SPGL1

Oct 05, 2021

Contents

SPGL1 is a solver for large-scale one-norm regularized least squares.

It is designed to solve any of the following three problems:

1. Basis pursuit denoise (BPDN):

$$min \quad ||\mathbf{x}||_1 \quad subj.to \quad ||\mathbf{Ax}-\mathbf{b}||_2 <= \sigma$$

2. Basis pursuit (BP):

$$min \quad ||\mathbf{x}||_1 \quad subj.to \quad \mathbf{Ax}=\mathbf{b}$$

3. Lasso:

$$min \quad ||\mathbf{Ax}-\mathbf{b}||_2 \quad subj.to \quad ||\mathbf{x}||_1 <= \tau$$

The matrix $\mathbf{A}$ can be defined explicitly, or as a `scipy.sparse.linalg.LinearOperator` that returns both both $\mathbf{Ax}$ and $\mathbf{A}^H\mathbf{b}$.

SPGL1 can solve these three problems in both the real and complex domains.

CHAPTER 1

References

The algorithm implemented by SPGL1 is described in these two papers:

- E. van den Berg and M. P. Friedlander, *Probing the Pareto frontier for basis pursuit solutions*, SIAM J. on Scientific Computing, 31(2):890-912, November 2008
- E. van den Berg and M. P. Friedlander, *Sparse optimization with least-squares constraints*, Tech. Rep. TR-2010-02, Dept of Computer Science, Univ of British Columbia, January 2010

CHAPTER 2

History

SPGL1 has been initially implemented in MATLAB by E. van den Berg and M. P. Friedlander. This project is aimed at porting of their algorithm in Python. Small modifications are implemented in some areas of the code where more appropriate implementation choices were identified for the Python programming language.

2.1 Installation

Python 3.5 or greater is required. This package may also work for Python 2.7 or greater, however we do not provide any guarantee neither we will make any effort to maintain back compatibility with Python 2.

2.1.1 From PyPI

To install `spgl1` within your current environment, simply type:

```
>> pip install spgl1
```

2.1.2 From source

First of all clone the repository. To install `spgl1` within your current environment, simply type:

```
>> make install
```

or as a developer:

```
>> make dev-install
```

To install `spgl1` in a new conda environment, type:

```
>> make install_conda
```

or as a developer:

```
>> make dev-install_conda
```

2.2 SPGL1 API

2.2.1 Main Solver

spgl1(A, b[, tau, sigma, x0, fid, . . .])	SPGL1 solver.

spgl1.spgl1

spgl1.**spgl1** (*A, b, tau=0, sigma=0, x0=None, fid=None, verbosity=0, iter_lim=None, n_prev_vals=3, bp_tol=1e-06, ls_tol=1e-06, opt_tol=0.0001, dec_tol=0.0001, step_min=1e-16, step_max=100000.0, active_set_niters=inf, subspace_min=False, iscomplex=False, max_matvec=inf, weights=None, project=<function _norm_l1_project>, primal_norm=<function _norm_l1_primal>, dual_norm=<function _norm_l1_dual>*)

SPGL1 solver.

Solve basis pursuit (BP), basis pursuit denoise (BPDN), or LASSO problems [1] [2] depending on the choice of `tau` and `sigma`:

```
(BP)      minimize  ||x||_1  subj. to  Ax = b

(BPDN)    minimize  ||x||_1  subj. to  ||Ax-b||_2 <= sigma

(LASSO)   minimize  ||Ax-b||_2  subj, to  ||x||_1 <= tau
```

The matrix `A` may be square or rectangular (over-determined or under-determined), and may have any rank.

Parameters

A [{sparse matrix, ndarray, LinearOperator}] Representation of an m-by-n matrix. It is required that the linear operator can produce `Ax` and `A^T x`.

b [array_like, shape (m,)] Right-hand side vector `b`.

tau [float, optional] LASSO threshold. If different from `None`, spgl1 solves LASSO problem

sigma [float, optional] BPDN threshold. If different from `None`, spgl1 solves BPDN problem

x0 [array_like, shape (n,), optional] Initial guess of x, if None zeros are used.

fid [file, optional] File ID to direct log output, if None print on screen.

verbosity [int, optional] 0=quiet, 1=some output, 2=more output.

iter_lim [int, optional] Max. number of iterations (default if `10*m`).

n_prev_vals [int, optional] Line-search history lenght.

bp_tol [float, optional] Tolerance for identifying a basis pursuit solution.

ls_tol [float, optional] Tolerance for least-squares solution. Iterations are stopped when the ratio between the dual norm of the gradient and the L2 norm of the residual becomes smaller or equal to `ls_tol`.

opt_tol [float, optional] Optimality tolerance. More specifically, when using basis pursuit denoise, the optimility condition is met when the absolute difference between the L2 norm of the residual and the `sigma` is smaller than `opt_tol`.

dec_tol [float, optional] Required relative change in primal objective for Newton. Larger `decTol` means more frequent Newton updates.

step_min [float, optional] Minimum spectral step.

step_max [float, optional] Maximum spectral step.

active_set_niters [float, optional] Maximum number of iterations where no change in support is tolerated. Exit with EXIT_ACTIVE_SET if no change is observed for `activeSetIt` iterations

subspace_min [bool, optional] Subspace minimization (`True`) or not (`False`)

iscomplex [bool, optional] Problem with complex variables (`True`) or not (`False`)

max_matvec [int, optional] Maximum matrix-vector multiplies allowed

weights [{float, ndarray}, optional] Weights `W` in `||Wx||_1`

project [func, optional] Projection function

primal_norm [func, optional] Primal norm evaluation fun

dual_norm [func, optional] Dual norm eval function

Returns

x [array_like, shape (n,)] Inverted model

r [array_like, shape (m,)] Final residual

g [array_like, shape (h,)] Final gradient

info [dict] Dictionary with the following information:

`tau`, final value of tau (see sigma above)

`rnorm`, two-norm of the optimal residual

`rgap`, relative duality gap (an optimality measure)

`gnorm`, Lagrange multiplier of (LASSO)

`stat`, `1`: found a BPDN solution, `2`: found a BP solution; exit based on small gradient, `3`: found a BP solution; exit based on small residual, `4`: found a LASSO solution, `5`: error: too many iterations, `6`: error: linesearch failed, `7`: error: found suboptimal BP solution, `8`: error: too many matrix-vector products

`niters`, number of iterations

`nProdA`, number of multiplications with A

`nProdAt`, number of multiplications with A'

`n_newton`, number of Newton steps

`time_project`, projection time (seconds)

`time_matprod`, matrix-vector multiplications time (seconds)

`time_total`, total solution time (seconds)

`niters_lsqr`, number of lsqr iterations (if `subspace_min=True`)

`xnorm1`, L1-norm model solution history through iterations `rnorm2`, L2-norm residual history through iterations

`lambdaa`, Lagrange multiplier history through iterations

References

[1], [2]

Examples using `spgl1.spgl1`

- sphx_glr_tutorials_spgl1.py

2.2.2 Other Solvers

oneprojector(b, d, tau)	One projector.
spg_bp(A, b, **kwargs)	Basis pursuit (BP) problem.
spg_bpdn(A, b, sigma, **kwargs)	Basis pursuit denoise (BPDN) problem.
spg_lasso(A, b, tau, **kwargs)	LASSO problem.
spg_mmv(A, B[, sigma])	MMV problem.

spgl1.oneprojector

spgl1.**oneprojector**(*b*, *d*, *tau*)

One projector.

Projects b onto the (weighted) one-norm ball of radius tau. If d=1 solves the problem:

```
minimize_x  ||b-x||_2  subject to  ||x||_1 <= tau.
```

else:

```
minimize_x  ||b-x||_2  subject to  ||Dx||_1 <= tau.
```

Parameters

b [ndarray] Input vector to be projected.

d [{ndarray, float}] Weight vector (or scalar)

tau [float] Radius of one-norm ball.

Returns

x [array_like] Projected vector

spgl1.spg_bp

spgl1.**spg_bp**(*A*, *b*, ***kwargs*)

Basis pursuit (BP) problem.

`spg_bp` is designed to solve the basis pursuit problem:

```
(BP)  minimize  ||x||_1  subject to  Ax = b,
```

where A is an M-by-N matrix, b is an M-vector. A can be an explicit M-by-N matrix or a scipy.sparse.linalg.LinearOperator.

This is equivalent to calling ‘‘spgl1(A, b, tau=0, sigma=0)

Parameters

A [{sparse matrix, ndarray, LinearOperator}] Representation of an m-by-n matrix. It is required that the linear operator can produce Ax and A^T x.

b [array_like, shape (m,)] Right-hand side vector b.

kwargs [dict, optional] Additional input parameters (refer to *spgl1.spgl1* for a list of possible parameters)

Returns

x [array_like, shape (n,)] Inverted model

r [array_like, shape (m,)] Final residual

g [array_like, shape (h,)] Final gradient

info [dict] See splg1.

Examples using spgl1.spg_bp

- sphx_glr_tutorials_spgl1.py

spgl1.spg_bpdn

spgl1.**spg_bpdn**(*A*, *b*, *sigma*, ***kwargs*)

Basis pursuit denoise (BPDN) problem.

spg_bpdn is designed to solve the basis pursuit denoise problem:

```
(BPDN)  minimize  ||x||_1  subject to  ||A x - b|| <= sigma
```

where A is an M-by-N matrix, b is an M-vector. A can be an explicit M-by-N matrix or a scipy.sparse.linalg.LinearOperator.

This is equivalent to calling ‘‘spgl1(A, b, tau=0, sigma=sigma)

Parameters

A [{sparse matrix, ndarray, LinearOperator}] Representation of an m-by-n matrix. It is required that the linear operator can produce Ax and A^T x.

b [array_like, shape (m,)] Right-hand side vector b.

kwargs [dict, optional] Additional input parameters (refer to *spgl1.spgl1* for a list of possible parameters)

Returns

x [array_like, shape (n,)] Inverted model

r [array_like, shape (m,)] Final residual

g [array_like, shape (h,)] Final gradient

info [dict] See spgl1.

Examples using `spgl1.spg_bpdn`

- sphx_glr_tutorials_spgl1.py

spgl1.spg_lasso

`spgl1.`**`spg_lasso`**(*A*, *b*, *tau*, ***kwargs*)

LASSO problem.

`spg_lasso` is designed to solve the Lasso problem:

```
(LASSO)  minimize  ||Ax - b||_2  subject to  ||x||_1 <= tau
```

where `A` is an M-by-N matrix, `b` is an M-vector. `A` can be an explicit M-by-N matrix or a `scipy.sparse.linalg.LinearOperator`.

This is equivalent to calling ``spgl1(A, b, tau=tau, sigma=0)

Parameters

A [{sparse matrix, ndarray, LinearOperator}] Representation of an m-by-n matrix. It is required that the linear operator can produce `Ax` and `A^T x`.

b [array_like, shape (m,)] Right-hand side vector `b`.

kwargs [dict, optional] Additional input parameters (refer to *`spgl1.spgl1`* for a list of possible parameters)

Returns

x [array_like, shape (n,)] Inverted model

r [array_like, shape (m,)] Final residual

g [array_like, shape (h,)] Final gradient

info [dict] See spgl1.

Examples using `spgl1.spg_lasso`

- sphx_glr_tutorials_spgl1.py

spgl1.spg_mmv

`spgl1.`**`spg_mmv`**(*A*, *B*, *sigma=0*, ***kwargs*)

MMV problem.

`spg_mmv` is designed to solve the multi-measurement vector basis pursuit denoise:

```
(MMV)  minimize  ||X||_1,2  subject to  ||A X - B||_2,2 <= sigma
```

where `A` is an M-by-N matrix, `b` is an M-by-G matrix, and ``sigma` is a nonnegative scalar. `A` can be an explicit M-by-N matrix or a `scipy.sparse.linalg.LinearOperator`.

Parameters

A [{sparse matrix, ndarray, LinearOperator}] Representation of an M-by-N matrix. It is required that the linear operator can produce `Ax` and `A^T x`.

b [array_like, shape (m,)] Right-hand side matrix b of size M-by-G.

sigma [float, optional] BPDN threshold. If different from None, spgl1 solves BPDN problem

kwargs [dict, optional] Additional input parameters (refer to *spgl1.spgl1* for a list of possible parameters)

Returns

x [array_like, shape (n,)] Inverted model

r [array_like, shape (m,)] Final residual

g [array_like, shape (h,)] Final gradient

info [dict] See spgl1.

Examples using `spgl1.spg_mmv`

- sphx_glr_tutorials_mmvnn.py
- sphx_glr_tutorials_spgl1.py

2.3 Contributors

- Andreas Doll, andreasdoll
- Matteo Ravasi, mrava87
- David Relyea, drrelyea

Bibliography

[1] E. van den Berg and M. P. Friedlander, "Probing the Pareto frontier for basis pursuit solutions", SIAM J. on Scientific Computing, 31(2):890-912. (2008).

[2] E. van den Berg and M. P. Friedlander, "Sparse optimization with least-squares constraints", Tech. Rep. TR-2010-02, Dept of Computer Science, Univ of British Columbia (2010).

Index

O

S