PyROL Documentation

Release 0.1.1

Chris Richardson and Florian Wechsung

Feb 18, 2019

Contents

Contents:

CHAPTER 1

Introduction

ROL (Rapid Optimization Library) is an optimization library written in C++ and is part of Trilinos. PyROL is a Python wrapper for ROL using the <https://github.com/pybind/pybind11> library. It allows you to create objectives, constraints and vectors all in python. For the vectors you can use your own underlying data storage.

This enables the use of ROL for large scale PDE Constrained Optimization with FEniCS or Firedrake. Have a look at the examples to see how to to solve the motherproblem of PDE Constrained Optimization with either of the two libraries.

CHAPTER 2

Installation

We assume that you have Trilinos 12.10 or later installed on your machine.

```
git clone https://bitbucket.org/pyrol/pyrol/
cd src
git clone https://github.com/pybind/pybind11
```

We use cmake for the configuration. Sometime there are issues with finding the correct python version. On macOS, try running

```
cmake -DPYTHON_LIBRARY=$(python-config --prefix)/lib/libpython2.7.dylib \-DPYTHON_
↪INCLUDE_DIR=$(python-config --prefix)/include/python2.7 -DTRILINOS_DIR="/path/to/
↪Trilinos/" ..
```

on Ubuntu run

```
cmake -DTRILINOS_DIR="/path/to/Trilinos/" -DPYTHON_EXECUTABLE:FILEPATH=/usr/bin/
↪python2.7 -DPYTHON_INCLUDE_DIR:PATH=/usr/include/python2.7 -DPYTHON_
↪LIBRARY:FILEPATH=/usr/lib/x86_64-linux-gnu/libpython2.7.so ..
```

CHAPTER 3

API

CHAPTER 4

Vectors

To write your own vector class, simply inherit from *ROL.CustomLA*.

PyROL provides a *ROL.StdVector* class, which is based on the *ROL::StdVector* class. Using it is as simple as

```
import ROL
x = ROL.StdVector(2)
x[0] = 1.0
x[1] = 2.0
print x.norm()
```

However, you are likely to want to implement your own vector class, for example based on the data storage of a finite element library. This class is then able to implement a custom inner product, for example an L^2 inner product.

TODO: document one of the CustomLA classes and somehow include them here.

CHAPTER 5

Objectives

To write your own objective class, simply inherit from *ROL.Objective*.

Let's say we want to minimize the function $f(x, y) = (x - 1)^2 + y^2$.

```
class MyObj(ROL.Objective):
    def __init__(self):
        ROL.Objective.__init__(self)

    def value(self, x, tol):
        return (x[0] - 1)**2 + x[1]**2

    def gradient(self, g, x, tol):
        g[0] = 2 * (x[0] - 1)
        g[1] = 2 * x[1]
```

If we omit the definition of the *gradient* function, then ROL will use a finite difference approximation instead.

CHAPTER 6

Constraints

ROL supports three different types of constraints.

6.1 Bound Constraints

Creating a bound constraint is a simple as

```
x_lo = ROL.StdVector(2)
x_lo[0] = -1.0
x_lo[1] = -1.0
x_up = ROL.StdVector(2)
x_up[0] = +1.0
x_up[1] = +1.0
scale = 1.0
bnd = ROL.BoundConstraint(x_lo,x_up, scale)
```

Note that in order to use this with your own vector class, the class needs to implement the *__getitem__* method.

6.2 Equality Constraints

In order to write your own equality constraint, inherit from the class *ROL.EqualityConstraint*

6.3 Inequality Constraints

Not yet implemented in PyROL.

CHAPTER 7

Indices and tables

- genindex
- modindex
- search