User's Manual for OSGA (Optimal SubGradient Algorithm)

Masoud Ahookhosh
Faculty of Mathematics, University of Vienna, Vienna, Austria

December 29, 2019

Abstract

This document provides a user's guide describing the design of a MATLAB package called OSGA and how to use it for solving large-scale unconstrained, bound-constrained, and simply constrained convex optimization.

Contents

1 Introduction

OSGA is a MATLAB package designed to solve large-scale convex optimization problems with simple constraints using optimal subgradient algorithm proposed by Neumaier in [7]. It was developed for practical unconstrained, bound-constrained, and simply constrained convex optimization by Ahookhosh [1] and Ahookhosh & Neumaier [2, 3]. The algorithm requires first-order information and can solves problems involving high-dimensional data due to its low memory requirement.

This version of OSGA solves minimization problems with the following multi-term affine composite objective function

$$\min_{x\in C} \quad \left[\Psi(x) = \sum_{i=1}^{n_1} f_i(\mathcal{A}_i x) + \sum_{j=1}^{n_2} \varphi_j(\mathcal{W}_j x)\right], \tag{1}$$

where $\mathcal{X}$ is a finite-dimensional real vector space, $f_i : \mathcal{U}_i \to \mathbb{R}$ for $i = 1, 2, \cdots, n_1$ are convex smooth functions, $\varphi_j : \mathcal{V}_j \to \mathbb{R}$ for $j = 1, 2, \cdots, n_2$ are smooth or nonsmooth convex functions and $\mathcal{A}_i : \mathcal{X} \to \mathcal{U}_i$ for $i = 1, 2, \cdots, n_1$, $\mathcal{W}_j : \mathcal{X} \to \mathcal{V}_j$ for $j = 1, 2, \cdots, n_2$ are linear operators, ans C is a simple domain. Under considered properties of Ψ, the problem (1) has a global optimizer denoted by x^* and $\Psi^* = \Psi(x^*)$ is its global optimum.

There are lots of applications like those arising in signal and image processing, compressed sensing, geophysics, sparse optimization, statistics and so on involving a minimization problem of the form (1). For example, the LASSO problem

$$\min_{x\in\mathbb{R}^n} \quad \frac{1}{2}\|\mathcal{A}x - y\|_2^2 + \lambda\|x\|_1, \tag{2}$$

appears frequently in sparse optimization, compressed sensing and statistics. Moreover, many problems in signal and image processing is formulated as follows

$$\min_{x\in C} \quad \frac{1}{2}\|\mathcal{A}x - y\|_2^2 + \lambda\|x\|_{TV}, \tag{3}$$

where $\|.\|_{TV}$ denotes either isotropic or anisotropic total variation, see [6], and C is the nonnegativity constraint or a box $C = \{x \in \mathcal{X} \mid x \in [\underline{x}, \overline{x}]\}$. To observe more examples, see [1, 2, 3] and the references therein.

We would invite the user to see the algorithmic framework, theory and numerical experiments of OSGA in [1, 2, 3, 7] before using OSGA's software.

2 Software description

OSGA is available from the URL:

`http://homepage.univie.ac.at/masoud.ahookhosh/`

In order to use OSGA, simply unpack the compressed file and add the base directory to your MATLAB path. Once it is installed, some drivers are available and ready to run, see Examples Section. You also can solve your own problem by OSGA, where it will be discussed in the rest of this section.

The package includes the following m-files and folders:

- Images: a folder involves some test images;
- Test_problems: a folder consists of test problems:
  - `L22L22R.m`: a least squares problem with the l_2^2 regularizer;
  - `L22L1R.m`: a least squares problem with the l_1 regularizer;
  - `L22ITVR.m`: a least squares problem with the isotropic TV regularizer;
  - `L22R.m`: a least squares problem;
- Drivers: a folder involves the following drivers:
  - `Deblurring_driver.m`: a deblurring driver for unconstrained problems;
  - `Denoising_driver.m`: a denoising driver for unconstrained problems;
  - `Inpainting_driver.m`: an inpainting driver for unconstrained problems;
  - `L1_driver.m`: a driver for an unconstrained l_1 problem;
  - `Sparse_recovery_driver.m`: a driver for an unconstrained sparse recovery problem;
  - `Deblurring_non_driver.m`: a deblurring driver for a nonnegativity constrained problem;
  - `L22L22R_BounCon_driver.m`: a driver for a bound-constrained l_2^2 problem;
  - `Ridge_regression_driver.m`: a driver for a Ridge regression problem;
- Subproblem_solvers: a folder involves solvers for OSGA's subproblem:
  - `Subuncon.m`: a solver for unconstrained OSGA's subproblem;
  - `Subnocon.m`: a solver for nonnegativity constrained OSGA's subproblem;

  - `Subbocon.cpp`: a mex-file of a solver for bound-constrained OSGA's subproblem (Algorithm 3 in [2]);
  - `Subbocon_fzero.m`: a solver for bound-constrained OSGA's subproblem (Algorithm 3 in [3]);
  - `Subebcon.m`: a solver for l_2-ball constrained OSGA's subproblem;

- main m-files:
  - `Contents.m`: a script describing the OSGA package;
  - `LOPMVM.m`: a function for applying forward and adjoint linear operators and matrix-vector multiplications;
  - `OSGA.m`: OSGA's main function;
  - `OSGA_Init.m`: a function for initializing OSGA's parameters;
  - `proj_l2.m`: a function for projecting onto l_2-ball;
  - `Prox_Quad.m`: a function for applying quadratic prox-function defined in [1];
  - `Prox_Quad_Con.m`: a function for applying quadratic prox-function defined in [3];
  - `StopCrit.m`: a function for stopping the algorithm;
  - `SubGradEval.m`: a function used in `LOPMVM.m` for applying forward and adjoint linear operators and matrix-vector multiplications;
  - `Update_Pars.m`: A function for updating OSGA's parameters (Algorithm 3.1 in [7]);

2.1 Inputs and outputs

OSGA has a simple interface with 5 input arguments and 1, 2 or 3 output arguments described in the following.

Input arguments. OSGA requires the following input arguments:

- `func`: a function handle for an objective function;
- `prox`: a function handle for a prox-function;
- `subprob`: a function handle for OSGA's subproblem solver;

- `x0`: an initial point;
- `options`: a structure involving the parameters needed for OSGA.

The structure `options` involves the subsequent options that can be specified by users. Otherwise, OSGA uses its default parameters. To see the default value of the parameters, users are invited to take a look at the function `OSGA_Init.m`.

- `A`: a cell including matrices or linear operators;
- `MaxNumIter`: the maximum number of iterations;
- `MaxNumFunEval`: the maximum number of function evaluations;
- `MaxNumSubGradEval`: the maximum number of subgradient evaluations;
- `MaxNumLinOper`: the maximum number of linear operators;
- `TimeLimit`: the maximum running time;
- `StepLenLowBound`: the lower bound for difference of two last points;
- `StepDiffLowBound`:
- `epsilon`: the accuracy parameter;
- `f_target`: the lower bound on function values;
- `delta`: a global tuning parameter for updating scheme PUS, see [1];
- `alpha_max`: the maximum step size;
- `kappa`: a global tuning parameter for updating scheme PUS, see [1];
- `kappap`: a global tuning parameter for updating scheme PUS, see [1];
- `mu`: the strong convexity parameter;
- `D`: the preconditioner matrix for the quadratic norm;
- `q0`: value of prox-function Q in its center;
- `xs`: the optimizer;
- `fs`: the optimum;

- `cons`: type of the constraint;
- `flag_MSE`: saves `MSE` if it sets to 1;
- `flag_ISNR`: saves `ISNR` if it sets to 1;
- `flag_x_error`: saves `x_error` if it sets to 1;
- `flag_f_error`: saves `f_error` if it sets to 1;
- `Stopping_Crit`: a number from 1 to 8 determining a stopping criterion.

Output arguments. The outputs of OSGA are as follows:

- `x`: the best approximation of the optimizer obtained by OSGA;
- `f`: the best approximation of the optimum obtained by OSGA;
- `out`: the structure involving more output information.

OSGA offers various output options for the user using the structure `out`. In details, it involves the following options:

- `T`: the running time;
- `Niter`: the total number of iterations;
- `Nfunc`: the total number of function evaluations;
- `Nsubgrad`: the total number of subgradient evaluations;
- `Nlinop`: the total number of employed linear operators;
- `F`: the array including all function values
- `MSE`: the mean squared error defined by

$$\mathrm{MSE} = \frac{1}{n}\|x_b - x_0\|^2,$$

  where x_0 is the original signal, and x_b is the current approximation;

- `ISNR`: the improvement signal to the noise defined by

$$\mathrm{ISNR} = 20\log_{10}\left(\frac{\|Y - X_0\|_F}{\|X_b - X_0\|_F}\right),$$

  where X_0 denotes the $m \times n$ clean image, Y is the observed image and X_b is the current approximation;

- `x_error`: the relative error of points defined by

$$\delta_1 := \frac{\|x_b - x^*\|_2}{\|x^*\|_2};$$

- `f_error`: the relative error of function values defined by

$$\delta_2 := \frac{f_k - f^*}{f_0 - f^*};$$

- `Status`: the reason indicating why OSGA stopped.

To get the best performance of OSGA, it is often requiring some parameters to be tuned, but users have requested viewing the default values before any tuning.

2.2 Forward and adjoint operations

Considering the problem (1), OSGA requires that the cell array `A` containing matrices or linear operators is defined by

$$\mathtt{A} = \{\{\mathtt{A}_1, \cdots, \mathtt{A}_{\mathtt{n}_1}, \mathtt{A}_1^*, \cdots, \mathtt{A}_{\mathtt{n}_1}^*\}, \{\mathtt{W}_1, \cdots, \mathtt{W}_{\mathtt{n}_2}, \mathtt{W}_1^*, \cdots, \mathtt{W}_{\mathtt{n}_2}^*\}\},$$

where $\mathtt{A}_1, \cdots, \mathtt{A}_{\mathtt{n}_1}$ and $\mathtt{A}_1^*, \cdots, \mathtt{A}_{\mathtt{n}_1}^*$ respectively stand for forward and adjoint linear operators (matrices) of smooth functions, and $\mathtt{W}_1, \cdots, \mathtt{W}_{\mathtt{n}_2}$ and $\mathtt{W}_1^*, \cdots, \mathtt{W}_{\mathtt{n}_2}^*$ stand for forward and adjoint linear operators (matrices) of nonsmooth functions. For example, for the LASSO problem (2), `A` is constructed by

$$\mathtt{A} = \{\{\mathtt{A}_1, \mathtt{A}_1^*\}, \{\}\},$$

Let consider the scaled elastic net problem

$$\min_{x \in \mathbb{R}^n} \quad \frac{1}{2}\|A_1 x - b\|_2^2 + \frac{\lambda_2}{2}\|A_2 x\|_2^2 + \lambda_1 \|W_1 x\|_1. \tag{4}$$

Then the cell array `A` is constructed by

$$\mathtt{A} = \{\{\mathtt{A}_1, \mathtt{A}_2, \mathtt{A}_1^*, \mathtt{A}_2^*\}, \{\mathtt{W}_1, \mathtt{W}_1^*\}\}$$

because first and second function in (4) are smooth and the third one is nonsmooth.

In the package forward and adjoint linear operations are managed by the function `LOPMVM` as follows

- `Ax` = `feval`(`LOPMVM`(`A`), `x`, `1`);
- `Asx` = `feval`(`LOPMVM`(`A`), `x`, `2`);

where `Ax` stands for the forward operation and `Asx` stands for the adjoint operation.

2.3 Stopping criteria

The package offers 8 commonly used stopping criteria to terminate OSGA, which can be specified by setting the parameter `Stopping_Crit` equal to 1 to 8. In particular, each number means that the algorithm stops

- 1 : if the maximum number of iterations is reached;
- 2 : if the maximum number of function evaluations is reached;
- 3 : if the maximum number of subgradient evaluations is reached;
- 4 : if the maximum number of linear operators is reached;
- 5 : if the maximum running time is reached;
- 6 : if $\eta <= \epsilon$;
- 7 : if $\|x_{x_b} - x_{old}\| / \max(1, \|x_{x_b}\|) <= \epsilon$;
- 8 : if $f(x_b) <= f_{target}$;

where x_b is the current best founded approximate of the optimizer and x_{old} is the former best founded approximate of the optimizer.

2.4 Building your own problem

To solve your own problem of the form (1), one just needs to write a routine to calculate function values and subgradients of the objective function. Based on construction of the cell array `A`, we require to divide the objective function to the following four parts:

- those parts that are smooth and involve linear operators;
- those parts that are nonsmooth and involve linear operators;
- the part that is smooth and involves no linear operators;
- the part that is nonsmooth and involves no linear operators.

Therefore, the output arguments of the routine are

- `fx` : a number returning the objective function value at `x`;

- `gx1` : a cell array returning the subgradients of those parts involving linear operators defined by

$$\texttt{gx1} = \big\{\{\texttt{g}_1^1, \cdots, \texttt{g}_{\texttt{n}_1}^1\}, \{\texttt{g}_1^2, \cdots, \texttt{g}_{\texttt{n}_2}^2\}\big\},$$

  where $g_1^1, \cdots, g_{n_1}^1$ are gradients of smooth parts, and $g_1^2, \cdots, g_{n_2}^2$ are subgradients of nonsmooth parts;

- `gx2` : a cell array returning the subgradients of those parts involving no linear operators defined by

$$\texttt{gx1} = \big\{\texttt{g}^3, \texttt{g}^4\big\},$$

  where g^3 is the gradient of the smooth part, and g^4 is a subgradient of the nonsmooth part.

Finally, the subgradient of whole function is constructed using the m-file `SubGradEval.m` by setting

$$\texttt{gx} = \sum_{\texttt{i}=1}^{\texttt{n}_1} \texttt{A}_\texttt{i}^* \texttt{g}_\texttt{i}^1 + \sum_{\texttt{i}=1}^{\texttt{n}_2} \texttt{W}_\texttt{i}^* \texttt{g}_\texttt{i}^2 + g^3 + g^4.$$

To see examples of this construction, observe the functions `L22L1R.m` and `L22ITVR.m` for the problems (2) and (3), respectively. Note that in this construction adjoint operations are done in `SubGradEval.m`.

2.5 Examples

To illustrate how to use the package, OSGA includes 8 practical examples for solving problems of the form (1).

We here only begin by going though the examples

- `Deblurring_driver.m` for an unconstrained problem;
- `Deblurring_non_driver.m` for a nonnegativity constrained problem;
- `L22L22R_BonCon_driver.m` for a bound-constrained problem.

We invite the user to first take a look at these scripts to see how problems are constructed and solved by OSGA.

Example. 1 *The m-file* `Deblurring_driver.m` *creates a blurred/noisy version of the* 512×512 *Goldhill image and restores it with OSGA by using the model of the form (3). To see details of the implementation, we advise to observe* `Deblurring_driver.m` *in which the details are fully described. To execute this m-file, it is enough to in MATLAB command line write down:*

>> `Deblurring_driver`

The outputs should be the same as Figures 1 and 2, where OSGA-1 and OSGA-2 are versions of OSGA stopped at 5 and 15 seconds of the running time.

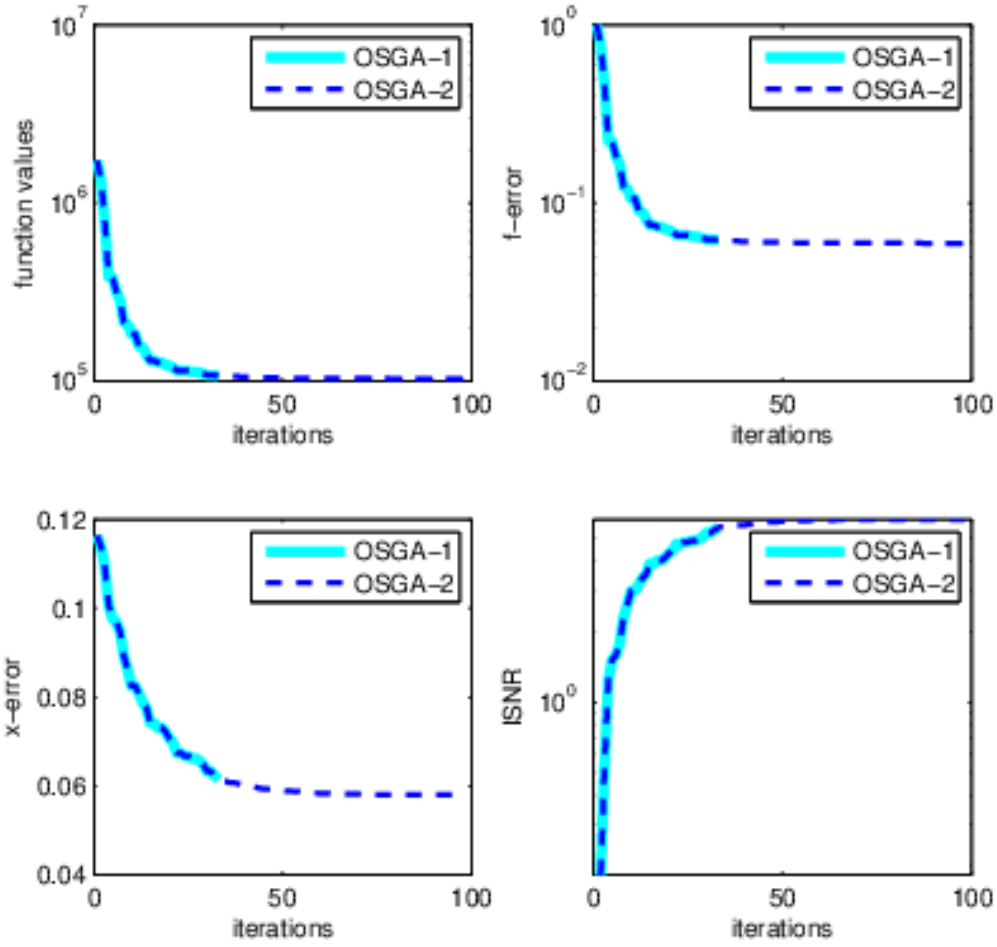

Figure 1: The details of deblurring of the 512 × 512 Goldhill image with OSGA-1 and OSGA-2

One can execute the next examples in the same way of Example 1:

- `Denoising_driver.m`,

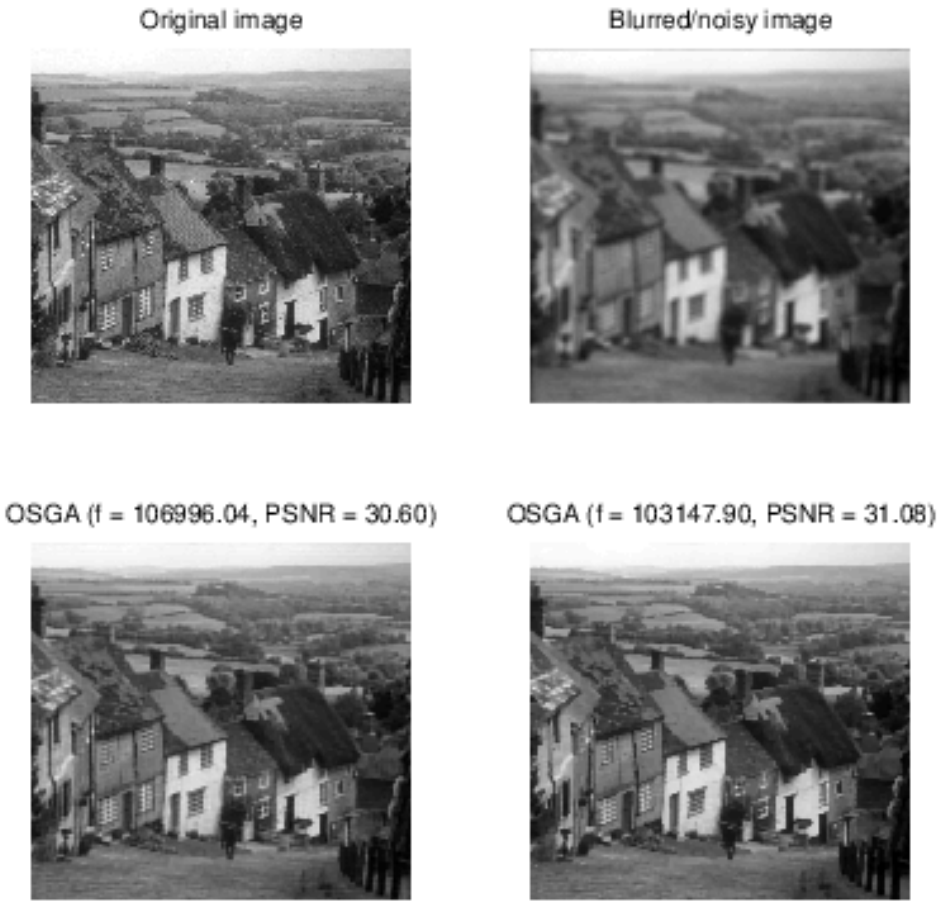

Figure 2: Deblurring of the 512 × 512 Goldhill image with OSGA-1 and OSGA-2

- `Inpainting_driver.m`,
- `L1_driver.m`,
- `Sparse_recovery_driver.m`

Example. 2 *The m-file* `L22L22R_BonCon_driver.m` *is used to create a bound-constrained problem to compare two versions of OSGA: a version uses Algorithm 3 in [2] to solve OSGA's subproblem (OSGA-1); a version uses Algorithm 3 in [3] to solve OSGA's subproblem (OSGA-2). It is executed by:*

>> `L22L22R_BonCon_driver`

The algorithms are stopped after 100 iterations, and the result of the implementation is illustrated in Figure 3.

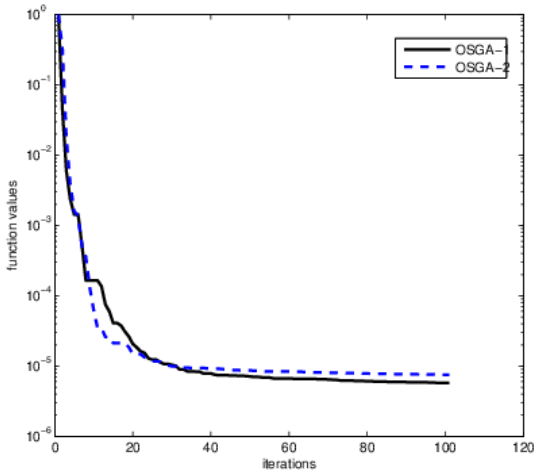

Figure 3: A comparison among OSGA-1 and OSGA-2 for solving a bound-constrained problem

Example. 3 *The m-file* `Deblurring_non`$_{\mathtt{d}}$`river.m` *creates a blurred/noisy version of the* 256×256 *MR-brain image and restores it with two versions of OSGA by the model of the form (3). We execute this m-file by:*

>> `Deblurring_driver`

The results are illustrated in Figures 4 and 5, where OSGA-1 and OSGA-2 are unconstrained and nonnegativity constrained versions of OSGA, respectively. The algorithms are stopped after 100 iterations.

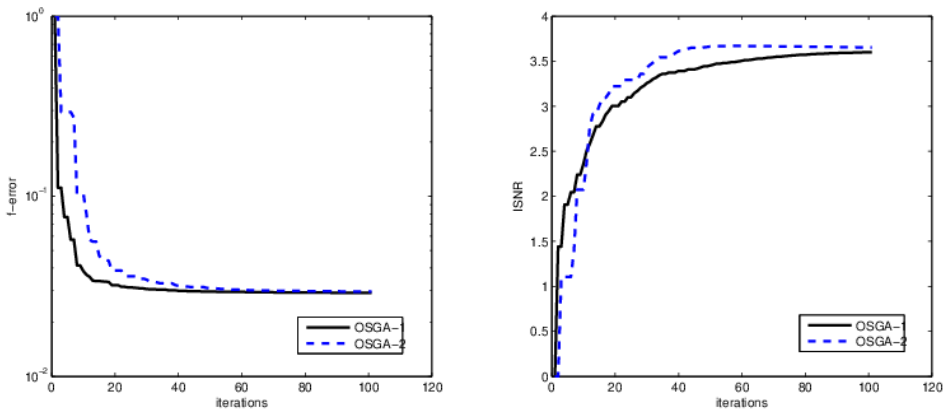

Figure 4: Deblurring of the 256×256 MR-brain image with OSGA-1 and OSGA-2

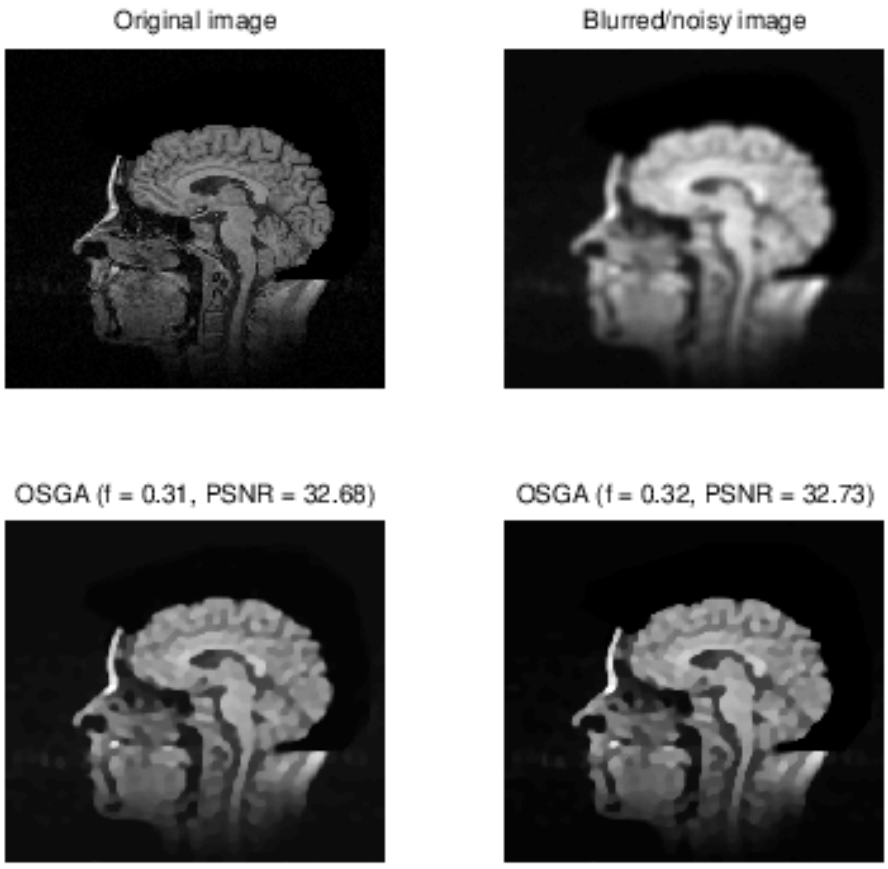

Figure 5: The details of deblurring of the 256×256 MR-brain image with OSGA-1 and OSGA-2

3 Feedback and support

Your feedback including suggestions and bug reports is welcome and appreciated. Moreover, if you use OSGA in your publications, you are requested to cite the accompanying papers [1, 2, 3, 4, 5, 7].

4 Acknowledgements

I would like to thank Arnold Neumaier for his useful feedback on this package.

References

[1] M. Ahookhosh, Optimal subgradient methods: computational properties for large-scale linear inverse problems. Optimization and Engineering, 19(4), 815-844 (2018).

[2] M. Ahookhosh, A. Neumaier, An optimal subgradient algorithm for large-scale bound-constrained convex optimization. Mathematical Methods of Operations Research, 86(1), 123-147 (2017).

[3] M. Ahookhosh, A. Neumaier, Optimal subgradient algorithms for large-scale convex optimization in simple domains, Numerical Algorithms, 76(4), 1071-1097 (2017).

[4] M. Ahookhosh, A. Neumaier, An optimal subgradient algorithm with subspace search for costly convex optimization problems, Bulletin of the Iranian Mathematical Society, 45(3), 883–910 (2019).

[5] M. Ahookhosh, A. Neumaier, Solving structured nonsmooth convex optimization with complexity $\mathcal{O}(\varepsilon^{-1/2})$, Top, 26(1), 110-145, (2018).

[6] A. Chambolle, V. Caselles, D. Cremers, M. Novaga, T. Pock, An introduction to total variation for image analysis, in Theoretical Foundations and Numerical Methods for Sparse Recovery. De Gruyter, 2010.

[7] A. Neumaier, OSGA: a fast subgradient algorithm with optimal complexity, Mathematical Programming, 158(1-2), 1-21 (2016).