skprocrustes Documentation

Release 0.1

Melissa Weber Mendonça

Jul 12, 2017

CONTENTS

Collection of solvers for the (Weighted) Orthogonal Procrustes Problem.

$$\min \|AXC - B\|_F^2 \qquad s.t. \quad X^T X = I$$

where $A_{m\times n}, B_{m\times q}, C_{p\times q}, X_{n\times p}$. Usually n >> p, which means we can solve unbalanced problems.

CHAPTER

ONE

AVAILABLE SOLVERS

- `SPGSolver` Nonmonotone Spectral Projected Gradient Method for the (unbalanced) WOPP, as described in *[FB12]*.
- `GKBSolver` Nonmonotone Spectral Projected Gradient Method using incomplete Lanczos (Golub-Kahan) Bidiagonalization, as described in *[FBM17]*.
- `EBSolver` Expansion-Balance method, as described in *[tBK84]*.
- `GPISolver` Generalized Power Iteration for the WOPP, as described in *[NZL17]*.

CHAPTER TWO

USAGE

To use the package to solve a given problem with predefined matrices A, B and C using the SPG solver, for example, use

```
>>> import skprocrustes as skp
>>> problem = skp.ProcrustesProblem((m,n,p,q),      # tuple
                                    matrices=[A, B, C])
>>> mysolver = skp.SPGSolver(**kwargs)
>>> result = mysolver.solve(problem)
```

where ***kwargs* are the selected solver's options (see the Module Reference for more details).

To use the package to solve one of the three predefined problems (as described in *[ZD06]*), using the GKB solver, for example, use

```
>>> import skprocrustes as skp
>>> problem = skp.ProcrustesProblem((m,n,p,q),      # tuple
                                    problemnumber=1)
>>> mysolver = skp.GKBSolver(**kwargs)
>>> result = mysolver.solve(problem)
```

CHAPTER THREE

REFERENCES

CHAPTER

FOUR

INSTALLATION

4.1 Quick Installation

In the root directory of the package, just do:

```
python setup.py install
```

4.2 Latest Software

The latest software can be downloaded from GitHub

4.3 Installation Dependencies

`scikit-procrustes` requires the following software packages to be installed:

- Python 3.6.1 or later.
- NumPy 1.13.0 or later.
- SciPy 0.19.0 or later.
- Matplotlib 2.0.2 or later.

CHAPTER

FIVE

CONTENTS

5.1 skprocrustes package

5.1.1 Module contents

class `skprocrustes.`**`ProcrustesProblem`**(*sizes*, *problemnumber=None*, *matrices=[]*)

Bases: `object`

The problem we want to solve.

Usage example (default problem):

```
>>> import skprocrustes as skp
>>> problem = skp.ProcrustesProblem((10,10,2,2), problemnumber=1)
```

Usage example (user defined problem):

```
>>> import skprocrustes as skp
>>> import numpy as np
>>> A = ... # given by the user
>>> B = ... # given by the user
>>> C = ... # given by the user
>>> X = ... # given by the user (optional)
>>> problem = skp.ProcrustesProblem((m,n,p,q), matrices=(A,B,C,X))
```

Input Parameters:

`sizes:` **tuple** `(m,n,p,q)`, where $A_{m\times n}, B_{m\times q}, C_{p\times q}$ and $X_{n\times p}$.

(optional) **`problemnumber:`** **int** Can be `1`, `2` or `3`, and selects one of the predefined problems as described in reference *[ZD06]*. (for more details, see the documentation for `_setproblem`)

(optional) **`matrices:`** **list of ndarrays** If present, must contain a list of three or four matrices corresponding to A, B, C, and optionally X (known solution) with adequate sizes.

Note: Currently, m must be equal do n, and p must be equal do q. This is the case for all three solvers. (However, n can be greater than p)

Note: If `matrices` is not given by the user, `problemnumber` (1, 2 or 3) must be selected so that one of the default problems is built.

Attributes:

The problem matrices (generated or given) are accessible via

```
>>> problem.A
>>> problem.B
>>> problem.C
>>> problem.Xsol
```

_setproblem(*matrices*, *problemnumber*)

Method to effectively build A, B, and C if they are not already given.

This method should not be called directly; it is called by the ProcrustesProblem constructor.

Available problems are all based on reference *[ZD06]*: All problems have the form

$$A = U\Sigma V^T$$

where Σ varies between problems, and

$$\begin{aligned} U &= I_{m\times m} - 2uu^T \\ V &= I_{n\times n} - 2vv^T \end{aligned}$$

where u and v are randomly generated using `np.random.randn` (normal distribution) and then normalized.

C can be built, but for our predefined problems it is always the identity matrix.

problemnumber = 1: Well conditioned problem: the singular values are randomly and uniformly distributed in the interval [10,12].

problemnumber = 2: For this problem, the singular values are

$$\sigma_i = 1 + \frac{99(i-1)}{(m-1)} + 2r_i$$

and r_i are random numbers chosen from a uniform distribution on the interval [0,1].

problemnumber = 3: For this problem, the singular values are

$$\sigma_i = \begin{cases} 10 + r, & 1 \le i \le m_1 \\ 5 + r, & m_1 + 1 \le i \le m_2 \\ 2 + r, & m_2 + 1 \le i \le m_3 \\ \frac{r}{1000}, & m_3 + 1 \le i \le m \end{cases}$$

Thus, A has several small singular values and is ill-conditioned.

Note: `problemnumber = 3` can only be used if $n = 50$, $n = 95$, $n = 500$ or $n = 1000$.

class skprocrustes.**OptimizeResult**

Bases: `dict`

Represents the optimization result. (*based on scipy.optimize.OptimizeResult*)

This class is constructed as a dictionary of parameters defined by the creation of the instance. Thus, its attributes may vary.

Possible attributes:

- **success** [`bool`] Whether or not the optimizer exited successfully.

- **`status`** [`int`] Termination status of the optimizer. Its value depends on the underlying solver. Refer to *message* for details.
- **`message`** [`str`] Description of the cause of the termination.
- **`fun`** [`float`] Value of the objective function at the solution.
- **`normgrad`** [`float`] Value of the norm of the gradient at the solution.
- **`nbiter`** [`int`] Number of iterations performed by the optimizer.
- **`nfev`** [`int/float`] Number of evaluations of the objective function (if called by GKBSolver, nfev is a float representing the proportional number of calls to the objective function at each block step).
- **`blocksteps`** [`int`] Number of blocksteps performed (if called by GKBSolver)
- **`total_fun`: list** List of objective function values for each iteration performed (used to report and compare algorithms). Only if `full_results` is True.
- **`total_grad`: list** List of gradient norm values for each iteration performed (used to report and compare algorithms). Only if `full_results` is True, and only for SPGSolver and GKBSolver.
- **`total_crit`: list** List of criticality measure values for each iteration performed (used to report and compare algorithms). Only if `full_results` is True, and only for EBSolver and GPISolver.

Notes: There may be additional attributes not listed above depending of the specific solver. Since this class is essentially a subclass of dict with attribute accessors, one can see which attributes are available using the *keys()* method.

class `skprocrustes.`**`ProcrustesSolver`**(**args*, ***kwargs*)

Bases: `object`

Abstract class to implement a solver for the ProcrustesProblem.

All subclasses should implement the following methods:

`_setoptions`(**args*, ***kwargs*)

Choose which options are valid and applicable to this solver.

`solve`(**args*, ***kwargs*)

Call a solver function and set up the `OptimizeResult` instance with the result and statistics as convenient for this solver. Should be something like this:

```
output = somesolver(problem, *args, **kwargs)
result = OptimizeResult(output)
return result
```

class `skprocrustes.`**`SPGSolver`**(***kwargs*)

Bases: `skprocrustes.skprocrustes.ProcrustesSolver`

Subclass containing the call to the `spectral_setup()` function corresponding to the Spectral Projected Gradient solver described in *[FB12]* and *[FBM17]*.

Usage example:

```
>>> mysolver = skp.SPGSolver(verbose=3)
>>> result = mysolver.solve(problem)
```

Input:

`key = value`: keyword arguments available

- **`full_results`: (*default*: `False`)** Return list of criticality values at each iteration (for later comparison between solvers)
- **`strategy`: (*default*: `"newfw"`)**
  - **`"monotone"`:** monotone trust region

    - **"bazfr"** [] nonmonotone method according to *[FB12]*
    - **"newfw"** [] nonmonotone method according to *[FBM17]*
- **gtol: (*default*: 1e-3)** tolerance for detecting convergence on the gradient
- **maxiter: (*default*: 2000)** maximum number of iterations allowed
- **verbose: (*default*: 1)** verbosity level. Current options: - 0: only convergence info - 1: only show time and final stats - 2: show outer iterations - 3: everything (except debug which is set separately)
- **changevar: (*default*: False)** boolean option to allow for a change of variables before starting the method. Currently disabled due to bad performance.

Output:

`solver`: `ProcrustesSolver` instance

_setoptions (*options*)

Sets and validates options for the SPGSolver.

This method should not be called directly; it is called by the SPGSolver constructor.

solve (*problem*)

Effectively solve the problem using the SPG method.

Input:

`problem`: `ProcrustesProblem` instance

Output:

`result`: `OptimizationResult` instance

class skprocrustes.**GKBSolver** (***kwargs*)

Bases: `skprocrustes.skprocrustes.SPGSolver`

Subclass containing the call to the `spectral_setup()` function corresponding to the Spectral Projected Gradient Method using incomplete Golub-Kahan Bidiagonalization (Lanczos) as described in *[FBM17]*. This class extends the `SPGSolver` class, with some variation in the input and output parameters.

Usage example:

```
>>> mysolver = skp.GKBSolver(verbose=3)
>>> result = mysolver.solve(problem)
```

Input:

key = value: keyword arguments available

- **full_results: (*default*: False)** Return list of criticality values at each iteration (for later comparison between solvers)
- **strategy: (*default*: "newfw")**
    - **"monotone":** monotone trust region
    - **"bazfr"** [] nonmonotone method according to *[FB12]*
    - **"newfw"** [] nonmonotone method according to *[FBM17]*
- **gtol: (*default*: 1e-3)** tolerance for detecting convergence on the gradient
- **maxiter: (*default*: 2000)** maximum number of iterations allowed
- **verbose: (*default*: 1)** verbosity level. Current options: - 0: only convergence info - 1: only show time and final stats - 2: show outer iterations - 3: everything (except debug which is set separately)
- **changevar: (*default*: False)** boolean option to allow for a change of variables before starting the method. Currently disabled due to bad performance.

Output:

`solver`: `ProcrustesSolver` instance

Note: Since this subclass extends SPGSolver class, we use `SPGSolver._setoptions` directly.

---

solve (*problem*)

Effectively solve the problem using the GKB method.

Input:

`problem`: `ProcrustesProblem` instance

Output:

`result`: `OptimizationResult` instance

class skprocrustes.**EBSolver** (***kwargs*)

Bases: `skprocrustes.skprocrustes.ProcrustesSolver`

Subclass containing the call to the `eb_solver()` function corresponding to the Expansion-Balance method as described in *[tBK84]*.

Usage example:

```
>>> mysolver = skp.EBSolver(verbose=3)
>>> result = mysolver.solve(problem)
```

Input:

key = value: keyword arguments available

- **full_results: (*default*: False)** Return list of criticality values at each iteration (for later comparison between solvers)
- **tol: (*default*: 1e-6)** tolerance for detecting convergence
- **maxiter: (*default*: 2000)** maximum number of iterations allowed
- **verbose: (*default*: 1)** verbosity level. Current options: - `0`: only convergence info - `1`: only show time and final stats

Output:

`solver`: `ProcrustesSolver` instance

_setoptions (*options*)

Sets and validates options for the EBSolver.

This method should not be called directly; it is called by the EBSolver constructor.

solve (*problem*)

Effectively solve the problem using the Expansion-Balance method.

Input:

`problem`: `ProcrustesProblem` instance

Output:

`result`: `OptimizationResult` instance

class skprocrustes.**GPISolver** (***kwargs*)

Bases: `skprocrustes.skprocrustes.ProcrustesSolver`

Subclass containing the call to the `gpi_solver()` function corresponding to the Generalized Power Iteration method as described in *[NZL17]*.

Usage example:

```
>>> mysolver = skp.GPISolver(verbose=3)
>>> result = mysolver.solve(problem)
```

Input:

`key = value`: keyword arguments available

- **`full_results`: (*default*: `False`)** Return list of criticality values at each iteration (for later comparison between solvers)
- **`tol`: (*default*: `1e-3`)** tolerance for detecting convergence
- **`maxiter`: (*default*: `2000`)** maximum number of iterations allowed
- **`verbose`: (*default*: `1`)** verbosity level. Current options: - `0`: only convergence info - `1`: only show time and final stats

Output:

`solver`: `ProcrustesSolver` instance

`_setoptions`(*options*)
Sets and validates options for the GPISolver.

This method should not be called directly; it is called by the GPISolver constructor.

`solve`(*problem*)
Effectively solve the problem using the Generalized Power Iteration method.

Input:

`problem`: `ProcrustesProblem` instance

Output:

`result`: `OptimizationResult` instance

`skprocrustes.`**`spectral_solver`**(*problem*, *largedim*, *smalldim*, *X*, *A*, *B*, *solvername*, *options*)
Nonmonotone Spectral Projected Gradient solver for problems of the type

$$\min \|AXC - B\|_F^2 \qquad s.t. X^T X = I$$

The method is described in references *[FB12]* and *[FBM17]*, and we implement a few variations (including a monotone version, a nonmonotone version using the strategy described in *[FB12]*, and a nonmonotone version using the strategy described in *[FBM17]*; check below for more details on how to select these different algorithms).

This function is called by `spectral_solver` from both GKBSolver and SPGSolver, with different parameters.

Input:

- `problem`: `ProcrustesProblem` instance
- `largedim`: `int`
- **`smalldim`: `int`** Since this function is called by `spectral_solver`, it is possible we are solving a smaller version of the original problem (when using GKBSolver, for instance). Thus, `lagedim` and `smalldim` are the dimensions of the current problem being solved by `spectral_solver`.
- **`X`: `ndarray(smalldim, p)`** Initial guess for the solution X of the Procrustes Problem being solved.
- `A`: `ndarray(largedim, smalldim)`
- `B`: `ndarray(largedim, q)`

- **`solvername`: str** Takes values `spg` or `gkb` (used to decide if `full_results` can be reported).
- **`options`: `dict`**

  Solver options. Keys available are:

  - **`maxiter`: `int`** Maximum number of iterations allowed
  - **`strategy`: `str`** `monot` (Monotone strategy), `bazfr` (Nonmonotone strategy described in *[FB12]*) or `newfw` (Nonmonotone strategy described in *[FBM17]*)
  - **`verbose`: `int`** Can take values in (0,1,2,3)
  - **`gtol`: `float`** Tolerance for convergence.

Output:

- **`exitcode`: `int`** 0 (success) or 1 (failure)
- **`f`: `float`** Value of the objective function at final iterate
- **`X`: `ndarray(smalldim, p)`** Approximate solution (final iterate)
- **`normg`: `float`** Criticality measure at final iterate
- **`outer`: `int`** Final number of outer iterations performed.

`skprocrustes.`**`eb_solver`**(*problem*, *options*)

Expansion-Balance solver

Here we consider always $m = n$, $p = q$, $C = I$. Thus the problem has to be

$$\min \|A_{n\times n}X_{n\times p} - B_{n\times p}\|_F^2 \qquad s.t. X^TX = I_{p\times p}$$

References: *[ZD06]* and *[tBK84]*.

`skprocrustes.`**`gpi_solver`**(*problem*, *options*)

Generalized Power Iteration solver

Here we consider always C=I. Thus the problem has to be

$$\min \|A_{m\times n}X_{n\times p} - B_{m\times p}\|_F^2 \qquad s.t. X^TX = I_{p\times p}$$

References: *[NZL17]*

5.2 BSD 2-Clause License

Copyright (c) 2017, Melissa Weber Mendonça All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS “AS IS” AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED.

IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

5.3 Authors

Melissa Weber Mendonça

Department of Mathematics

Federal University of Santa Catarina (UFSC)

Florianópolis, SC - Brasil

CHAPTER

SIX

INDICES AND TABLES

- genindex
- modindex
- search

BIBLIOGRAPHY

[FB12] J. B. Francisco and F. S. Viloche Bazán. Nonmonotone algorithm for minimization on closed sets with applications to minimization on stiefel manifolds. *J. Comp. Appl. Math.*, 236(10):2717–2727, 2012. doi:10.1016/j.cam.2012.01.014.

[FBM17] J. B. Francisco, F. S. Viloche Bazán, and M. Weber Mendonça. Non-monotone algorithm for minimization on arbitrary domains with applications to large-scale orthogonal procrustes problem. *Appl. Num. Math.*, 112:51–64, 2017. doi:10.1016/j.apnum.2016.09.018.

[NZL17] F. Nie, R. Zhang, and X. Li. A generalized power iteration method for solving quadratic problem on the stiefel manifold. *Sci. China Inf. Sci.*, 60:112101:1–112101:10, 2017. doi:10.1007/s11432-016-9021-9.

[tBK84] J. M. F. ten Berge and D. L. Knol. Orthogonal rotations to maximal agreement for two or more matrices of different column orders. *Psychometrika*, 49:49–55, 1984. doi:10.1007/BF02294205.

[ZD06] Z. Zhang and K. Du. Successive projection method for solving the unbalanced procrustes problem. *Sci. China Ser. A*, 49:971–986, 2006. doi:10.1007/s11425-006-0971-2.

PYTHON MODULE INDEX

INDEX