optimix Documentation

Release 1.2.13

Danilo Horta

Jul 07, 2017

Table of contents

CHAPTER 1

Install

The recommended way of installing it is via conda:

```
conda install -c conda-forge optimix
```

An alternative way would be via pip:

```
pip install optimix
```

CHAPTER 2

Quick start

A function in `optimix` sense is a map of input and variable values to output values. Those values can be scalars or vectors but what is most important is that input values and variable values are not treated in the same way. Derivatives are always over variables (not inputs) and thus the optimisation is always performed over variables. The reason for this is that input values are meant to be datasets and variable values are meant to be model parameters. I hope the next examples help clarify this matter.

2.1 Single input

We create a class that inherits from *optimix.function.Function*, define a scalar variable (which we named here as *scale*), and implement the *optimix.function.Function.value()* and *optimix.function.Function.gradient()* methods.

```
>>> from optimix import Function, Scalar, minimize
>>>
>>> class Quadratic(Function):
...
...   def __init__(self):
...     super(Quadratic, self).__init__(scale=Scalar(1.0))
...
...   def value(self, x):
...     s = self.variables().get('scale').value
...     return (s - 5.0)**2 * x / 2.0
...
...   def gradient(self, x):
...     s = self.variables().get('scale').value
...     return dict(scale=(s - 5.0) * x)
>>>
>>> f = Quadratic()
>>> x = 1.2
>>>
>>> print("Function evaluation at x: %g" % f.value(x))
Function evaluation at x: 9.6
```

```
>>> print("Function gradient at x: %s" % f.gradient(x))
Function gradient at x: {'scale': ndarray_listener(-4.8)}
>>>
>>> # For optimizating the function, we need a dataset
>>> # associated with it. This is accomplished by calling
>>> # the set_data method as follows:
>>> f.set_data(x)
>>>
>>> func = f.feed()
>>>
>>> # We are now ready to minimize the function.
>>> minimize(func, verbose=False)
>>>
>>> # This will print the optimum found.
>>> print("Optimum found: %g" % f.variables().get('scale').value)
Optimum found: 5
```

And an example for two variables:

```
>>> from optimix import Function, Scalar, minimize
>>>
>>> class Quadratic(Function):
...
...   def __init__(self):
...     super(Quadratic, self).__init__(a=Scalar(1.0), b=Scalar(1.0))
...
...   def value(self, x):
...     a = self.variables().get('a').value
...     b = self.variables().get('b').value
...     return ((a - 5.0)**2 + (b + 5.0)**2 * x) / 2.0
...
...   def gradient(self, x):
...     a = self.variables().get('a').value
...     b = self.variables().get('b').value
...     return dict(a=(a - 5.0), b=(b + 5.0) * x)
>>>
>>> f = Quadratic()
>>> x = 1.2
>>> f.set_data(x)
>>> minimize(f.feed(), verbose=False)
>>> a = f.variables().get('a').value
>>> b = f.variables().get('b').value
>>> print("Optimum found: (%g, %g)" % (a, b))
Optimum found: (5, -5)
```

2.2 Double inputs

You can also define a function of two inputs (or more) in a very natural way:

```
>>> from optimix import Function, Scalar, minimize
>>>
>>> class Quadratic(Function):
...
...   def __init__(self):
...     super(Quadratic, self).__init__(a=Scalar(1.0), b=Scalar(1.0))
```

```
...
...   def value(self, x0, x1):
...     a = self.variables().get('a').value
...     b = self.variables().get('b').value
...     return ((a - 5.0)**2 * x0 + (b + 5.0)**2 * x1) / 2.0
...
...   def gradient(self, x0, x1):
...     a = self.variables().get('a').value
...     b = self.variables().get('b').value
...     return dict(a=2 * (a - 5.0) * x0, b=2 * (b + 5.0) * x1)
>>>
>>> f = Quadratic()
>>> x0 = 2.3
>>> x1 = 1.0
>>> f.set_data((x0, x1))
>>> minimize(f.feed(), verbose=False)
>>>
>>> a = f.variables().get('a').value
>>> b = f.variables().get('b').value
>>> print("Optimum found: (%g, %g)" % (a, b))
Optimum found: (5, -5)
```

CHAPTER 3

Types

class optimix.types.**Scalar**(*value*)
Scalar variable type.

It holds a `float64` value, listen to changes, and fix or unfix its value.

Parameters **value** (*float*) – initial value.

asarray()
Return a `numpy.ndarray` representation.

copy()
Return a copy.

fix()
Set it fixed.

isfixed
Return whether it is fixed or not.

listen(*you*)
Request a callback for value modification.

Parameters **you** (*object*) – an instance having `__call__` attribute.

shape
Shape according to `numpy`.

size
Size according to `numpy`.

unfix()
Set it unfixed.

class optimix.types.**Vector**(*value*)
Vector variable type.

It holds an array of `float64` values, listen to changes, and fix or unfix its values.

Parameters **value** (*float*) – initial value.

asarray()
Return a numpy.ndarray representation.

copy()
Return a copy.

fix()
Set it fixed.

isfixed
Return whether it is fixed or not.

listen(*you*)
Request a callback for value modification.

Parameters **you** (*object*) – an instance having __call__ attribute.

shape
Shape according to numpy.

size
Size according to numpy.

unfix()
Set it unfixed.

CHAPTER 4

Variables

class optimix.variables.**Variables**

Set of variables.

names()

Return the variable names.

select(*fixed*)

Return a subset of variables according to `fixed`.

set(*x*)

Set variable values via a dictionary mapping name to value.

CHAPTER 5

Function

class optimix.function.**Function**(***kwargs*)

Base-class for object representing functions.

Parameters **kwargs** (*dict*) – Map of variable name to variable value.

feed(*purpose='learn'*)

Return a function with attached data.

fix(*var_name*)

Set a variable fixed.

Parameters **var_name** (*str*) – variable name.

gradient(**args*)

Evaluate the gradient at the `args` point.

Parameters **args** (*tuple*) – Point at the gradient evaluation. The length of this `tuple()` is defined by the user.

Returns Map between variables to their gradient values.

Return type dict

isfixed(*var_name*)

Return whether a variable it is fixed or not.

Parameters **var_name** (*str*) – variable name.

set_data(*data*, *purpose='learn'*)

Set a named data source.

Parameters **purpose** (*str*) – Name of the data source.

set_nodata(*purpose='learn'*)

Disable data feeding.

Parameters **purpose** (*str*) – Name of the data source.

unfix(*var_name*)

Set a variable unfixed.

Parameters **var_name** (*str*) – variable name.

unset_data(*purpose='learn'*)
Unset a named data source.

Parameters **purpose** (*str*) – Name of the data source.

value(**args*)
Evaluate the function at the `args` point.

Parameters **args** (*tuple*) – Point at the evaluation. The length of this `tuple()` is defined by the user.

Returns Function evaluated at `args`.

Return type float or array_like

variables()
Function variables.

CHAPTER 6

Optimization methods

6.1 L-BFGS-B

optimix.optimize.**minimize**(*function*, *verbose=True*)

Minimize a function using L-BFGS-B.

Parameters

- **function** (*object*) – Objective function. It has to implement the *optimix.function.Function* interface.
- **verbose** (*bool*) – `True` for verbose output; `False` otherwise.

optimix.optimize.**maximize**(*function*, *verbose=True*)

Maximize a function using L-BFGS-B.

Parameters

- **function** (*object*) – Objective function. It has to implement the *optimix.function.Function* interface.
- **verbose** (*bool*) – `True` for verbose output; `False` otherwise.

6.2 Brent's method

optimix.optimize.**minimize_scalar**(*function*, *verbose=True*)

Minimize a scalar function using Brent's method.

Parameters

- **function** (*object*) – Objective function. It has to implement the *optimix.function.Function* interface.
- **verbose** (*bool*) – `True` for verbose output; `False` otherwise.

optimix.optimize.**maximize_scalar**(*function*, *verbose=False*)
Maximize a scalar function using Brent's method.

Parameters

- **function** (*object*) – Objective function. It has to implement the *optimix.function.Function* interface.
- **verbose** (*bool*) – `True` for verbose output; `False` otherwise.

optimix.optimize.**minimize**(*function*, *verbose=True*)
Minimize a function using L-BFGS-B.

Parameters

- **function** (*object*) – Objective function. It has to implement the *optimix.function.Function* interface.
- **verbose** (*bool*) – `True` for verbose output; `False` otherwise.

optimix.optimize.**maximize**(*function*, *verbose=True*)
Maximize a function using L-BFGS-B.

Parameters

- **function** (*object*) – Objective function. It has to implement the *optimix.function.Function* interface.
- **verbose** (*bool*) – `True` for verbose output; `False` otherwise.

optimix.optimize.**minimize_scalar**(*function*, *verbose=True*)
Minimize a scalar function using Brent's method.

Parameters

- **function** (*object*) – Objective function. It has to implement the *optimix.function.Function* interface.
- **verbose** (*bool*) – `True` for verbose output; `False` otherwise.

optimix.optimize.**maximize_scalar**(*function*, *verbose=False*)
Maximize a scalar function using Brent's method.

Parameters

- **function** (*object*) – Objective function. It has to implement the *optimix.function.Function* interface.
- **verbose** (*bool*) – `True` for verbose output; `False` otherwise.

CHAPTER 7

Comments and bugs

You can get the source and open issues on Github.

Python Module Index

o

Index