How to use piecewise_funcs

Lib `piecewise_funcs` has only one dependency and that is `portion` a package providing interval arithmetic to define each branch's domain and to ensure that each branch is valid and does not intersect with any of the other branches, as this would be ambiguous. There is no need excplicitly import `portion` as importing `piecewise_funcs` imports `portion`'s functionality while simultaneously namespaces it. For more info on `portion` visit its pypi project page.

Let's import our lib under an alias:

In [1]:
```
import piecewise_funcs as pwf
```

A quick showcase of how we define intervals using `piecewise_funcs` follows.

In [2]:
```
pwf.open(-1,0)
```

Out[2]:
```
(-1,0)
```

In [3]:
```
pwf.closedopen(-1, 0)
```

Out[3]:
```
[-1,0)
```

In [4]:
```
pwf.closed(-1, 0)
```

Out[4]:
```
[-1,0]
```

to get a complement of an interval we do:

In [5]:
```
~pwf.closed(-1, 0)
```

Out[5]:
```
(-inf,-1) | (0,+inf)
```

to merge two intervals we do:

In [6]:
```
pwf.closed(-1, 0) | pwf.closedopen(0,1)
```

Out[6]:
```
[-1,1)
```

to intersect them we do:

In [7]:
```
pwf.closed(-1, 0) & pwf.closedopen(0,1)
```

Out[7]:
```
[0]
```

There are two main classes in the `piecewise_funcs` module, namely `PiecewiseGeneric` an an abstract class that defines the interface, and the concrete `PiecewiseFunc` that implements the functionality, i.e. `__call__`, `min`, `max`.

For example, let's say we wish to represent the Heaviside step function:

$$f(x) = \begin{cases} 1, & \text{if } x > 0 \\ 0, & \text{if } x \leq 0 \end{cases}$$

In order this piecewise function using a `PiecewiseFunc` we can either explicitly construct it providing a `Sequence` of intervals and a `Sequence` of `Callables`, as below:

In [8]:
```
pw_1 = pwf.PiecewiseFunc([pwf.open(0, pwf.inf)],
                         [lambda x: 1., lambda x: 0.])
```

or we can construct it using the factory method `from_funcdef`, which expects a regular function object containig `if/elif/else` & `return` statements constant or linear expressions:

In [9]:
```
def heaviside(x: float) -> float:
    if x > 0:
        return 1.
    return 0.
```

In [10]:
```
pw_2 = pwf.PiecewiseFunc.from_funcdef(heaviside)
```

Then suppose we have the following list of numbers:

In [11]:
```
x = [-1., 0., 1.]
```

and we want to evaluate the heaviside function on `x`, we just call the `PiecewiseFunc` object passing `x` as an argument, the resulting iterator can be converted to a `list` using a starred expression:

In [12]:
```
[*pw_1(x)]
```

Out[12]:
```
[0.0, 0.0, 1.0]
```

and

In [13]:
```
[*pw_2(x)]
```

Out[13]:
```
[0.0, 0.0, 1.0]
```

Furthermore, to find argmin/min:

In [14]:
```
pw_1.min(x)
```

Out[14]:
```
(0, 0.0)
```

while for argmax/max:

In [15]:
```
pw_1.max(x)
```

Out[15]:
```
(2, 1.0)
```

and:

In [16]:
```
pw_2.min(x)
```

Out[16]:
```
(0, 0.0)
```

In [17]:
```
pw_2.max(x)
```

Out[17]:
```
(2, 1.0)
```

An example of a piecewise linear function:

In [18]:
```
def func(x: float) -> float:
    if -1.3 <= x < 4.2:
        return 2*x + 1
    elif x > 10.:
        return x
    return -x
```

In [19]:
```
pw = pwf.PiecewiseFunc.from_funcdef(func)
```

In [20]:
```
x = [-5., 0., 10.5]
```

In [21]:
```
[*pw(x)]
```

Out[21]:
```
[5.0, 1.0, 10.5]
```

In [22]:
```
pw.min(x)
```

Out[22]:
```
(1, 1.0)
```

In [23]:
```
pw.max(x)
```

Out[23]:
```
(2, 10.5)
```