)
Boundary factory function.

Parameters

- **function_or_number** (*callable or number*) – If it is a callable, it is assumed to be a function that maps x to y. If it is a number, the boundary is a straight line specified by *x* = <*number*>.
- **domain** (*(number, number) or None*) – The boundary is defined on this domain. If it is None, the boundary is defined for any real number. If the argument *function_or_number* is a callable, the domain is on x-axis. If *function_or_number* is a number, the domain is on y-axis.

Return type *BaseBoundary*

Returns An instance of *BaseBoundary* subclass.

fillplots.**annotate_regions**(*regions*, *text*, *horizontalalignment='center'*, *verticalalignment='center'*, ***kwds*)
Annotate *regions* with *text*.

Put only one annotation for each contiguous group of regions.

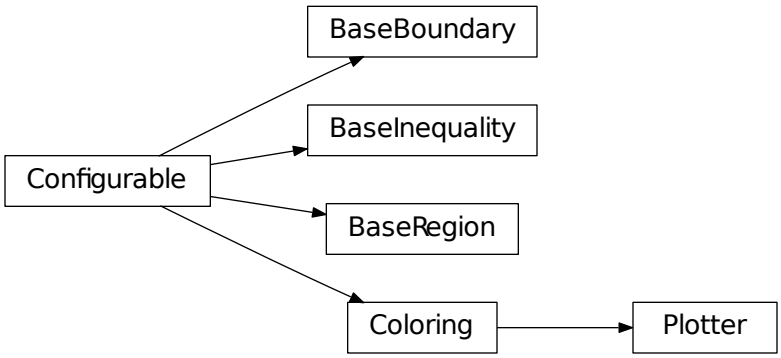
Parameters

- **regions** (list of *BaseRegion*) – These regions are annotated with the same text.
- **text** (*str*) – Annotation text.

Other keywords are passed to `matplotlib.axes.Axes.text()`.

2.1.1 Internal classes

These classes are not meant to initialized from outside of this library. But you can access their instance via `Plotter` and call their methods.



class fillplots.core.**Configurable**(*baseconfig*)

config
An instance of *Config*.

You can't set this attribute (i.e., `self.config = config` raises an error). Use `config._set_base` instead.

ax
`matplotlib.axes.Axes` instance used by this plotter.

class fillplots.boundaries.**BaseBoundary**(*config*, *domain=None*)

plot_boundary()
Plot this boundary.

class fillplots.inequalities.**BaseInequality**(*baseconfig*, *data*, *less=False*, *domain=None*)

boundary = None
An instance of *BaseBoundary* instance.

plot_positive_direction()
Plot direction that makes LHS of the inequality positive.

class fillplots.regions.**BaseRegion**(*config*, *ineqs*)

inequalities = None
List of *BaseInequality* instances.

plot_boundaries()
Plot boundaries.

plot_region()
Plot this region.

2.1.2 Configuration interface

class `fillplots.core.`**`Config`**(**args*, ***kwds*)

Configuration interface.

See also *`Struct`*, which is a parent class of this class.

`line_args` = **None**

Arguments passed to `matplotlib.axes.Axes.plot()` or `matplotlib.axes.Axes.axvline()`.

Default is {}.

`fill_args` = **None**

Arguments passed to `matplotlib.axes.Axes.fill_between()`.

It can take additional keyword argument *autocolor*, which is not defined in matplotlib. *autocolor* can be used to set *facecolor* and *edgecolor* to the same color. If *facecolor* is specified, it is used. Otherwise, the color is generated from `fill_color_cycle`. The reason to add this argument is because it looks like that there is no good way to eliminate edge color natively by matplotlib [1].

Default is {}.

`num_boundary_samples` = **None**

Number of points to be used to draw boundaries.

Default is 1000.

`num_com_samples` = **None**

Number of points to be used to estimate center of mass of region.

Default is 50.

`num_direction_arrows` = **None**

Number of arrows per boundary to indicate positive directions.

Default is 5. It is used by `fillplots.api.Plotter.plot_positive_direction()` etc.

`direction_arrows_size` = **None**

Size of direction arrow as a ratio to axis length.

Default is 0.03. See also `num_direction_arrows`.

class `fillplots.utils.chainstruct.`**`Struct`**(**args*, ***kwds*)

Chain-able struct (Like ChainMap, but for dot-access).

```
>>> zero = Struct()        # most upstream struct
>>> one = Struct(zero)
>>> two = Struct(one)      # most downstream struct
```

Setting attributes of the upstream objects "propagate" to downstream objects.

[1] See: http://stackoverflow.com/questions/14143092/ http://permalink.gmane.org/gmane.comp.python.matplotlib.general/996

```
>>> zero.alpha = 0
>>> two.alpha
0
>>> one.alpha = 1
>>> (zero.alpha, one.alpha, two.alpha)
(0, 1, 1)
```

`Struct` has special treatment for dictionary attributes. Dictionaries of the same attribute in the upstream structs are mixed in the down stream structs.

```
>>> zero.beta = {'gamma': 10}
>>> one.beta = {'delta': 20}
>>> two.beta['gamma']
10
>>> two.beta['delta']
20
>>> two.beta == {'gamma': 10, 'delta': 20}
True
```

Note that changing downstream dictionaries does not change upstream ones.

```
>>> 'delta' in zero.beta
False
>>> two.beta['epsilon'] = 30
>>> 'epsilon' in one.beta
False
```

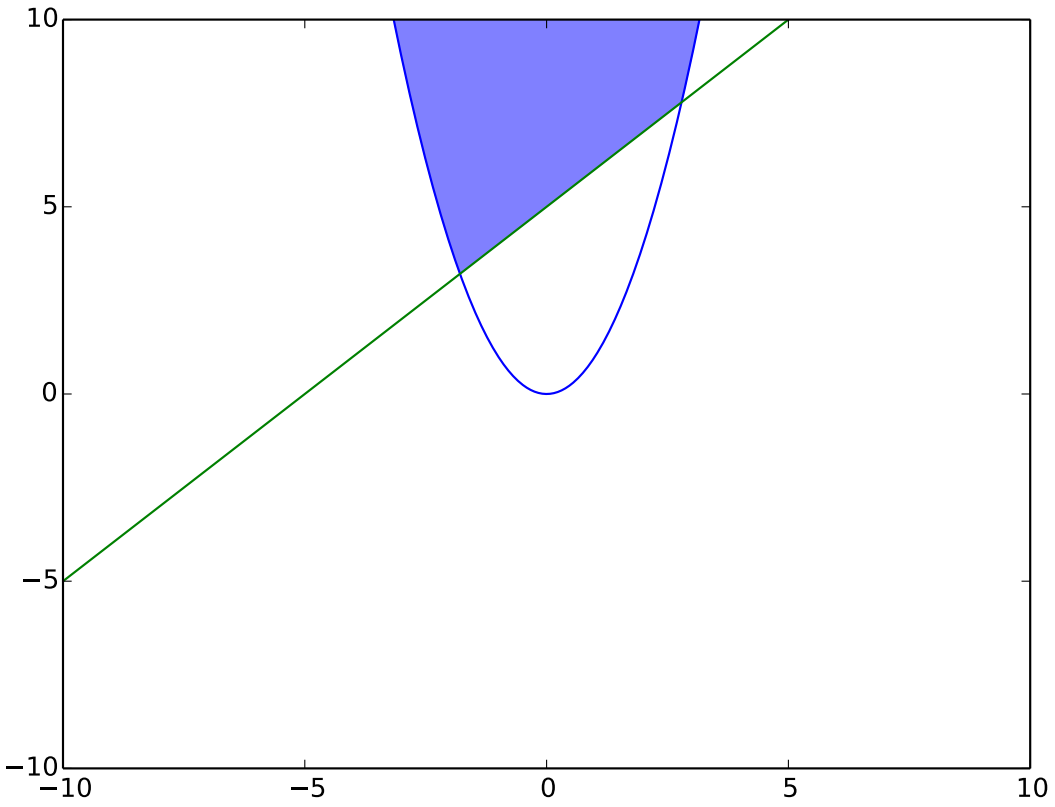
Setting whole dictionary works as expected.

```
>>> one.beta = {'gamma': 110, 'delta': 120}
>>> two.beta['gamma']
110
>>> two.beta['delta']
120
```

2.2 Examples

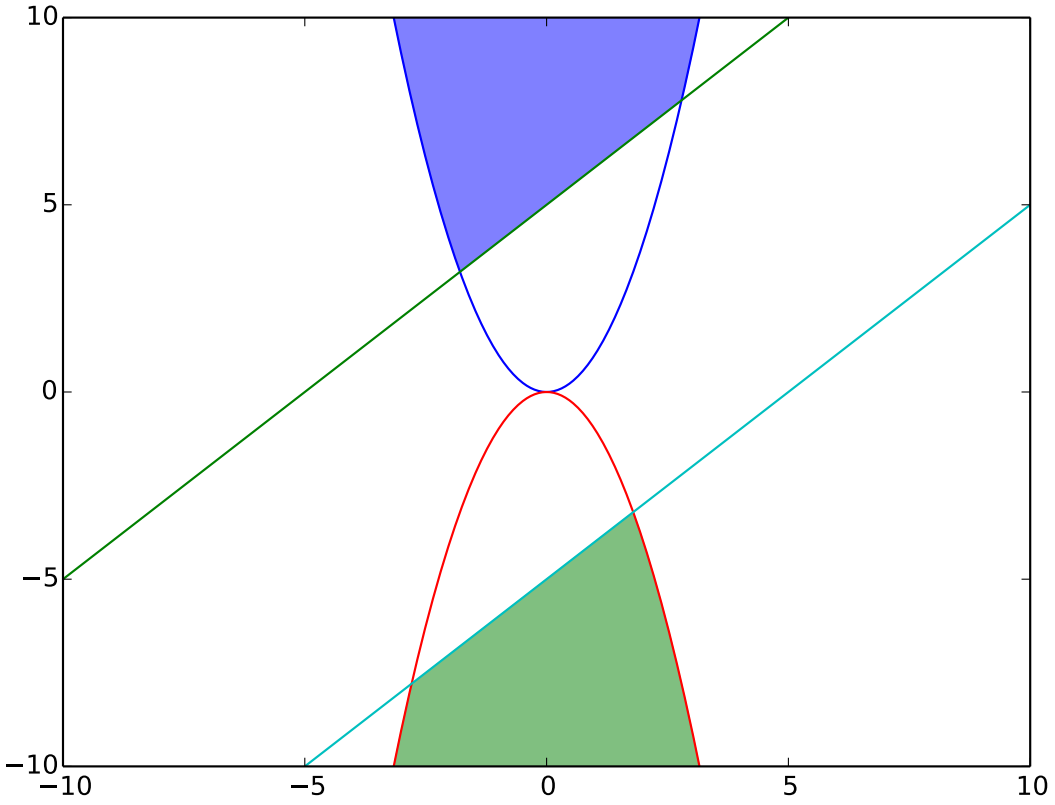
2.2.1 Simple example (`simple.py`)

```
from fillplots import plot_regions
plotter = plot_regions([
    [(lambda x: x ** 2,),  # x ^ 2 > 0 and
     (lambda x: x + 5,)],  # x + 5 > 0
])
```



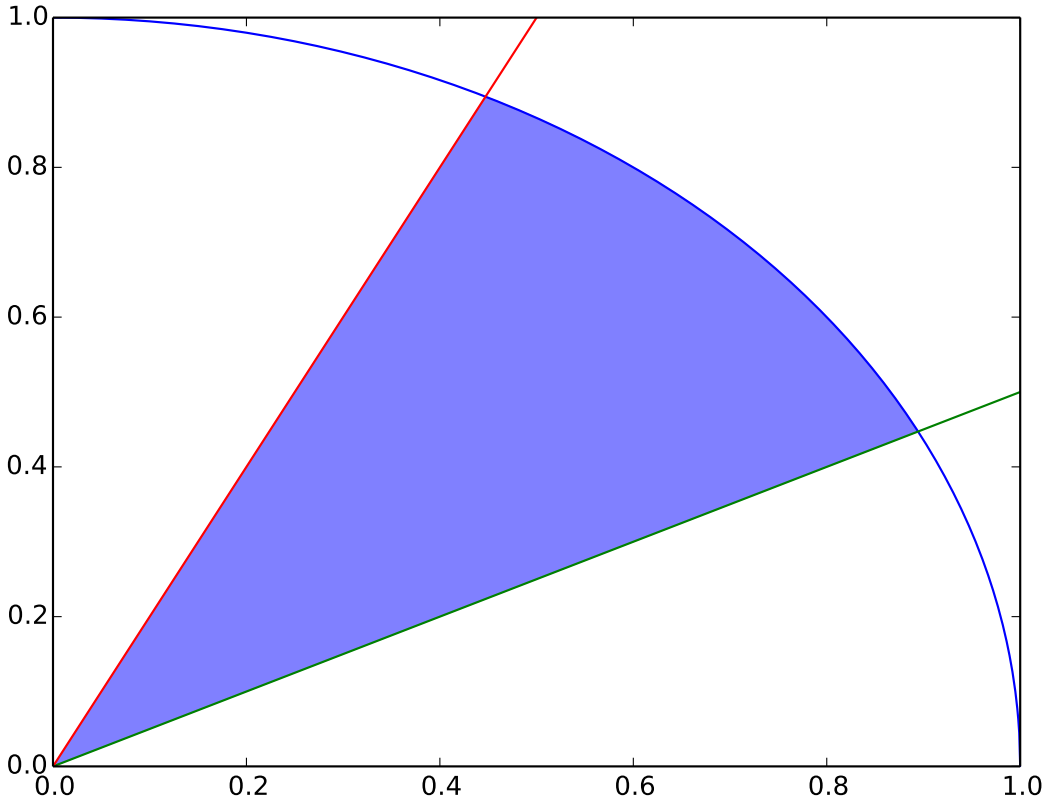
2.2.2 Two regions (`two.py`)

```
from fillplots import plot_regions
plotter = plot_regions([
    [(lambda x: x ** 2,),          # x ^ 2 > 0 and
     (lambda x: x + 5,)],          # x + 5 > 0
    # Another region (True means to use "<" instead of ">"):
    [(lambda x: - x ** 2, True),   # - x^2 < 0 and
     (lambda x: x - 5, True)],     # x - 5 < 0
])
```



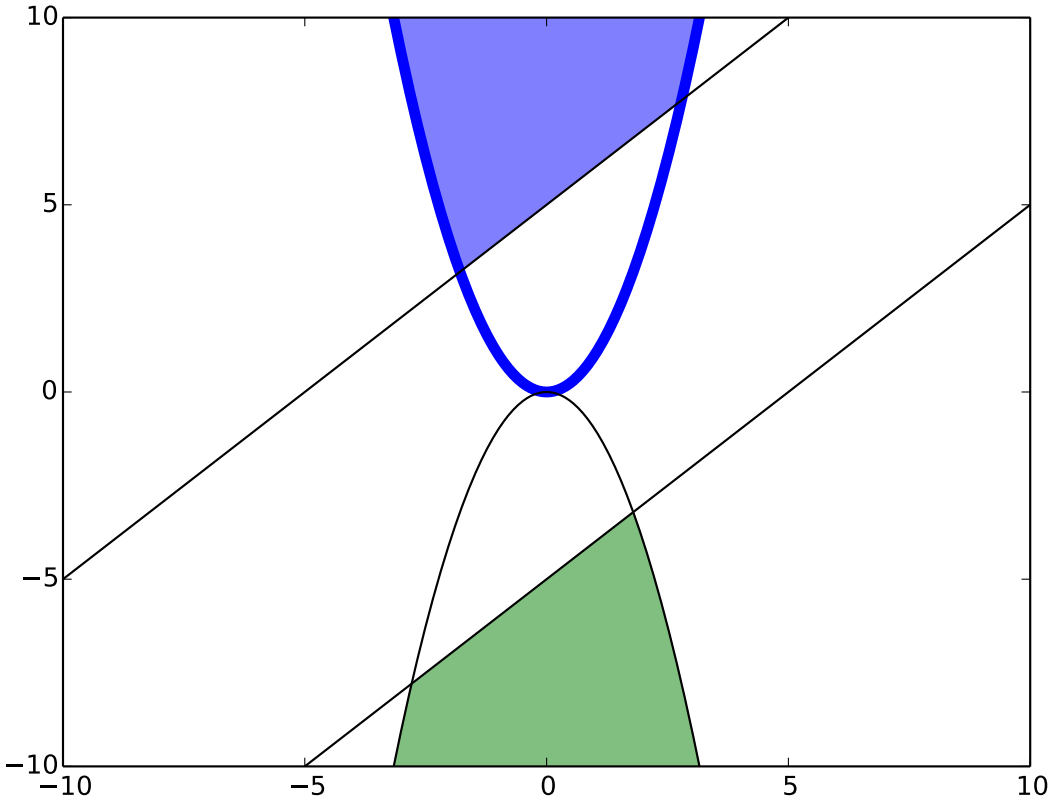
2.2.3 Specifying limits (`limits.py`)

```
from fillplots import plot_regions
plotter = plot_regions([
    [(lambda x: (1.0 - x ** 2) ** 0.5, True),
     (lambda x: 0.5 * x,),
     (lambda x: 2.0 * x, True)]
], xlim=(0, 1), ylim=(0, 1))
```



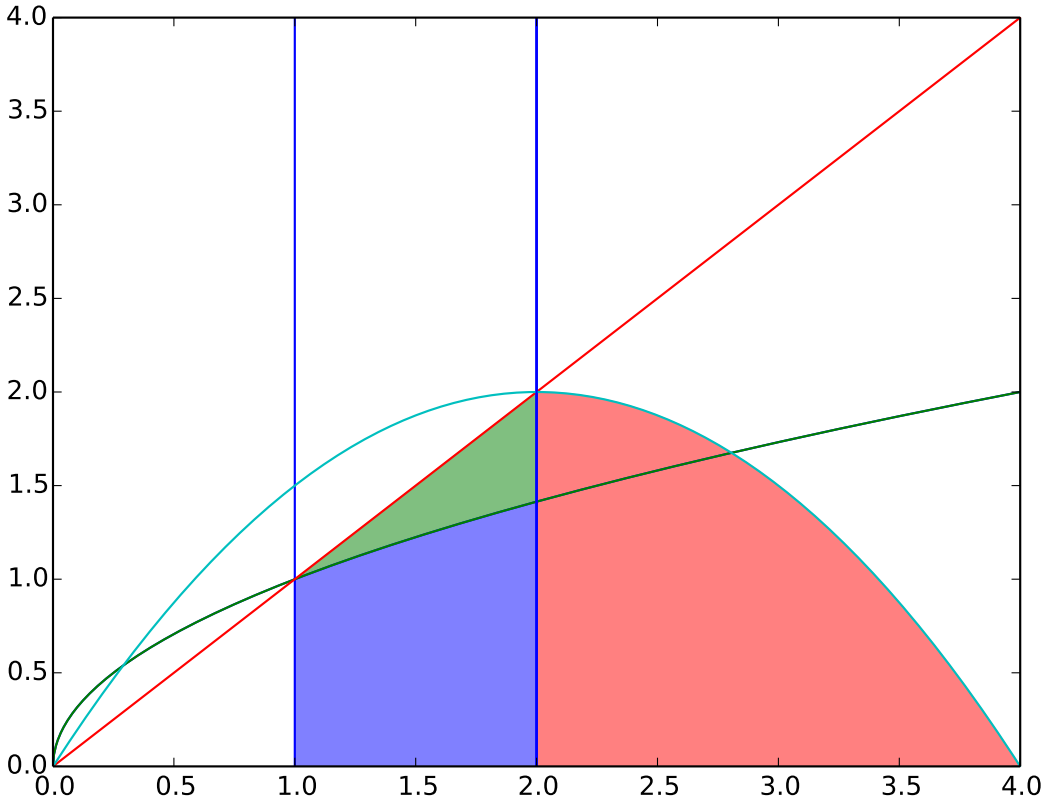
2.2.4 Configure interface (`config_inheritance.py`)

```
from fillplots import Plotter
plotter = Plotter([
    [(lambda x: x ** 2,),
     (lambda x: x + 5,)],
    [(lambda x: - x ** 2, True),
     (lambda x: x - 5, True)],
])

# Upstream configuration "propagates" to downstream ones:
plotter.config.line_args = {'color': 'black'}
# Downstream configuration can be tweaked individually:
plotter.regions[0].inequalities[0].config.line_args = {'color': 'blue',
                                                       'linewidth': 5}

plotter.plot()
```



2.2.5 Draw complex region (`switching.py`)

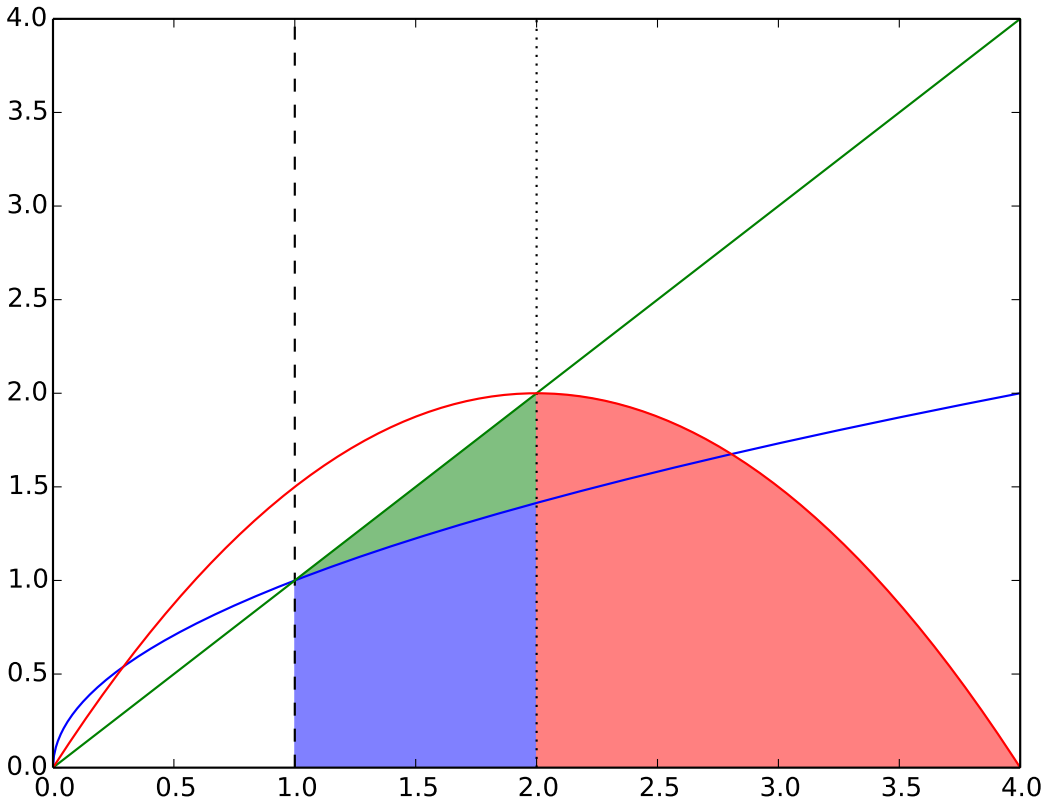
```
# To draw complex region, you need to conditionally switch boundaries.

from fillplots import Plotter
plotter = Plotter([
    [(lambda x: x ** 0.5, True),
     (1,),
     (2, True)],
    [(lambda x: x ** 0.5,),
     (lambda x: x, True),
     (2, True)],
    [(lambda x: x * (4 - x) / 2, True),
     (2,)],
], xlim=(0, 4), ylim=(0, 4))
plotter.plot()
```



2.2.6 Use of boundary object (`switching_uniq_boundary.py`)

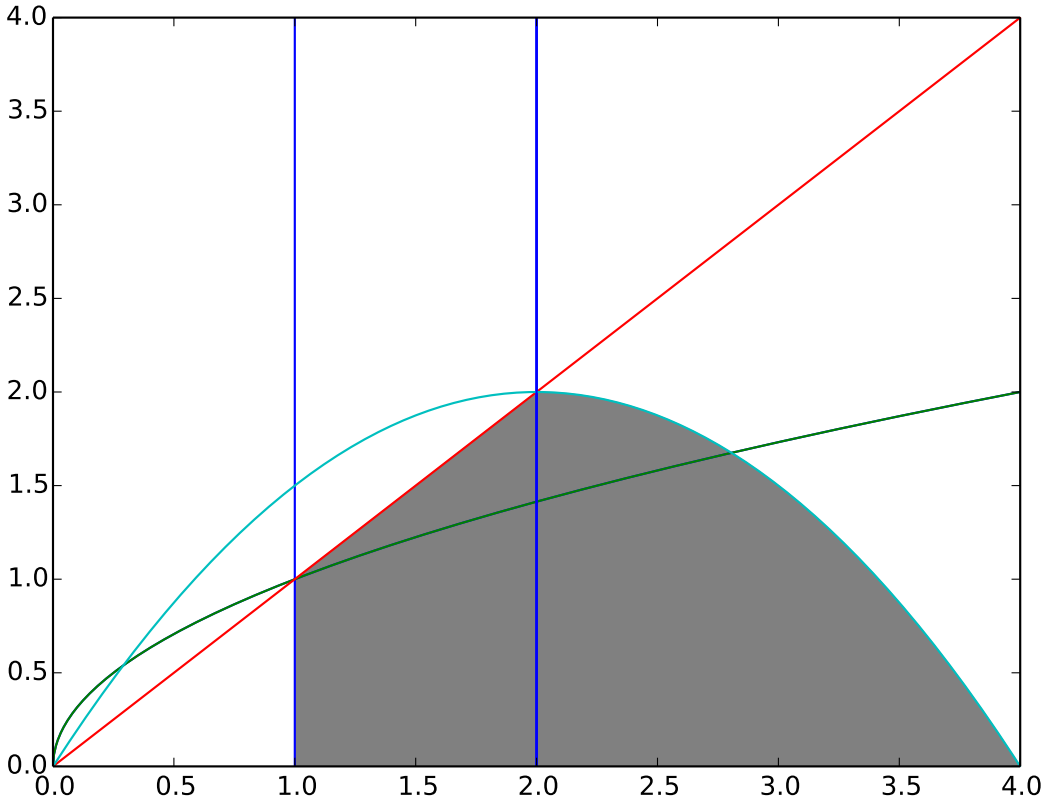
```
# Conditionally switching boundaries make many overlapping boundaries.
# To draw just one line per boundary, you can initialize the boundary
# object before creating the Plotter object.

from fillplots import boundary, Plotter

# Initialize boundaries individually, so that they are recognized as
# one line rather than line per region.
sqrt = boundary(lambda x: x ** 0.5)
one = boundary(1)
two = boundary(2)

# Boundaries can be configured before registering to `Plotter`.
one.config.line_args = {'color': 'k', 'linestyle': 'dashed'}
two.config.line_args = {'color': 'k', 'linestyle': 'dotted'}

plotter = Plotter([
    [(sqrt, True), (one,), (two, True)],
    [(sqrt,), (lambda x: x, True), (two, True)],
    [(lambda x: x * (4 - x) / 2, True),
     (two,)],
], xlim=(0, 4), ylim=(0, 4))
```

```
plotter.plot()
```



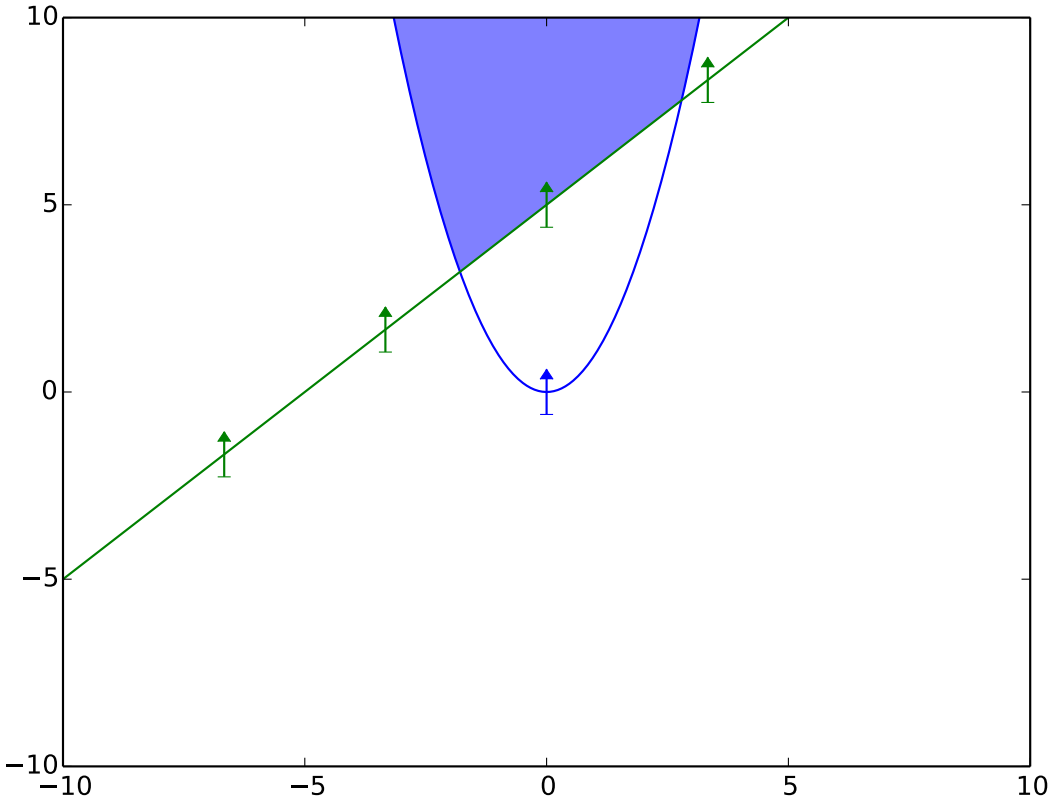
2.2.7 How to set region color (`switching_region_color.py`)

```
from fillplots import Plotter
plotter = Plotter([
    [(lambda x: x ** 0.5, True),
     (1,),
     (2, True)],
    [(lambda x: x ** 0.5,),
     (lambda x: x, True),
     (2, True)],
    [(lambda x: x * (4 - x) / 2, True),
     (2,)],
], xlim=(0, 4), ylim=(0, 4))

for reg in plotter.regions:
    reg.config.fill_args['facecolor'] = 'gray'

plotter.plot()
```



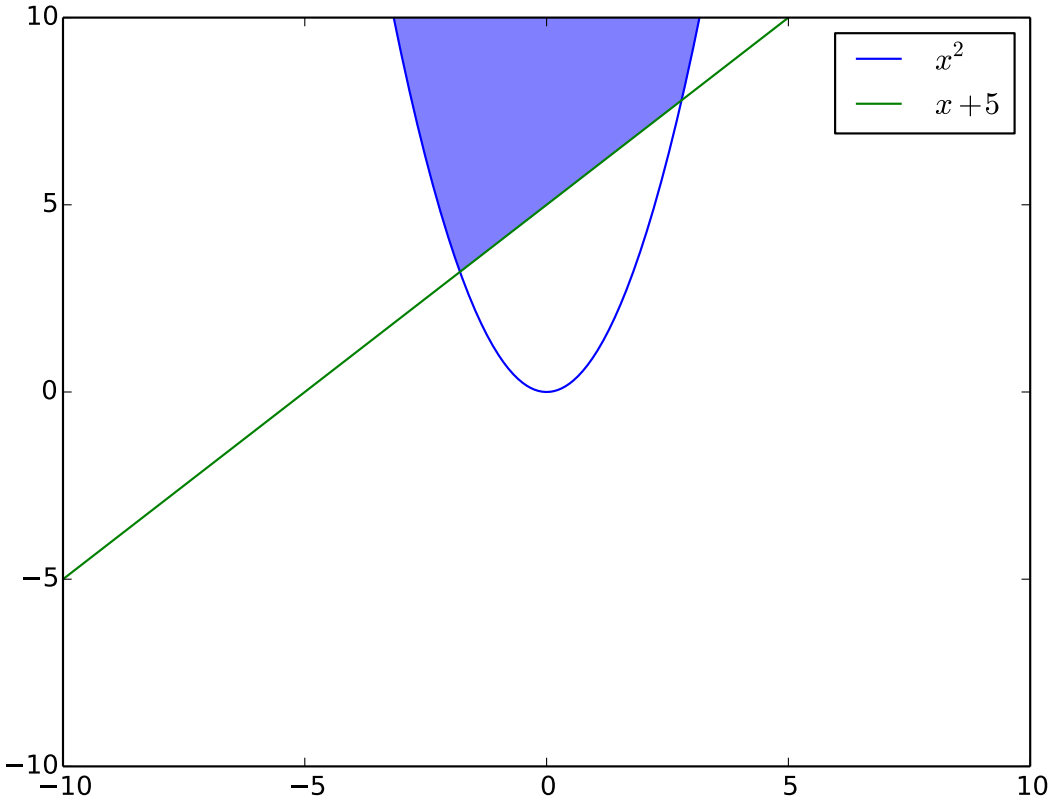
2.2.8 Show which direction makes it positive (`positive_direction.py`)

```
from fillplots import Plotter
plotter = Plotter([
    [(lambda x: x ** 2,),
     (lambda x: x + 5,)],
])
plotter.plot()
plotter.plot_positive_direction()
```



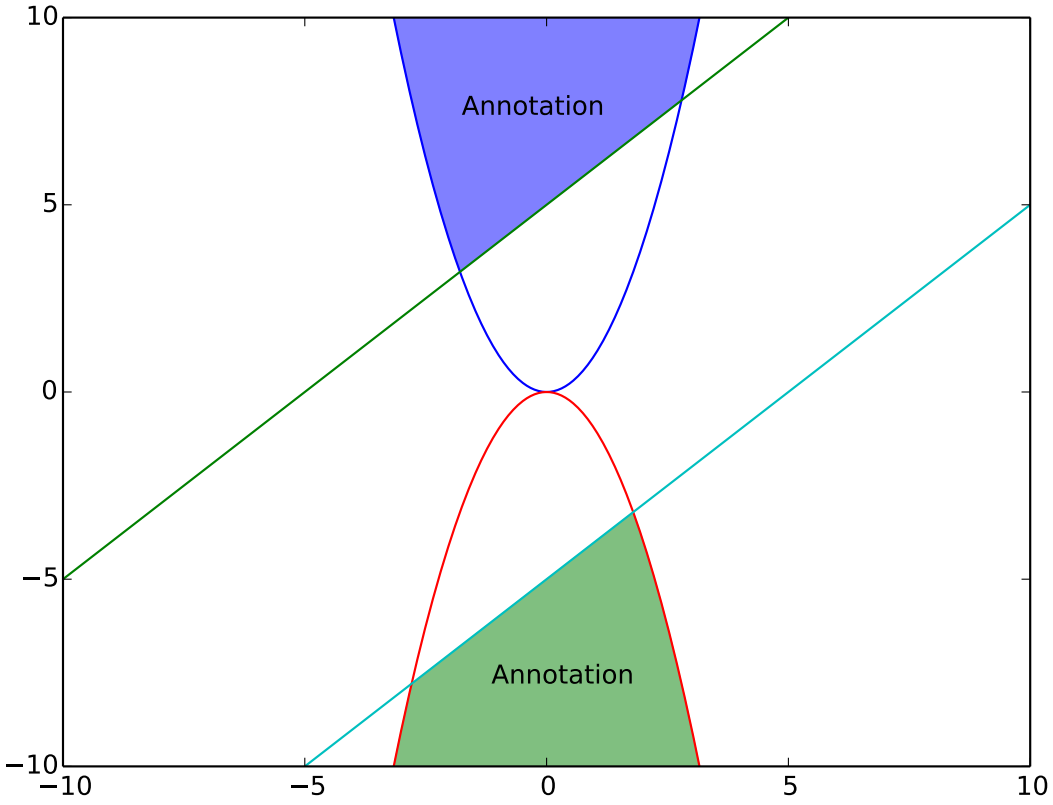
2.2.9 Show boundary labels (`boundary_labels.py`)

```
from fillplots import Plotter
plotter = Plotter([
    [(lambda x: x ** 2,),
     (lambda x: x + 5,)],
])
(ineq0, ineq1) = plotter.regions[0].inequalities
ineq0.boundary.config.line_args['label'] = '$x ^ 2$'
ineq1.boundary.config.line_args['label'] = '$x + 5$'
plotter.plot()
plotter.ax.legend(loc='best')
```



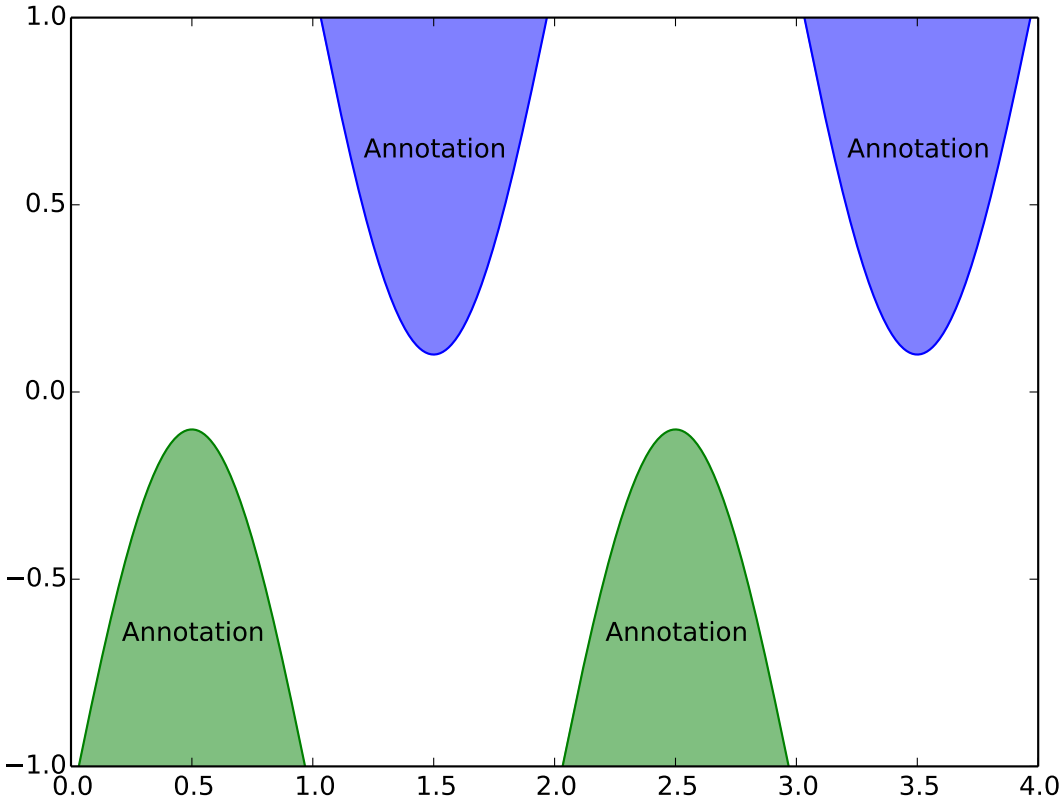
2.2.10 How to annotate regions (`annotate_regions.py`)

```
from fillplots import Plotter, annotate_regions
plotter = Plotter([
    [(lambda x: x ** 2,),
     (lambda x: x + 5,)],
    [(lambda x: - x ** 2, True),
     (lambda x: x - 5, True)],
])
plotter.plot()
annotate_regions(plotter.regions, 'Annotation')
```



2.2.11 Annotations with discontinuous regions (`divide_regions.py`)

```
import numpy
from fillplots import Plotter, annotate_regions
plotter = Plotter([
    [(lambda x: numpy.sin(numpy.pi * x) + 1.1,)],
    [(lambda x: numpy.sin(numpy.pi * x) - 1.1, True)],
], xlim=(0, 4), ylim=(-1, 1))
plotter.plot()
annotate_regions(plotter.regions, 'Annotation')
```



- genindex
- modindex
- search

f