SGP-dec

A Scaled Gradient Projection method for 2D and 3D images deconvolution

RICCARDO ZANELLA[1] LUCA ZANNI[2] GAETANO ZANGHIRATI[3] ROBERTO CAVICCHIOLI[2]

[1] *Laboratory for Technologies of Advanced Therapies, University of Ferrara, Italy*
E-mail: riccardo.zanella@unife.it

[2] *Department of Pure and Applied Mathematics, University of Modena and Reggio Emilia, Italy*
E-mail: luca.zanni@unimore.it ; roberto.cavicchioli@unimore.it

[3] *Department of Mathematics, University of Ferrara, Italy*
E-mail: g.zanghirati@unife.it

Version 1.0 – Revision: July 2011

Overview

SGP-dec is a Matlab package for the deconvolution of 2D and 3D images corrupted by Poisson noise. Following a maximum likelihood approach, SGP-dec computes a deconvolved image by early stopping an iterative method for the solution of the following minimization problem:

$$\underset{\boldsymbol{x}\in\Omega}{\text{minimize}} \;\; J(\boldsymbol{x}) = \sum_{i=1}^{n} \left(\sum_{j=1}^{n} A_{ij}x_j + b - g_i - g_i \log \frac{\sum_{j=1}^{n} A_{ij}x_j + b}{g_i} \right), \tag{1}$$

where $\boldsymbol{x} \in \mathbb{R}^n$ denotes the unknown image, A_{ij}, $i,j = 1,\ldots,n$, are the entries of the convolution matrix, $b > 0$ denotes a constant background, g_i, $i = 1,\ldots,n$, are the entries of the observed (noisy) image and the feasible set Ω describes nonnegativity constraints

$$\Omega = \{\boldsymbol{x} \in \mathbb{R}^n \,|\, x_i \geq 0, \;\; \forall\, i = 1,\ldots,n\} \tag{2}$$

or nonnegativity and flux conservation constraints

$$\Omega = \left\{\boldsymbol{x} \in \mathbb{R}^n \,\middle|\, x_i \geq 0, \;\; \forall\, i = 1,\ldots,n, \quad \textstyle\sum_{i=1}^{n} x_i = c\right\}. \tag{3}$$

The objective function $J(\boldsymbol{x})$ in (1) is known as *generalized Kullback-Leibler divergence* of the blurred image $(A\boldsymbol{x} + b)$ from the observed image $\boldsymbol{g}$. Inside the package, the vectors $\boldsymbol{x}$ and $\boldsymbol{g}$ are obtained by columnwise stacking (in Fortran-like fashion) the 2D or 3D arrays representing the unknown and the observed images, respectively. Furthermore, the normalization condition $A^T\mathbf{1} = \mathbf{1}$ is assumed for the deconvolution matrix, where $\mathbf{1} = (1,\ldots,1)^T$.

The iterative minimization method implemented by SGP-dec is the *Scaled Gradient Projection* (SGP) algorithm introduced in [4], that can be considered an acceleration of the *Expectation Maximization* method [8], also known as Richardson-Lucy method [7, 6]. The main feature of the SGP algorithm consists in the combination of non-expensive diagonally scaled gradient directions with adaptive Barzilai-Borwein steplength rules [1, 5] specially designed for these directions; global convergence properties are ensured by exploiting a line-search strategy (monotone or nonmonotone) along the feasible direction.

The algorithm SGP is provided to be used as iterative regularization method; this means that a regularized reconstruction can be obtained by early stopping the SGP sequence. Several early stopping strategies can be selected, based on different criteria: maximum number of iterations, distance of successive iterations or function values, discrepancy principle [3] (see Table 1 for the list of the available stopping criteria); the user must choose a stopping criterion and fix suited values for the parameters involved by the chosen criterion.

We refer to [4, 2] for a detailed analysis of the SGP behaviour in comparison with other deconvolution approaches.

Source Code

This software is distributed under the General Public License v. 3 and it's available at

`http://www.unife.it/prisma/software`

The package is developed within the research project

PRISMA – Optimization methods and software for inverse problems

funded by the Italian Ministry for University and Research (MIUR), under the PRIN2008 initiative, grant n. 2008T5KA4L, 2010-2012 (see `http://www.unife.it/prisma` for more details). This software is part of the package *IRMA – Image Reconstruction in Microscopy and Astronomy*, currently under development within the same PRISMA project.

The sources are tested on Matlab v. R2009a or newer, under both Linux and Windows systems.

Installing the code

To install the package, decompress the archive file in a folder of your choice. Then, open Matlab and add the package's directory to your Matlab path. Now you can use the package with your own data. Two predefined test problems are available for testing purposes.

Using the code

The current distribution is organized with three subfolders:

- `DEBLURR` : main program with functions;
- `input` : input data and predefined test problems;
- `output` : reconstructed image and output information.

The main function is `sgp_deblurring`, whose simpler calling syntax is the following:

```
[x, iter, err, discr, time] = sgp_deblurring(A, gn)
```

The input/output parameters are described in Tab. 1, together with the available options. The optional arguments must be provided in the form of keyword/value pairs. Look at the file `test_deblurring.m` for an example of the function call.

N.B. The code requires that the PSF is normalized. Wether the blurring operator is provided as a matrix or as a function, it is required that all rows sum-up to 1.

Testing

Here we provide some execution examples. Two test sets are available:

- `NGC7027_255`: image of a nebula with maximum values of data set to 255;
- `satellite_25500`: image of a satellite with maximum values of data set to 25500.

The test problems are obtained by convolving the original 256×256 images with given PSF, then adding a constant background term and perturbing the resulting images with poisson noise.

We run `sgp_deblurring` with a prefixed maximum number of iterations: the following table reports the minimum relative ℓ_2 error and the iteration in which this minimum has been obtained:

Image	iter.	minimum error
`NGC7027_255`	27	1.380E−1
`satellite_25500`	332	2.887E−1

Possible bugs, comments and/or suggestions can be reported to one of the authors by e-mail.

References

[1] J. Barzilai and J. M. Borwein. Two point step size gradient methods. *IMA J. Numer. Anal.*, 8:141–148, 1988.

[2] F. Benvenuto, R. Zanella, L. Zanni, and M. Bertero. Nonnegative least-squares image deblurring: improved gradient projection approaches. *Inverse Problems*, 26(2):025004, February 2010.

[3] M. Bertero, P. Boccacci, G. Talenti, R. Zanella, and L. Zanni. A discrepancy principle for poisson data. *Inverse Problems*, 26(10):105004, October 2010.

[4] S. Bonettini, R. Zanella, and L. Zanni. A scaled gradient projection method for constrained image deblurring. *Inverse Problems*, 25(1):015002, January 2009.

[5] G. Frassoldati, G. Zanghirati, and L. Zanni. New adaptive stepsize selections in gradient methods. *J. Industrial and Management Optim.*, 4(2):299–312, 2008.

[6] L. B. Lucy. An iterative technique for the rectification of observed distributions. *Astronom. J.*, 79:745–754, 1974.

[7] W. H. Richardson. Bayesian–based iterative method of image restoration. *J. Opt. Soc. Amer. A*, 62:55–59, 1972.

[8] L. A. Shepp and Y. Vardi. Maximum likelihood reconstruction for emission tomography. *IEEE Trans. Med. Imaging*, 1:113–122, 1982.

Table 1: available input/output arguments to the function `sgp_deblurring`. Each optional input argument must be provided in the form of keyword/value pairs.

name	type	default	meaning
		mandatory input arguments	
`A`	double array or function handle	none	measuring matrix or function handle, used to compute $A\boldsymbol{x}$. If `A` is a function handle, also `AT` (that is the operator applying A^T) is required. **Remark**: all columns of `A` must sum-up to 1.
`gn`	double array	none	measured image (known data).
		optional input arguments	
`'AT'`	function handle	none	operator computing $A^T\boldsymbol{x}$
`'OBJ'`	double array	none	"exact" solution (if available) for error calculation
`'BG'`	double	0	uniform background as a scalar value

`'INITIALIZATION'`	either $i \in \{0,1,2,3\}$ or a double array	0	`0` = all zero starting point
			`1` = random starting point
			`2` = initialization with `g`
			`3` = initialization with `ones(size(g))*sum(g(:) - b)/numel(g)`
			array = custom-provided starting point

`'MAXIT'`	positive integer	1000	maximum number of iterations allowed
`'VERBOSE'`	integer $i \in \{0,1,2\}$	0	`0` = silent `1` = print configuration of the parameters at startup `2` = in addition to the previous, print also some information at each iteration
`'STOPCRITERION'`	integer $i \in \{1,2,3,4\}$	1	`1`: `iter` $>$ `MAXIT` `2`: $\|\boldsymbol{x}^{(k)} - \boldsymbol{x}^{(k-1)}\| \leq$ `tol`$\|\boldsymbol{x}^{(k)}\|$ or 1 `3`: $\|J(\boldsymbol{x}^{(k)}) - J(\boldsymbol{x}^{(k-1)})\| \leq$ `tol`$\|J(\boldsymbol{x}^{(k)})\|$ or 1 `4`: $\frac{2}{N}J(\boldsymbol{x}^{(k)}) \leq$ `tol` or 1
`'TOL'`	positive double	either 1E−4	1E−4 for the stopping criterion 2 or 3
		or 1+1/`mean(g)`	1+1/`mean(g)` for the stopping criterion 4
`'M'`	positive integer	1	function values "memory" for the line-search (if `M` = 1, the algorithm is monotone)
`'GAMMA'`	positive double	1E−4	sufficient decrease parameter in the line-search
`'BETA'`	positive double	0.4	backtracking parameter in the line-search
`'ALPHA_MIN'`	positive double	1E−5	lower bound for the Barzilai-Borwein parameter α^{BB}
`'ALPHA_MAX'`	positive double	1E+5	upper bound for the Barzilai-Borwein parameter α^{BB}
`'MALPHA'`	positive integer	3	memory for α_2^{BB} values
`'TAUALPHA'`	positive double	0.5	parameter for alternating between α_1^{BB} and α_2^{BB}
`'INITALPHA'`	positive double	1.3	initial value α_0
`'OMEGA'`	string	`'nonneg'`	constraints type: `'nonneg'` = non negativity `'nonneg_eq'` = non negativity and flux conservation (estimated flux is `sum(g(:) - b)`)
`'SAVE'`	string	none	if available, it denotes the directory where $\boldsymbol{x}^{(k)}$ and the residual $(\boldsymbol{x}^{(k)} - \boldsymbol{g})/\sqrt{\boldsymbol{x}^{(k)}}$ are saved at each iteration
		output arguments	
`x`	double array		computed solution (reconstructed data)
		optional output arguments	
`iter`	positive integer		number of iterations performed
`err`	double array		error value at each iteration. If `OBJ` was not given, `err` is an empty matrix
`discr`	double array		discrepancy at each iteration: $D_k = 2J(\boldsymbol{x}^{(k)})/$`numel`$(\boldsymbol{g})$
`time`	double array		CPU time in seconds after each iteration