nsopy

Release 2.0

Robin Vujanic

Dec 27, 2022

CONTENTS

nsopy is a Python library implementing a set of first order methods to solve non-smooth, constrained convex optimization models.

It is applicable to problems of the form

$$\begin{array}{ll} \min & f(x) \\ \text{s.t.} & x \in \mathbb{X} \end{array}$$

where:

- $f(x)$ is convex, but not necessarily differentiable
- $\mathbb{X} \subseteq \mathbb{R}^n$ is convex

CHAPTER

ONE

INSTALLATION

```
$ pip install nsopy
```

CHAPTER

TWO

EXAMPLE

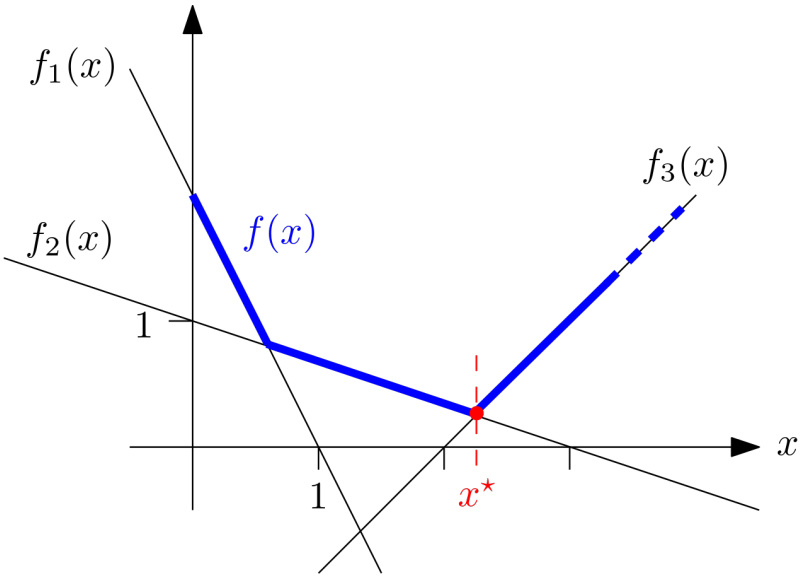
We seek to minimize the following piece-wise affine function:

$$\begin{array}{ll} \min\limits_{x} & \max_i f_i(x) \\ \text{s.t.} & x \geq 0 \end{array}$$

with

$$\begin{aligned} f_1(x) &= -2x + 2 \\ f_2(x) &= -\tfrac{1}{3}x + 1 \\ f_3(x) &= x - 2 \end{aligned}$$

We can visualize the problem and note that the solution is $x^\star = 2.25$.



Constraints: In the minimization problem we require that solutions satisfy $x \geq 0$. The blue color is used in the figure to indicate this. To enable **nsopy** to satisfy this, we need to supply it with a **projection function**: given a point x that does not necessarily satisfy $x \geq 0$, it returns the *closest* (in ℓ_2 sense) point that does.

For this example:

```
import numpy as np

def projection_function(x_k):
    return np.maximum(x_k, 0)
```

Note: A list of common projection functions can be found *here <https://github.com/robin-vjc/nsopy/blob/master/docs/img/simple_projections.png>*

We define the function to optimize:

```
def oracle(x_k):
    # evaluation of the f_i components at x_k
    fi_x_k = [-2*x_k + 2,  -1.0/3*x_k + 1,  x_k - 2]

    f_x_k = max(fi_x_k)  # function value at x_k

    diff_fi = [-2, -1.0/3.0, 1]  # gradients of the components
    max_i = fi_x_k.index(f_x_k)
    # subgradient at x_k is the gradient of the active function component; cast as (1x1␣
↪dimensional) np.array
    diff_f_xk = np.array([diff_fi[max_i], ])

    return 0, f_x_k, diff_f_xk
```

And solve:

```
from nsopy.methods.subgradient import SubgradientMethod
from nsopy.loggers import GenericMethodLogger

method = SubgradientMethod(oracle, projection_function, dimension=1, stepsize_0=0.1,␣
↪stepsize_rule='constant', sense='min')
logger = GenericMethodLogger(method)

for iteration in range(200):
    method.step()
```

Inspect the solution:

```
print(logger.x_k_iterates[-5:])
>>> [2.1999999999999904, 2.216666666666657, 2.2333333333333236, 2.2499999999999902, 2.
↪266666666666657]
```

Check out the *Usage* section for further information, including how to *Installation* the project.

Note: This project is under active development.

CHAPTER

THREE

CONTENTS

3.1 Usage

3.1.1 Installation

To use nsopy, first install it using pip:

```
(.venv) $ pip install lumache
```

3.1.2 Quick Start

To retrieve a list of random ingredients, you can use the `nsopy.loggers.GenericMethodLogger` function:

nsopy.utils.**invert_oracle_sense**(*oracle*)

The `kind` parameter should be either `"meat"`, `"fish"`, or `"veggies"`. Otherwise, `nsopy.loggers.GenericMethodLogger()` will raise an exception.

For example:

```
>>> import lumache
>>> lumache.get_random_ingredients()
['shells', 'gorgonzola', 'parsley']
```

3.2 API

nsopy.loggers

3.2.1 nsopy.loggers

Classes

DualDgmFgmMethodLogger(dual_method)	Additionally logs # of oracle calls, time
EnhancedDualMethodLogger(dual_method)	Additionally logs # of oracle calls, time
GenericDualMethodLogger(dual_method)	Works with all implemented dual nsopy, and logs only variables that are common across all of them (lambda_k, d_k, etc)
GenericMethodLogger(dual_method)	Works with all implemented dual nsopy, and logs only variables that are common across all of them (lambda_k, d_k, etc)
PGMVisualizationLogger(dual_method)	Additionally logs # of oracle calls, time
SlimDualMethodLogger(dual_method)	Don't store lambda_k nor x_k.
TemplateMethodLogger(template_method)	

PYTHON MODULE INDEX

INDEX