ProxPython User Manual

Stefan Ziehe

December 18, 2015

Contents

Chapter 1

Introduction

ProxPython is a port of the ProxToolbox 2.0 [1] to the Python programming language. It is a collection of interfaces and classes for solving mathematical problems using fixed point iterations with proximal operators. The primary motivation for the port was to make the Proxtoolbox easier to maintain and extend by using a modern and proven programming language and easily available tools and libraries.

[1] https://num.math.uni-goettingen.de/~r.luke/publications/publications.html

Chapter 2

ProxPython Basics

ProxPython is contained in a Python module called *proxtoolbox*. The interfaces and classes are distributed across different submodules. The following interfaces exist:

Problem
: Represents a mathematical problem. Responsible for getting input data, running an algorithm and processing the results.

Algorithm
: Represents a fixed point algorithm. Contains some ProxOperators which are used in each iteration.

ProxOperator
: Represents a proximal operator. Can be applied to data.

For details about the interface methods and a list of available classes see the API-reference.

2.1 Problem

The *Problem* interface is the linchpin of ProxPython. It is responsible for gathering input data, launching the computation and processing the result. For that purpose it has three class methods: *_presolve* (gather input data), *_solve* (run algorithm) and *_postsolve* (process output). These methods can be overloaded with the desired behaviour. However, they are usually not intended to be called directly. Instead use the *solve* method, which takes care of that.
The constructor of the *Problem* interface takes a Python dictionary as the only parameter. The dictionary contains key-value pairs that make up the configuration of the problem. When using other ProxPython interfaces within the problem, add their configuration to the dictionary and pass it to their constructors.

2.2 Algorithm

The *Algorithm* interface is used to represent an iterative fixed point algorithm. When starting one, it is required to specify a tolerance and a maximum number of iterations. Other options (e.g. the used *ProxOperators*) are defined in a configuration dictionary.

2.3 ProxOperator

The *ProxOperator* interface represents a proximal operator, which is usually a projection of a point onto a convex set. Additional options can again be specified in a configuration dictionary.

Chapter 3

Sudoku

Solving Sudokus is an unexpected but interesting application for the ProxToolbox. It also serves as a good entry point for learning about the inner workings of this software. So if you want to create your own ProxToolbox problems, looking at the *sudoku* module source code is a good way to start.

3.1 Using the Sudoku class

First we take a look at the possible entries in the configuration dictionary:

algorithm Default: RAAR
The algorithm to be used. RAAR and HPR will work.

proj1 Default: P_diag
The first ProxOperator to be used. Required by both RAAR and HPR.

proj2 Default: P_parallel
The second ProxOperator to be used. Required by RAAR.

projectors Default: (ProjRow,ProjColumn,ProjSquare,ProjGiven)
A sequence of ProxOperators. Required by P_parallel.

beta0 Default: 1
Relaxation parameter required by RAAR/HPR.

beta_max Default: 1
Relaxation parameter required by RAAR/HPR.

beta_switch Default: 1
Relaxation parameter required by RAAR/HPR.

maxiter Default: 2000
Maximum number of iterations for the algorithm.

tol Default: $1e-9$
Tolerance for the algorithm.

given_sudoku Default: A pre-calculated Sudoku
A 9x9 NumPy array containing a Sudoku. Empty cells are represented by the number 0.

Now we take a look at a small Python snippet that solves a Sudoku with a non-default configuration:

```
import numpy as np
from proxtoolbox import Sudoku
from proxtoolbox.algorithms import HPR

config = Sudoku.default_config.copy();
config['algorithm'] = HPR;
config['maxiter'] = 3000;
config['given_sudoku'] = np.array(((2,0,0,0,0,1,0,3,0),
                                   (4,0,0,0,8,6,1,0,0),
                                   (0,0,0,0,0,0,0,0,0),
                                   (0,0,0,0,1,0,0,0,0),
                                   (0,0,0,0,0,0,9,0,0),
                                   (0,0,5,0,0,3,0,0,7),
                                   (0,0,0,0,0,0,0,0,0),
                                   (1,0,0,0,0,7,4,9,0),
                                   (0,2,4,1,0,0,0,0,0)));

sudoku = Sudoku(config);
sudoku.solve();
```

Chapter 4

Ptychography

The ProxToolbox includes the possibility to solve diffraction-pattern phase problems using the Ptychography technique. The included default example is the reconstruction of a photo of the Gänseliesel fountain in Göttingen.

4.1 Using the Ptychography class

First, we review the most important values in the configuration dictionary for configuring the Gänseliesel example:

algorithm Default: PALM
: The algorithm used to solve the problem. Requires an algorithm from the *Ptychography* module.

maxiter Default: 300
: Maximum number of iterations.

tol Default: 0.625
: Tolerance used by the algorithm.

warmup Default: True
: Specifies whether to do a warmup first.

warmup_maxiter Default: 20
: Maximum number of warmup iterations.

warmup_alg Default: PALM
: The algorithm used for warmup. Requires an algorithm from the *Ptychography* module.

probe_guess_type Default: 'circle'
: The initial guess for the probe. Possible values are: 'exact', 'exact_amp', 'circle', 'robin_initial' and 'ones'.

object_guess _type Default: 'random'
Initial guess for the object. Possible values are: 'exact', 'ones', 'constant', 'exact_perturbed' and 'random'.

ignore_error Default: False
Specifies whether to omit the calculation of error statistics. This might result in significant performance gains but makes the results less meaningful.

ptychography_prox Default: 'Rodenburg'
Specifies which set of ProxOperators to use. Possible values are: 'PALM', 'PALMRegPhiPtwise', 'Rodenburg', 'Thibault' and 'Thibault_AP'.

rodenburg_inner_it Default: 1
Number of iterations inside the 'Rodenburg' ProxOperators.

beta0, beta_max Default: 1
Regularization parameters for RAAR-like algorithms.

beta_switch Default: 30
Regularization parameter for RAAR-like algorithms.

overrelax Deafult: 1
Regularization parameter for several ProxOperators. Higher values result in slower convergence but better stability.

4.2 Blocking

Ptychography problems usually can be easily parallelized by using blocking. The input data is partitioned into a number of blocks, which can be processed independently. After finishing, the intermediate results are merged.
By default the ProxToolbox includes an algorithm called $BlockingMeta$, which applies blocking schemes independently of the algorithm.
When configuring a Ptychography problem to utilize blocking, the following configuration entries are relevant:

blocking_switch Default: False
Enable or disable blocking.

blocking_scheme Default: 'divide'
Specifies how to partition the data.

block_cols, block_rows Default: 2
Specifies how many blocks to create.

block_maxiter Default: 1
Specifies how many iterations to apply to each block during every iteration of BlockingMeta. Possible values are: 'one', 'single_view', 'divide', 'distribute'.

between_blocks_scheme Default: 'sequential'
Specifies whether to parallelize processing the blocks. Possible values are: 'sequential', 'parallel'.

Keep in mind that Python's *multiprocessing* module is used for parallelization, so there is a rather large overhead when processing too many blocks in parallel.

4.3 Custom Ptychography problems

Creating custom Ptychography problems mostly is about getting the data into the right shape. This is done in the $_{g}et_{p}tychography_{d}ata$ method of the *Ptychography* class [1]. There are also some configuration entries that need to be adapted:

Nx, Ny Default: 64, 64
Product space dimensions.

Nz Default: 1
Product space dimension.

nx, ny Default: 25
Probe dimensions.

Here is an example of a custom Ptychography problem class:

```
class Ptychography_NTT_01_26210(Ptychography):

    default_config = {
        'Nx':192,
        'Ny':192,
        'Nz':1,
        'scan_stepsize':3.5e-7,
        'nx':26,
        'ny':26,
        'sample_area_center':(0.5,0.5),
        'noise':None,
        'switch_probemask':False,
        'probe_mask_gamma':1,
        'rmsfraction':0.5,
        'scan_type':'raster',
        'switch_object_support_constraint':False,
        'probe_guess_type':'robin_initial',
        'object_guess_type':'constant',
        'warmup':True,
        'warmup_alg':PALM,
        'warmup_maxiter':0,
        'algorithm':PALM,
```

[1] See the API reference for more information

```
        'maxiter':300,
        'tol':-1,
        'ignore_error':False,
        'ptychography_prox':'Rodenburg',
        'blocking_switch':False,
        'blocking_scheme':'distribute',
        'between_blocks_scheme':'averaged',
        'within_blocks_scheme':'none',
        'block_rows':2,
        'block_cols':2,
        'block_maxiter':1,
        'rodenburg_inner_it':1,
        'beta0':1,
        'beta_max':1,
        'beta_switch':30,
        'bs_mask':None,
        'bs_factor':1,
        'overrelax':1
    };

    def __init__(self,config=default_config):
        Ptychography.__init__(self,config);

    def _get_ptychography_data(self):
        data = sp.io.loadmat('data_NTT_01_26210_192x192.mat');
        d1y = data['d1y'][0][0];
        d1x = data['d1x'][0][0];
        i_exp = data['I_exp'];
        l = data['lambda'][0][0];
        z01 = data['z01'][0][0];
        self.config['fmask'] = data['fmask'].astype(np.float);
        data = sp.io.loadmat('reconstruction_data_NTT_01_26210_192x192.mat');
        probe = data['Probe'];
        sample_plane = data['object'];

        data = sp.io.loadmat('positions_NTT_01_26210.mat')
        positions = data['positions'].astype(np.float);
        positions[:,0] /= d1y;
        positions[:,0] -= np.amin(positions[:,0]);
        positions[:,1] /= d1x;
        positions[:,1] -= np.amin(positions[:,1]);
        # numpy.round is weird
        positions[np.modf(positions)[0] == 0.5] += 0.01;
        positions = np.round(positions).astype(np.int);

        self.config['trans_max_true'] = 1.0;
        self.config['trans_min_true'] = 0.0;

        return d1x,d1y,l,z01,positions,probe,sample_plane,i_exp;
```