BiOpt: Bilevel Optimization Toolboxes

Shenglong Zhou and Alain B. Zemkoho[†]

Abstract

To help accelerate the development of numerical toolboxes for bilevel optimization, BiOpt aims at providing i) a tool to calculate the first, second and third order derivatives of a single/set-valued function, ii) a tool to plot the graph of an optimal-value function in the form of $\psi(x) = \min_y\{f(x,y)|g(x,y) \leq 0\}$ and a tool to calculate the function value, iii) a collection of academic and real-world applications or case studies on the problem including 24 linear, 138 nonlinear and 11 simple bilevel optimization test examples and iv) three bilevel optimization solvers based on semi-smooth Newton method. All tools are programmed via Matlab and will be made freely available online.

Keywords: Bilevel optimization solvers, Set value function, Higher order derivatives

Mathematical Subject Classification: 90C26, 90C30, 90C90

Contents

1 Introduction **2**

2 GetDerivatives: A tool to calculate derivatives **2**

2.1 Format of derivatives 2

2.2 How to use GetDerivatives 4

3 Tools to process an optimal-value function **4**

3.1 SolOVF computes the function value 5

3.2 PlotOVF plots the function 6

4 Bilevel optimization examples from BOLIB **8**

4.1 Nonlinear examples 8

4.2 Linear examples 11

4.3 Simple examples 11

5 BiOpt solvers: SNLLVF, SNKKT and SNQVI **13**

5.1 Description of inputs and outputs 13

5.2 Examples and func 14

5.2.1 Examples with easy calculations of derivatives 15

5.2.2 Examples with complicated calculations of derivatives 19

5.3 Examples with parameters or extra data 21

5.4 Importance of pars.xy and pars.lam 24

5.5 Summary 25

[†]School of Mathematics, University of Southampton, Southampton SO17 1BJ, United Kingdom. E-mail: shenglong.zhou@soton.ac.uk, a.b.zemkoho@soton.ac.uk

1 Introduction

The general bilevel optimization problem can take the form

$$\begin{aligned} \min_{x,y} \quad & F(x,y) \\ \text{s.t.} \quad & G(x,y) \leq 0,\, H(x,y) = 0, y \in S(x), \end{aligned} \tag{1.1}$$

where the functions $G : \mathbb{R}^{n_x} \times \mathbb{R}^{n_y} \to \mathbb{R}^{n_G}$, $H : \mathbb{R}^{n_x} \times \mathbb{R}^{n_y} \to \mathbb{R}^{n_H}$ and $y \in S(x)$ define the upper-level constraints. The set valued mapping $S : \mathbb{R}^n \rightrightarrows \mathbb{R}^m$ describes the lower-level optimal solution set, for any upper-level selection x:

$$S(x) := \arg\min_y \ \{f(x,y) \mid g(x,y) \leq 0,\, h(x,y) = 0\}, \tag{1.2}$$

where $g : \mathbb{R}^{n_x} \times \mathbb{R}^{n_y} \to \mathbb{R}^{n_g}$ and $h : \mathbb{R}^{n_x} \times \mathbb{R}^{n_y} \to \mathbb{R}^{n_h}$ describe the lower-level constraints. On the other hand, $F : \mathbb{R}^{n_x} \times \mathbb{R}^{n_y} \to \mathbb{R}$ and $f : \mathbb{R}^{n_x} \times \mathbb{R}^{n_y} \to \mathbb{R}$ denote the upper-and lower-level objective/cost functions, respectively. Further recall that problem (1.1) as a whole is often called upper-level problem. We use the fact that $H(x,y) = 0$ (similarly to $h(x,y) = 0$) can be expressed as $H(x,y) \leq 0$ and $-H(x,y) \leq 0$. Hence, our focus here will be on bilevel optimization problems of the form

$$\begin{aligned} \min_{x,y} \quad & F(x,y) \\ \text{s.t.} \quad & G(x,y) \leq 0,\ S(x) := \arg\min_y \{f(x,y) :\ g(x,y) \leq 0\}. \end{aligned} \tag{1.3}$$

2 `GetDerivatives`: A tool to calculate derivatives

It is well known that Matlab provides users a function `jacobian` to compute the Jacobian matrix of a given single/set-valued function. When it comes to calculate the second order or third order derivatives of such given function, the usage of `jacobian` seems not to be straightforward and thus the computations are quite complicated. In this section, we offer a tool `GetDerivatives` whose core function is `jacobian` with ability to calculate the first, second and third order derivatives of an input (vector) function. Before to describe how to use this tool, we need some notational definitions.

2.1 Format of derivatives

Let $a \in \{x, y\}$ with $a \in \mathbb{R}^{n_a}$. Clearly, $n_a = n_x$ if $a = x$ and $n_a = n_y$ if $a = y$. Similarly, we define $b, c \in \{x, y\}$ with $b \in \mathbb{R}^{n_b}, c \in \mathbb{R}^{n_c}$. For a single-valued function $F(x,y) : \mathbb{R}^{n_x} \times \mathbb{R}^{n_y} \to \mathbb{R}$, its first and second derivatives are defined as follows,

$$\nabla_a F(x,y) = \begin{bmatrix} \nabla_{a_1} F \\ \vdots \\ \nabla_{a_{n_a}} F \end{bmatrix} \in \mathbb{R}^{n_a}, \tag{2.1}$$

$$\nabla^2_{ab} F(x,y) = \begin{bmatrix} \nabla^2_{a_1 b_1} F & \cdots & \nabla^2_{a_{n_a} b_1} F \\ \vdots & \ddots & \vdots \\ \nabla^2_{a_1 b_{n_b}} F & \cdots & \nabla^2_{a_{n_a} b_{n_b}} F \end{bmatrix} \in \mathbb{R}^{n_b \times n_a}. \tag{2.2}$$

The third derivative $\nabla^3_{abc}F(x,y)$ is given by

$$\nabla^3_{abc}F(x,y) = \begin{bmatrix} \nabla^2_{bc}(\nabla_{a_1}F) \\ \vdots \\ \nabla^2_{bc}(\nabla_{a_{n_a}}F) \end{bmatrix} = \begin{bmatrix} \nabla^3_{b_1c_1a_1}F & \cdots & \nabla^3_{b_{n_b}c_1a_1}F \\ \vdots & \ddots & \vdots \\ \nabla^3_{b_1c_{n_c}a_1}F & \cdots & \nabla^3_{b_{n_b}c_{n_c}a_1}F \\ \vdots & \ddots & \vdots \\ \nabla^3_{b_1c_1a_{n_a}}F & \cdots & \nabla^3_{b_{n_b}c_1a_{n_a}}F \\ \vdots & \ddots & \vdots \\ \nabla^3_{b_1c_{n_c}a_{n_a}}F & \cdots & \nabla^3_{b_{n_b}c_{n_c}a_{n_a}}F \end{bmatrix} \in \mathbb{R}^{n_a n_c \times n_b}. \tag{2.3}$$

For a set-valued function $G : \mathbb{R}^{n_x} \times \mathbb{R}^{n_y} \to \mathbb{R}^{n_G}$, its first and second derivatives are given by,

$$\nabla_a G(x,y) = \begin{bmatrix} \nabla_a G_1 \\ \vdots \\ \nabla_a G_{n_G} \end{bmatrix} = \begin{bmatrix} \nabla_{a_1} G_1 & \cdots & \nabla_{a_{n_a}} G_1 \\ \vdots & \ddots & \vdots \\ \nabla_{a_1} G_{n_G} & \cdots & \nabla_{a_{n_a}} G_{n_G} \end{bmatrix} \in \mathbb{R}^{n_G \times n_a}, \tag{2.4}$$

$$\nabla^2_{ab}G(x,y) = \begin{bmatrix} \nabla^2_{ab} G_1 \\ \vdots \\ \nabla^2_{ab} G_{n_G} \end{bmatrix} = \begin{bmatrix} \nabla^2_{a_1b_1} G_1 & \cdots & \nabla^2_{a_{n_a}b_1} G_1 \\ \vdots & \ddots & \vdots \\ \nabla^2_{a_1b_{n_b}} G_1 & \cdots & \nabla^2_{a_{n_a}b_{n_b}} G_1 \\ \vdots & \ddots & \vdots \\ \nabla^2_{a_1b_1} G_{n_G} & \cdots & \nabla^2_{a_{n_a}b_1} G_{n_G} \\ \vdots & \ddots & \vdots \\ \nabla^2_{a_1b_{n_b}} G_{n_G} & \cdots & \nabla^2_{a_{n_a}b_{n_b}} G_{n_G} \end{bmatrix} \in \mathbb{R}^{(n_G n_b) \times n_a}. \tag{2.5}$$

The third derivative $\nabla^3_{abc}G(x,y)$ is given by

$$\nabla^3_{abc}G(x,y) = \begin{bmatrix} \nabla^3_{abc} G_1 \\ \vdots \\ \nabla^3_{abc} G_{n_G} \end{bmatrix} = \begin{bmatrix} \nabla^2_{bc}(\nabla_{a_1} G_1) & \cdots & \nabla^2_{bc}(\nabla_{a_{n_a}} G_1) \\ \vdots & \ddots & \vdots \\ \nabla^2_{bc}(\nabla_{a_1} G_{n_G}) & \cdots & \nabla^2_{bc}(\nabla_{a_{n_a}} G_{n_G}) \end{bmatrix} \in \mathbb{R}^{n_G n_c \times n_a n_b}$$

$$= \begin{bmatrix} \nabla^3_{b_1c_1a_1} G_1 & \cdots & \nabla^3_{b_{n_b}c_1a_1} G_1 & \cdots & \nabla^3_{b_1c_1a_{a_n}} G_1 & \cdots & \nabla^3_{b_{n_b}c_1a_{a_n}} G_1 \\ \vdots & \ddots & \vdots & \cdots & \vdots & \ddots & \vdots \\ \nabla^3_{b_1c_{n_c}a_1} G_1 & \cdots & \nabla^3_{b_{n_b}c_{n_c}a_1} G_1 & \cdots & \nabla^3_{b_1c_{n_c}a_{a_n}} G_1 & \cdots & \nabla^3_{b_{n_b}c_{n_c}a_{a_n}} G_1 \\ \vdots & \ddots & \vdots & & & & \\ \nabla^3_{b_1c_1a_1} G_{n_G} & \cdots & \nabla^3_{b_{n_b}c_1a_1} G_{n_G} & \cdots & \nabla^3_{b_1c_1a_{a_n}} G_{n_G} & \cdots & \nabla^3_{b_{n_b}c_1a_{a_n}} G_{n_G} \\ \vdots & \ddots & \vdots & \cdots & \vdots & \ddots & \vdots \\ \nabla^3_{b_1c_{n_c}a_1} G_{n_G} & \cdots & \nabla^3_{b_{n_b}c_{n_c}a_1} G_{n_G} & \cdots & \nabla^3_{b_1c_{n_c}a_{a_n}} G_{n_G} & \cdots & \nabla^3_{b_{n_b}c_{n_c}a_{a_n}} G_{n_G} \end{bmatrix} \tag{2.6}$$

Note that the formats of derivatives $\nabla_a F(x,y)$ and $\nabla_{ab}F(x,y)$ of F and $\nabla_a G(x,y)$ are defined in a standard way, while the storage of $\nabla_{abc}F(x,y)$, $\nabla_{ab}(x,y)$ and $\nabla_{abc}G(x,y)$ in matrix might differ with methods from literature. We adopt this way because it allows us to use these formats clearly in terms of numerical computations. Since $a, b, c \in \{x, y\}$, we have

$$ab \in \{xx, xy, yx, yy\}, \quad abc \in \{xxx, xxy, xyx, xyy, yxx, yxy, yyx, yyy\}.$$

2.2 How to use `GetDerivatives`

Open folder `GetDerivatives` which contains 2 Matlab m-files: `GetDerivatives.m` and `demon.m`. Now we are ready to describe how to use this tool. The basic citation is

```
                DFunc  =  GetDerivatives(Func,dim,keyxy)
[DFunc,symDFunc]  =  GetDerivatives(Func,dim,keyxy)
```

Corresponding inputs and outputs are described as in Table 1.

Inputs:		
	`DFunc` :	The input function handle, $\mathbb{R}^{n_x \times n_y} \to \mathbb{R}^{n_f}$ with $n_f \geq 1$.
	`dim` :	$= [n_x \ n_y]$, the dimensions of variables $x \in \mathbb{R}^{n_x}$ and $y \in \mathbb{R}^{n_y}$.
	`keyxy` :	$\in$ {'x','y','xx','xy','yx','yy','xxx','xxy','xyx','xyy', 'yxx','yxy','yyx','yyy'}, derivative of `DFunc` w.r.t. `keyxy`.
Outputs:		
	`DFunc`:	Derivative of `DFunc` w.r.t. `keyxy`, a function handle.
	`symDFunc`:	Derivative of `DFunc` w.r.t. `keyxy`, a symbolic function handle.

Table 1: Inputs and outputs of `GetDerivatives.m`.

Type (or copy) the following codes in a new command window (or alternatively you can simply open `demon.m` and run it):

```
clc; clear; close all;                                  % line 1
Func   = @(x,y)([sin(x)+sum(y.^3); y.^norm(x,1)]);      % line 2
dim    = [2 3];                                         % line 3
keyxy  = 'xy';                                          % line 4
DFunc  = GetDerivatives(Func, dim, keyxy);              % line 5
Dkeyxy = DFunc(rand(dim(1),1),rand(dim(2),1))           % line 6
```

Line 2 defines the function handle `Func` as

$$\left[\sin(x_1) + \sum_{i=1}^{3} y_i^3,\ \sin(x_2) + \sum_{i=1}^{3} y_i^3,\ y_1^{\|x\|_1},\ y_2^{\|x\|_1},\ y_3^{\|x\|_1}\right]^\top.$$

In line 5, `DFunc` is the derivative w.r.t. to xy. To see other derivatives, just pick one `keyxy` from {'x','y','xx','xy','yx','yy','xxx', 'xxy','xyx','xyy','yxx','yxy','yyx','yyy'}.

3 Tools to process an optimal-value function

`BiOpt` also provides a folder `OptValFunc` which contains two tools to process an optimal-value function in the form of

$$\psi(x) = \min_{y \in \mathbb{R}^{n_y}} \{f(x,y) \mid g(x,y) \leq 0\}, \tag{3.1}$$

where $f : \mathbb{R}^{n_x} \times \mathbb{R}^{n_y} \to \mathbb{R}$ and $g : \mathbb{R}^{n_x} \times \mathbb{R}^{n_y} \to \mathbb{R}^{n_g}$. It highly suggests that for each fixed x, both $f(x, y)$ and $g(x, y)$ are convex functions with respect to y, so that (3.1) is a convex program which admits a unique optimal objective function value. In other words, if either $f(x, y)$ or $g(x, y)$ is non-convex , then these two tools may present incorrect (or even fail to generate) results as the optimal objective function value is not a singleton.

3.1 `SolOVF` computes the function value

Open folder `OptValFunc` in which there are 2 Matlab m-files: '`SolOVF.m`' (the main function handle) and '`demonSol.m`' (to do demonstration). This tool `SolOVF` solves (3.1) with providing the optimal solution y^* and the optimal objective function value $\psi(x) = f(x, y^*)$ for a given x. The citation of '`SolOVF.m`' has forms as

```
out  =  SolOVF(ny,objf,cong,varx),
out  =  SolOVF(ny,objf,cong,varx,yo).
```

Their inputs and outputs are described as in Table 3.

Inputs:	
`ny` :	Dimension of $y \in \mathbb{R}^{n_y}$.
`funf` :	Function handle describes the objective function, $f(x, y) : \mathbb{R}^{n_x \times n_y} \to \mathbb{R}$.
`cong` :	Function handle describes the inequality constraints, $g(x, y) : \mathbb{R}^{n_x \times n_y} \to \mathbb{R}^{n_g}$ with $n_g \geq 1$.
`varx` :	The given variable x.
`yo`:	Starting point of y. This is optional, with default one `yo=zeros(ny,1);`.
Outputs:	
`out.xvar`:	The given variable x
`out.yopt`:	The optimal solution.
`out.fopt`:	The optimal objective function value.

Table 3: Inputs and outputs of `SolOVF`.

Now, take one simple example to illustrate the usage of `SolOVF`. Consider

$$\psi(x) = \min_{y \in \mathbb{R}^{n_y}} \left\{ \|x\|^2 + \|y\|^2 - x^\top Q y \;\middle|\; \|y\|^2 \leq 25,\ \sum_{i=1}^{n_x} x_i + \sum_{i=1}^{n_y} y_i \leq 5 \right\}, \tag{3.2}$$

with $x \in \mathbb{R}^{n_x}$ and $Q \in \mathbb{R}^{n_x \times n_y}$. Type (or copy) the following codes in a new command window (or alternatively you can simply open `demonSol.m` and run it):

```
clc; clear; close all;                                          % line 1
dim    = [20 40];                                               % line 2
rng('default'); rng(1);                                         % line 3
Q      = randi(10,dim)/10;                                      % line 4
objf   = @(x,y)(sum(x.^2)-sum(x.*(Q*y))+sum(y.^2));             % line 5
cong   = @(x,y)[sum(x)+sum(y)-5; y.^2-25];                      % line 6
out    = SolOVF(dim(2),objf,cong,randn(dim(1),1))               % line 7
```

The second line gives the dimensions dim= $[n_x\ n_y] = [10\ 20]$. The output is

```
out =
    struct with fields:
      yopt: [40x1 double]
      fopt: 3.5189
      varx: [20x1 double]
```

For tool SolOVF, there is no restriction on the dimensions as long as $n_x \geq 1, n_y \geq 1$. Of course, the larger dimensions are, the much longer computational time this tool will cost. One can also change the dimensions to see other outputs.

Inputs:	
dim :	$= [n_x\ n_y]$, the dimensions of variables $x \in \mathbb{R}^{n_x}$ and $y \in \mathbb{R}^{n_y}$. $n_x \in \{1, 2\}$, other choices of n_x will fail this code. $n_y \in \{0, 1, 2, 3, \cdots\}$.
funf :	Function handle describes the objective function, $f(x, y) : \mathbb{R}^{n_x \times n_y} \to \mathbb{R}$.
cong :	Function handle describes the inequality constraints, $g(x, y) : \mathbb{R}^{n_x \times n_y} \to \mathbb{R}^{n_g}$ with $n_g \geq 1$.
range :	The range of x when plotting $\psi(x)$. This is optional. If $n_x = 1$, range = [xmin, xmax]. Default one, range= $[-1, 1]$; If $n_x = 2$, range = [x1min, x1max; x2min, x2max]; Default one, range= $[-1, 1; -1, 1]$;
nox:	The number of samples x. This is optional. Default one nox = 100 when $n_x = 1$, nox = 20 when $n_x = 2$, NOTE: The larger nox is, the longer the time is taken but the more accurate the graph will be.
Outputs:	
out.succ:	=1 plots successfully, =0 fails to plot.
out.xvar:	The variable with nox elements. $\in \mathbb{R}^{1 \times \texttt{nox}}$ when $n_x = 1$, $\in \mathbb{R}^{\texttt{nox} \times \texttt{nox} \times 2}$ when $n_x = 2$.
out.yopt:	The optimal solutions, i.e., $\text{argmin}_y\{f(x, y) \vert g(x, y) \leq 0\}$. $\in \mathbb{R}^{n_y \times \texttt{nox}}$ when $n_x = 1$, $\in \mathbb{R}^{\texttt{nox} \times \texttt{nox} \times n_y}$ when $n_x = 2$.
out.fopt:	The optimal objective function values with out.fopt$\in \mathbb{R}^{\texttt{nox}}$.

Table 5: Inputs and outputs of PlotOVF.

3.2 PlotOVF plots the function

Again open folder OptValFunc where one can find 2 Matlab m-files: 'PlotOVF.m' (the main function handle) and 'demonPlot.m' (to do demonstration). This tool PlotOVF presents the graph of (3.1) in two and three dimensional space and outputs the calculated optimal solutions and optimal objective function values. Thus for the purpose of visualization, the dimension n_x of x must be 1 (for two dimensional space) or 2 (for three dimensional space). And $n_y \in \{0, 1, 2, 3, \cdots\}$. The citation of 'PlotOVF.m' has forms as

```
out = PlotOVF(dim,funf,cong),
out = PlotOVF(dim,funf,cong,range),
out = PlotOVF(dim,funf,cong,range,nox).
```

Their inputs and outputs are described as in Table 5. Again we use Example 3.2 to illustrate the usage of `PlotOVF`. Type (or copy) the following codes in a new command window (or alternatively you can simply open `demonPlot.m` and run it):

```
clc; clear; close all;                                          % line 1
dim    = [1 1];                                                 % line 2
rng('default'); rng(1);                                         % line 3
Q      = randi(10,dim)/10;                                      % line 4
objf   = @(x,y)(sum(x.^2)-sum(x.*(Q*y))+sum(y.^2));             % line 5
cong   = @(x,y)[sum(x)+sum(y)-5; y.^2-25];                      % line 6
range  = [-10*ones(dim(1),1) 10*ones(dim(1),1)];                % line 7
out    = PlotOVF(dim,objf,cong,range,100);                      % line 8
```

The second line gives the dimensions `dim`$= [n_x\ n_y] = [1\ 1]$. The plot range of x-axis is $[-10\ 10]$ in line 7. The last input 100 in line 8 defines the number of x values in the range $[-10\ 10]$. Namely, there is 100 points x will be computed to derive $\psi(x)$ and the graph. Corresponding plots are presented in Figure 1, where the left (resp. right) sub-figure presents the optimal objective function value $\psi(x)$ (resp. optimal solution $y^*(x)$). Namely, $\psi(x) = f(x, y^*(x))$.

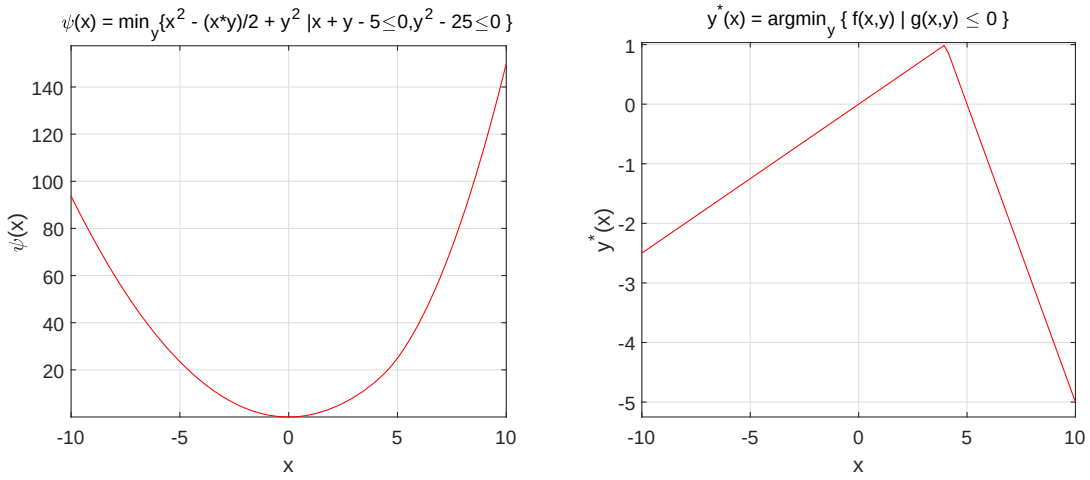

Figure 1: Optimal objective function value $\psi(x)$ and optimal solution $y^*(x)$ in $\mathbb{R}^2$.

One can also change the dimensions as `dim`$= [2\ 1]$ and

```
out = PlotOVF(dim,objf,cong,range,20);
```

This means there are 20 x values in the range $[-10\ 10]$ will be considered. Since $n_x =$`dim(1)`$= 2$, there are $400 = 20 \times 20$ points x in the square $[-10\ 10] \times [-10\ 10]$ will be computed to derive $\psi(x)$ and the graph (see Figure 2). It is worth mentioning that, in order to visualize, the graph of optimal solution $y^*(x)$ can be only plotted when $n_y = 1$. In other words, when $n_y \neq 1$, only one graph on $\psi(x)$ will be plotted.

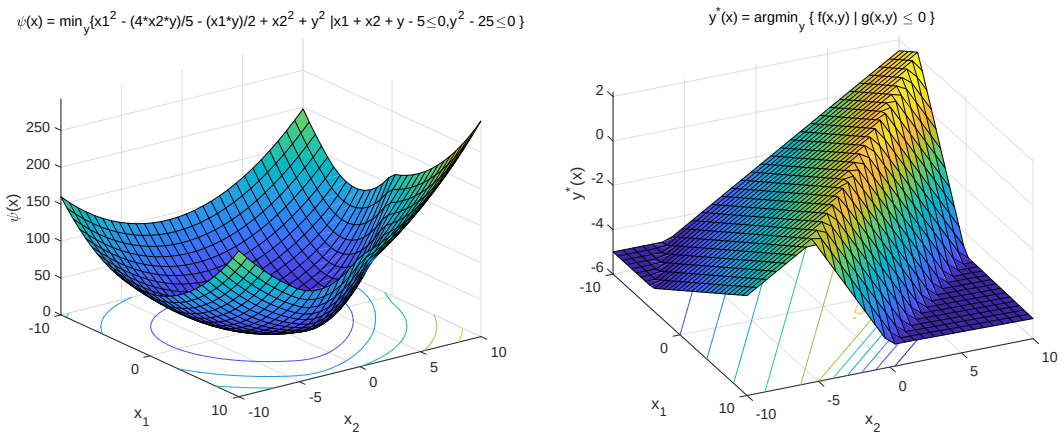

Figure 2: Optimal objective function value $\psi(x)$ and optimal solution $y^*(x)$ in $\mathbb{R}^3$.

4 Bilevel optimization examples from BOLIB

All examples are taken from BOLIB library [10]. Open folder `BOLIBExamples` which contains following files:

- One text file `readme.txt` describing how to use BOLIB library.
- Three Matlab m-files:
  - `startup.m`. Run this file to add the path to current folder and all sub-folders.
  - `demon1.m` demonstrating one way to call an example from BOLIB.
  - `demon2.m` demonstrating another way to call an example from BOLIB.
- One folder `Examples` in which there are some files as follows.
  - Folder `Nonlinear` containing 138 nonlinear bilevel optimization test examples;
  - Folder `Linear` containing 24 linear bilevel optimization test examples;
  - Folder `Simple` containing 11 Simple bilevel optimization test examples;
  - One Matlab m-file `InfomAllExamp.m` recording information of all 173 test examples, such as the dimensions, best known optimal upper and lower-level objective function values, or the starting points.

All examples are coded through Matlab and saved in m-files. The codes for each example share the same pattern so that they are easy to be called. We use files in folder `Nonlinear` to illustrate the creation of an m-file and its usage.

4.1 Nonlinear examples

`Nonlinear` folder contains 138 Matlab m-files. Each one specifies a nonlinear bilevel optimization test example, basically named by a combination of authors' surnames, year of publication, and when necessary, the order of the example in the corresponding reference,

see Figure 3. For example, as in following figure (showing a partial list of the examples), `AiyoshiShimizu1984Ex2.m` stands for the Example 2 considered by Aiyoshi and Shimizu in 1984, see [1] for more details.

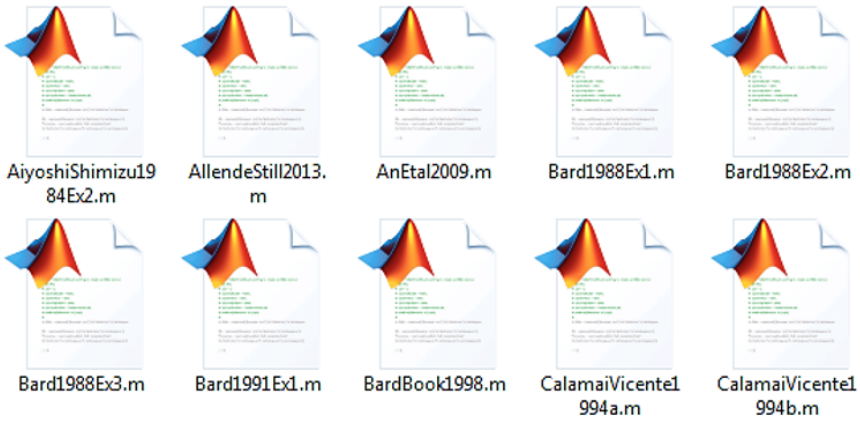

Figure 3: BOLIB examples.

Now we describe the inputs and outputs for all m-files which have a uniform citation form,

```
w = example_name(x,y,keyf,keyxy).
```

For the inputs, we have

$$\begin{aligned} \texttt{x} &\in \mathbb{R}^{n_x}, \\ \texttt{y} &\in \mathbb{R}^{n_y}, \\ \texttt{keyf} &\in \{\texttt{'F'},\texttt{'G'},\texttt{'f'},\texttt{'g'}\}, \\ \texttt{keyxy} &\in \{[\,],\texttt{'x'},\texttt{'y'},\texttt{'xx'},\texttt{'xy'},\texttt{'yy'}\}, \end{aligned}$$

where `'F'`, `'G'`, `'f'`, and `'g'` respectively stand for the four functions involved in (1.3). `'x'` and `'y'` represent the first order derivative with respect to x and y, respectively. Finally, `'xx'`, `'xy'`, and `'yy'` correspond to the second order derivative of the function F, G, f, and g, with respect to xx, xy, and yy, respectively. For the outputs, `w=example_name(x,y,keyf)` or `w=example_name(x,y,keyf,[])` returns the function value of `keyf`. When `keyxy` $\neq$ [], it returns the first or second order derivative of `keyf` with respect to choice of `keyxy` as described above. We summarize the input-inputs scenarios in the following table:

keyf\keyxy	[ ]	'x'	'y'	'xx'	'xy'	'yy'
'F'	$F(x,y)$	$\nabla_x F(x,y)$	$\nabla_y F(x,y)$	$\nabla^2_{xx} F(x,y)$	$\nabla^2_{xy} F(x,y)$	$\nabla^2_{yy} F(x,y)$
'G'	$G(x,y)$	$\nabla_x G(x,y)$	$\nabla_y G(x,y)$	$\nabla^2_{xx} G(x,y)$	$\nabla^2_{xy} G(x,y)$	$\nabla^2_{yy} G(x,y)$
'f'	$f(x,y)$	$\nabla_x f(x,y)$	$\nabla_y f(x,y)$	$\nabla^2_{xx} f(x,y)$	$\nabla^2_{xy} f(x,y)$	$\nabla^2_{yy} f(x,y)$
'g'	$g(x,y)$	$\nabla_x g(x,y)$	$\nabla_y g(x,y)$	$\nabla^2_{xx} g(x,y)$	$\nabla^2_{xy} g(x,y)$	$\nabla^2_{yy} g(x,y)$

Table 7: Inputs of `keyf` and `keyxy`

For the dimension of w in each scenario, see (2.1)–(2.2). If $n_G = 0$ (or $n_g = 0$), all outputs related to G (or g) should be empty, namely, $w = [\,]$. Let us look at some specific usage:

1) `w = example_name(x,y,'F')` or `w = example_name(x,y,'F',[])` returns the function value of F, i.e., $w = F(x, y)$; this is similar for G, f, and g;

2) `w = example_name(x,y,'F','x')` returns the partial derivative of F with respect to x, i.e., $w = \nabla_x F(x, y)$; this is similar for G, f, and g;

3) `w = example_name(x,y,'G','y')` returns the Jacobian matrix of G with respect to y, i.e., $w = \nabla_y G(x, y)$; this is similar for F, f, and g;

4) `w = example_name(x,y,'f', 'xy')` returns the Hessian matrix of f with respect to xy, i.e., $w = \nabla^2_{xy} f(x, y)$; this is similar for F, G, and g;

5) `w = example_name(x,y,'g','yy')` returns the second order derivative of g with respect to yy, i.e., $w = \nabla^2_{yy} g(x, y)$; this is similar for F, G, and f.

Example 4.1 *Shimizu et al. (1997), see [9], considered the bilevel program (1.3) with*

$$\begin{array}{rcl} F(x,y) & := & (x-5)^2 + (2y+1)^2, \\ f(x,y) & := & (y-1)^2 - 1.5xy, \\ g(x,y) & := & \begin{bmatrix} -3x+y+3 \\ x-0.5y-4 \\ x+y-7 \end{bmatrix}. \end{array}$$

Clearly, $n_x = 1, n_y = 1, n_G = 0, n_g = 3$. The m-file in `NonlinearExamples` *folder is named by* `ShimizuEtal1997a` *(i.e.,* `exmaple_name= ShimizuEtal1997a`*), which was coded through Matlab as in Table 8. If we are given some inputs (as in left column of the table below), then* `ShimizuEtal1997a` *will return us corresponding results as in the right column of Table 9.*

```
function w=ShimizuEtal1997a(x,y,keyf,keyxy)
if nargin<4 || isempty(keyxy)
    switch keyf
    case 'F'; w = (x-5)^2+(2*y+1)^2;
    case 'G'; w = [];
    case 'f'; w = (y-1)^2-1.5*x*y;
    case 'g'; w = [-3*x+y+3; x-0.5*y-4; x+y-7];
    end
else
    switch keyf
    case 'F'
        switch keyxy
        case 'x' ; w = 2*(x-5);
        case 'y' ; w = 4*(2*y+1);
        case 'xx'; w = 2;
        case 'xy'; w = 0;
        case 'yy'; w = 8;
        end
    case 'G'
       switch keyxy
        case 'x' ; w = [];
```

```
        case 'y' ; w = [];
        case 'xx'; w = [];
        case 'xy'; w = [];
        case 'yy'; w = [];
        end
    case 'f'
        switch keyxy
        case 'x' ; w = -1.5*y;
        case 'y' ; w = 2*(y-1)-1.5*x;
        case 'xx'; w = 0;
        case 'xy'; w = -1.5;
        case 'yy'; w = 2;
        end
    case 'g'
        switch keyxy
        case 'x' ; w = [-3;  1;  1];
        case 'y' ; w = [ 1;-0.5; 1];
        case 'xx'; w = [ 0;  0;  0];
        case 'xy'; w = [ 0;  0;  0];
        case 'yy'; w = [ 0;  0;  0];
        end
     end
end
end
```

Table 8: Function description of `ShimizuEtal1997a.m`.

Inputs	Outputs
`x = 4`	`x = 4`
`y = 0`	`y = 0`
`F = ShimizuEtal1997a(x,y,'F',[])`	`F = 2`
`Gy = ShimizuEtal1997a(x,y,'G','y')`	`Gy = []`
`fxy = ShimizuEtal1997a(x,y,'f','xy')`	`fxy = -1.5`
`gyy = ShimizuEtal1997a(x,y,'g','yy')`	`gyy = [0;0;0]`

Table 9: Outputs of `ShimizuEtal1997a.m`.

4.2 Linear examples

`Linear` folder contains 24 Matlab m-files. Each one specifies a linear bilevel optimization test example. We say a bilevel optimization problem (1.3) is linear if all its involved functions (F, G, f, g) are linear. Otherwise it is nonlinear. The rule of naming each example and the citation of each m-file are the same as those mentioned above for nonlinear examples.

4.3 Simple examples

Simple bilevel optimization is defined by

$$\begin{aligned} (4.1) \qquad & \min_{y} \quad F(y) \\ & \text{s.t.} \quad G(y) \le 0, \quad y \in S := \arg\min_{y} \{f(y) : \ g(y) \le 0\}. \end{aligned}$$

`Simple` folder contains 11 Matlab m-files. Each one specifies a simple bilevel optimization test example. The rule of naming each example and the citation of each m-file are the same as those mentioned above for nonlinear examples. Namely,

(4.2) `w = example_name(x,y,keyf,keyxy).`

As described in [10] that despite the lack of variable x in (4.1), for the sake of unifying the inputs of the function handle as in (4.2), we still treat it as an input. Here, for all simple bilevel examples, we input `x` as a scalar. In this way, `x` has no impact on the example itself. Now we use one example to illustrate this.

Example 4.2 *Franke et al. (2018), see [6], considered the bilevel program* (1.3) *with*

$$\begin{array}{rcl} F(y) & := & -y_2 \\ f(y) & := & y_3 \\ g(y) & := & \begin{bmatrix} y_1^2 - y_3 \\ y_1^2 + y_2^2 - 1 \\ -y_3 \end{bmatrix} \end{array}$$

Clearly, $n_y = 3, n_G = 0, n_g = 3$. *We let* $n_x = 1$. *The* ***m-file*** *is named by* `FrankeEtal2018Ex513` *(i.e.,* `exmaple_name` $=$ `FrankeEtal2018Ex513` *), which was coded through Matlab as follows.*

```
function w=FrankeEtal2018Ex513(x,y,keyf,keyxy)
if nargin<4 || isempty(keyxy)
    switch keyf
    case 'F'; w = -y(2);
    case 'G'; w = [];
    case 'f'; w = y(3);
    case 'g'; w = [y(1)^2-y(3); y(1)^2+y(2)^2-1; -y(3)];
    end
else
    switch keyf
    case 'F'
        switch keyxy
        case 'x' ; w = 0;
        case 'y' ; w = [0; -1; 0];
        case 'xx'; w = 0;
        case 'xy'; w = zeros(3,1);
        case 'yy'; w = zeros(3,3);
        end
    case 'G'
       switch keyxy
        case 'x' ; w = [];
        case 'y' ; w = [];
        case 'xx'; w = [];
        case 'xy'; w = [];
        case 'yy'; w = [];
        end
    case 'f'
        switch keyxy
        case 'x' ; w = 0;
        case 'y' ; w = [0; 0; 1];
```

```
        case 'xx'; w = 0;
        case 'xy'; w = zeros(3,1);
        case 'yy'; w = zeros(3,3);
        end
    case 'g'
        switch keyxy
        case 'x' ; w = zeros(3,1);
        case 'y' ; w = [2*y(1) 0 -1; 2*y(1) 2*y(2) 0; 0 0 -1];
        case 'xx'; w = zeros(3,1);
        case 'xy'; w = zeros(9,1);
        case 'yy'; w = [2 0 0;0 0 0;0 0 0;2 0 0; 0 2 0;zeros(4,3)];
        end
     end
end
end
```

5 BiOpt solvers: SNLLVF, SNKKT and SNQVI

In this section, we introduce three bilevel optimization solvers provided in this BiOpt toolboxes. They are SNLLVF, SNKKT and SNQVI, which are programmed based on semismooth Newton method. Three solves share similar algorithmic frameworks but are constructed from different perspectives. Details about their constructions can be found in [5, 12, 11], respectively. Their citations have the same format:

(5.1) `Out = solver_name(func,dim,pars),`

which is also specified as

```
Out = SNLLVF(func,dim,pars);
Out = SNKKT(func,dim,pars);
Out = SNQVI(func,dim,pars);
```

5.1 Description of inputs and outputs

In order to make use of those solvers, the first issue confronted us is the inputs and the output, which are described as in Table 11.

Inputs:	
`dim` :	A row/column vector with 4 elements, i.e., `dim`= $[n_x\ n_y\ n_G\ n_g]$.
`func` :	A function handle/a string must contain 4 functions $[F\ G\ f\ g]$,[required]
	preferably, including 1st and 2nd order derivatives of F, G, f, g, [optional]
	or 3rd order derivatives of f and g. [optional]
`pars` :	Starting point, parameters and other information.
	All are optional except for the last `pars.data`.
`pars.xy` :	The starting point for x and y, i.e.,`pars.xy`= $[x_0; y_0]$.
	It is a COLUMN vector whose dimension is $(n_x + n_y)$.
	Default one `pars.xy=zeros`$(n_x + n_y, 1)$.
`pars.lam` :	A positive penalty parameter, default one `pars.lam`=1.
	NOTE: it is an important parameter.
	Different choices may produce different results.

`pars.check`:	Check the completeness of all 1st,2nd order derivatives of F, G, f and g or 3rd order derivatives of f and g. Do (resp. do not) check if `pars.check=1`[default] (resp. =0).
`pars.iteron`:	Show results for each iteration if `pars.iteron`=1[default]. Don't show results for each iteration if `pars.iteron`=0.
`pars.maxit`:	Maximum iteration number, default one `pars.maxit`=2000.
`pars.tol`:	Tolerance for the stopping criteria, default one pars.tol=1e-8.
`pars.draw`:	A (resp. no) graph will be drawn if `pars.draw`=1 (resp. =0[default]).
`pars.data`:	This extra data is relied on the example that will be solved. If one example does not need extra data, then no need `pars.data`. Otherwise, input the data. The latter case means `nargin(func)=5`.
`pars.keep`:	This is only valid when `pars.check=1`. Keep (resp. delete) all calculated derivatives in `DerivativesFile` folder if `pars.keep=1` (resp.=0 [default]).
Outputs:	
`Out.x`:	Solution x.
`Out.y`:	Solution y.
`Out.F`:	Upper level objective function value.
`Out.G`:	Upper level constraint.
`Out.f`:	Lower level objective function value.
`Out.g`:	Lower level constraint.
`Out.time`:	CPU time.
`Out.iter`:	Number of iterations.
`Out.error`:	Error.

Table 11: Inputs and outputs of `SNLLVF`, `SNKKT` and `SNQVI`.

5.2 Examples and `func`

As shown in [5, 12, 11], besides using functions F, G, f and g themselves, `SNLLVF`, `SNKKT` and `SNQVI` make use of their different order derivatives, which are summarized in Table 12. All of them compute the 1st and 2nd order derivatives, whilst `SNKKT` need calculate the 3rd order derivatives of f and g, and `SNQVI` need the 3rd order derivative of f.

	SNLLVF	SNKKT	SNQVI
'F'	1st, 2nd	1st, 2nd	1st, 2nd
'G'	1st, 2nd	1st, 2nd	1st, 2nd
'f'	1st, 2nd	1st, 2nd, 3rd	1st, 2nd, 3rd
'g'	1st, 2nd	1st, 2nd, 3rd	1st, 2nd

Table 12: Oder of derivatives used by `SNLLVF`, `SNKKT` and `SNQVI`.

To solve a bilevel example, the input function handle (or a string) `func` in (5.1) has a unified format

```
func = @(x,y,keyf,keyxy)example_name(x,y,keyf,keyxy).
```

It should include information of at least four functions F, G, f and g, and preferably their derivatives based on Table 12. However, some examples have very complicated 2nd or 3rd order derivatives, which makes the construction of `func` difficult. Therefore, to ease the usage

of these three solvers, the construction of the input `func` is flexible. But the basic rule is that the more information you put into `func`, the much faster the solvers run. We explain this by using a few examples. Some have simple calculations of all derivatives and some involves functions with very complicated structures.

5.2.1 Examples with easy calculations of derivatives

Example 5.1 *Dempe and Dutta 2012 defined Example 2.4 [4] as follows*

$$\begin{aligned} F(x,y) &:= (x-1)^2+y^2 \\ f(x,y) &:= x^2y \\ g(x,y) &:= y^2. \end{aligned}$$

a) The first way. The simplest way to construct the file `func` is described in Table 13.

```
function w=DempeDutta2012Ex24_ver1(x,y,keyf,keyxy)
if nargin<4 || isempty(keyxy)
    switch keyf
    case 'F'; w = (x-1)^2 + y^2;
    case 'G'; w = [];
    case 'f'; w = x^2*y;
    case 'g'; w = y^2;
    end
end
```

Table 13: Function description of `DempeDutta2012Ex24_ver1.m`.

Note that in Table 13, despite that there is no G, we still need the information of G with defining $G = []$. Save this into a Matlab m-file and name it as `DempeDutta2012Ex24_ver1.m`. In this way, only functions themselves are given, while their derivatives are missing. However, as described before, `SNLLVF`, `SNKKT` and `SNQVI` need derivatives in Table 12. Therefore, to make solvers work normally, we need check the completeness of all derivatives of input functions. This can be done by setting

```
pars.check = 1.
```

Each solver has the ability to check the completeness of all derivatives if you set `pars.check=1`. Now to solve Example 5.1, type (or copy) the following codes in a new command window and run it (Or run `demon1stway.m` to see one of solvers to solve Example 5.1).

```
clc; clear; close all;                    % line 1
ExName     = 'DempeDutta2012Ex24_ver1';   % line 2
func       = str2func(ExName);            % line 3
dim        = [1 1 0 1];                   % line 4
pars.xy    = [1;1];                       % line 5
pars.lam   = 1;                           % line 6
pars.check = 1;                           % line 7
pars.keep  = 1;                           % line 8
Solvers    = {'SNLLVF','SNQVI','SNKKT'};  % line 9
SolNo      = 1;   % choose the solve'     % line 10
```

```
solver      = str2func(Solvers{SolNo});    % line 11
Out         = solver(func, dim,  pars);    % line 12
```

We would like to describe the function of each line in above codes.

- In line 3, it defines `func` as a function handle, namely,

  `func = @(x,y,keyf,keyxy)DempeDutta2012Ex24_ver1(x,y,keyf,keyxy).`

- In line 5, the starting points are given by `pars.xy`. This is optional.

- In line 6, `pars.lam` is an important parameter. Despite that `pars.lam` is an optional parameter with default value 1 (see Table 11), it is highly suggested that users adjust this parameter for different problems to pursuit better solutions.

- In line 7, since the file `DempeDutta2012Ex24_ver1.m` has incomplete information, we value `pars.check = 1` to check (or to calculate) all derivatives that will be used by one of solvers: `SNLLVF`, `SNKKT` and `SNQVI`.

- In line 8, `pars.keep = 1` means that all calculated derivatives' files will be kept after one solver solving the problem. If `pars.keep = 0`, then all calculated derivatives' files will be deleted. As described in Table 11, `pars.keep` only makes sense when `pars.check = 1`. In fact, if one m-file includes all derivatives, then there is no point to check the completeness and thus set `pars.check = 0`. Because of this, no derivatives' files will be created. By contrast, if one m-file has some derivatives missing, (e.g, $\nabla_x f$ is missing), then `pars.check = 1` reminds the solver to calculate $\nabla_x f$ and store it in a new m-file '`fx.m`'. After solving the problem, `pars.keep = 0` (resp. `pars.keep = 1`) makes the solver delete (resp. keep) the file '`fx.m`'.

- In lines 10 and 11, we choose the solve `SNLLVF` to solve this problem. One can change another solver just by altering `SolNo = 2` or `SolNo = 3` in line 10. In fact, to simplify the codes, one could replace lines $9-12$ by following line

  `Out1 = SNLLVF(func, dim,  pars);`

Of course, the checking procedure will take some time before it solve the example. So to save time, one also could construct the function file with putting more information, which gives rise to the second way.

b) The second way. This way is exact same as that defines examples in BOLIB (see Section 4). So the construction of the file `func` is presented in Table 15, where all 1st and 2nd order derivatives are given. In this way, according to Table 12, solver `SNLLVF` no longer need the checking procedure any more, but `SNKKT` (resp. `SNQVI`) still need compute the 3rd order derivatives of f and g (resp. f).

```
function w=DempeDutta2012Ex24_ver2(x,y,keyf,keyxy)
if nargin<4 || isempty(keyxy)
    switch keyf
    case 'F'; w = (x-1)^2+y^2;
```

```
    case 'G'; w = [];
    case 'f'; w = x^2*y;
    case 'g'; w = y^2;
    end
else
    switch keyf
    case 'F'
        switch keyxy
        case 'x' ; w = 2*(x-5);
        case 'y' ; w = 2*y;
        case 'xx'; w = 2;
        case 'xy'; w = 0;
        case 'yy'; w = 2;
        end
    case 'G'
       switch keyxy
        case 'x' ; w = [];
        case 'y' ; w = [];
        case 'xx'; w = [];
        case 'xy'; w = [];
        case 'yy'; w = [];
        end
    case 'f'
        switch keyxy
        case 'x' ; w = 2*x*y;
        case 'y' ; w = x^2;
        case 'xx'; w = 2*y;
        case 'xy'; w = 2*x;
        case 'yy'; w = 2;
        end
    case 'g'
        switch keyxy
        case 'x' ; w = 0;
        case 'y' ; w = 2*y;
        case 'xx'; w = 0;
        case 'xy'; w = 0;
        case 'yy'; w = 2;
        end
     end
end
end
```

Table 15: Function description of `DempeDutta2012Ex24_ver2.m`.

To solve Example 5.1, type/copy the following codes in a new command window and run it (Or run `demon2ndway.m` to see one of solvers to solve Example 5.1).

```
clc; clear; close all;                  % line 1
func       = 'DempeDutta2012Ex24_ver2'; % line 2
dim        = [1 1 0 1];                 % line 3
pars.check = 0;                         % line 4
Out1       = SNLLVF(func, dim, pars);   % line 5
```

```
pars.check = 1;                          % line 6
pars.keep  = 0;                          % line 7
Out2       = SNQVI(func, dim, pars);     % line 8
```

Here, line 2 defines `func` as a string, which is also allowed (based on Table 11) as long as the there is a Matlab m-file with name `DempeDutta2012Ex24_ver2.m`. Since the m-file has all 1st and 2nd order derivatives but without 3rd order derivatives of f and g, we put `pars.check = 0` for `SNLLVF` while set `pars.check = 1` for `SNQVI`.

c) The third way. This way to construct the function file is to input all information described in Table 12. The construction of the function is presented in Table 18. Note that in Table 18, if the 3rd derivative is a zero scalar (or matrix), we could just set it as an empty variable to save the storage of the computer. To solve Example 5.1, type (or copy) the following codes in a new command window and run it (Or run `demon3rdway.m` to see solvers to solve Example 5.1).

```
clc; clear; close all;                   % line 1
func       = 'DempeDutta2012Ex24_ver3';  % line 2
dim        = [1 1 0 1];                  % line 3
pars.check = 0;                          % line 4
Out1       = SNLLVF(func, dim, pars);    % line 5
Out2       = SNQVI(func, dim, pars);     % line 6
Out3       = SNKKT(func, dim, pars);     % line 7
```

Since `DempeDutta2012Ex24_ver3.m` has all derivatives in Table 12, we set `pars.check = 0` for `SNLLVF`, `SNKKT` and `SNQVI` to remind them to skip the checking procedure, which of course fastens the computation.

```
function w=DempeDutta2012Ex24_ver3(x,y,keyf,keyxy)
if nargin<4 || isempty(keyxy)
    switch keyf
    case 'F'; w = (x-1)^2+y^2;
    case 'G'; w = [];
    case 'f'; w = x^2*y;
    case 'g'; w = y^2;
    end
else
    switch keyf
    case 'F'
        switch keyxy
        case 'x' ; w = 2*(x-5);
        case 'y' ; w = 2*y;
        case 'xx'; w = 2;
        case 'xy'; w = 0;
        case 'yy'; w = 2;
        end
    case 'G'
       switch keyxy
        case 'x' ; w = [];
```

```
        case 'y' ; w = [];
        case 'xx'; w = [];
        case 'xy'; w = [];
        case 'yy'; w = [];
        end
    case 'f'
        switch keyxy
        case 'x' ; w = 2*x*y;
        case 'y' ; w = x^2;
        case 'xx'; w = 2*y;
        case 'xy'; w = 2*x;
        case 'yy'; w = 2;
        case 'yxx'; w = 2;
        case 'yxy'; w = 0;
        case 'yyy'; w = 0;
        end
    case 'g'
        switch keyxy
        case 'x' ; w = 0;
        case 'y' ; w = 2*y;
        case 'xx'; w = 0;
        case 'xy'; w = 0;
        case 'yy'; w = 2;
        case 'yxx'; w = [];
        case 'yxy'; w = [];
        case 'yyy'; w = [];
        end
     end
end
end
```

Table 18: Function description of `DempeDutta2012Ex24_ver3.m`.

5.2.2 Examples with complicated calculations of derivatives

When the example involves some functions with complicated calculations of derivatives, `SNLLVF`, `SNKKT` and `SNQVI` also enable users to create the input function m-file in an easier way.

d) The fourth way. In this way, users could input simple derivatives of some functions while leave complicated derivatives blank. And those blank derivatives will be calculated by solvers themselves.

Example 5.2 *Lu, Deb and Sinha [8] defined Example* `LuDebSinha2016b_ver4` *by,*

$$\begin{aligned} F(x,y) &:= (x-0.5)^2+(y-1)^2 \\ G(x,y) &:= [-x, x-1, -y, y-2]^\top \\ f(x,y) &:= 2-\exp\left[-\left(\frac{1.5y-x}{0.055}\right)^{0.4}\right]-0.8\exp\left[-\left(\frac{2y-3+x}{0.5}\right)^2\right] \end{aligned}$$

This example has three simple functions F, G and g, while has a complicated one f. The construction of the input file `func` can be done by Table 19.

```
function w = SinhaMaloDeb2014TP9_ver4(x,y,keyf,keyxy)
if nargin<4 || isempty(keyxy)
    switch keyf
    case 'F'; w = (x-0.5)^2+(y-1)^2;
    case 'G'; w = [-x; x-1; -y; y-2];
    case 'f'; a = (1.5*y-x)/0.055;'
              w = 2-exp(-a^0.4 )-0.8*exp(-( (2*y+x-3)/0.5 )^2 );
    case 'g'; w = [];
    end
else
    switch keyf
    case 'F'
        switch keyxy
        case 'x' ; w = 2*(x-0.5);
        case 'y' ; w = 2*(y-1);
        case 'xx'; w = 2;
        case 'xy'; w = 0;
        case 'yy'; w = 2;
        end
    case 'G'
       switch keyxy
        case 'x' ; w = [-1;1;0;0];
        case 'y' ; w = [ 0;0;-1;1];
        case 'xx'; w = zeros(4,1);
        case 'xy'; w = zeros(4,1);
        case 'yy'; w = zeros(4,1);
        end
    case 'g'
       switch keyxy
        case 'x' ;  w = [];
        case 'y' ;  w = [];
        case 'xx';  w = [];
        case 'xy';  w = [];
        case 'yy';  w = [];
        case 'yxx'; w = [];
        case 'yxy'; w = [];
        case 'yyy'; w = [];
        end
     end
end
end
```

Table 19: Function description of `SinhaMaloDeb2014TP9.m`.

To solve Example 5.2, type/copy the following codes in a new command window and run it (Or run `demon4thway.m` to see one of solvers to solve Example 5.2).

```
clc; clear; close all;                  % line 1
func       = 'LuDebSinha2016b_ver4';    % line 2
dim        = [1 1 4 0];                 % line 3
pars.xy    = [1;1];                     % line 4
pars.lam   = 1;                         % line 5
```

```
pars.check = 1;                              % line 6
SolNo      = 2;   % choose the solve'       % line 7
Solvers    = {'SNLLVF','SNQVI','SNKKT'};     % line 8
solver     = str2func(Solvers{SolNo});       % line 9
Out        = solver(func, dim,  pars);       % line 10
```

Since the m-file has missing derivatives of f in Table 19, we need set `pars.check = 1` to complete them. Note that solvers will only complete the derivatives of f since all required derivatives of F, G and g are existed.

5.3 Examples with parameters or extra data

As mentioned above, four ways to construct the m-file share the similar citation format, i.e.,

`func = @(x,y,keyf,keyxy)example_name(x,y,keyf,keyxy).`

There are four inputs `x`, `y`, `keyf` and `keyxy`. However, apart from containing these four inputs, some examples also involve parameters or extra data. In such case, one could construct the m-file with the citation format,

(5.2) `func = @(x,y,keyf,keyxy)example_name(x,y,keyf,keyxy,data).`

Here, `data` should be defined or given before it is used. Then put the data into `pars` as

`pars.data = data.`

Note that if there is `data` that is substituted in (5.2), then this `data` must be integrated into `pars`, namely, `pars.data = data`. We use two examples to demonstrate how `SNLLVF`, `SNKKT` and `SNQVI` to solve problems with parameters or extra data. Or, alternatively, open folder `BiOpt-Solvers` and run Matlab m-file `demon5thway.m` to see this.

Example 5.3 *Henrion and Surowiec 2011 [7] defined one example as follows,*

$$\begin{aligned} F(x,y) &:= x^2 + cy \\ f(x,y) &:= 0.5y^2 - xy, \end{aligned}$$

where c is a parameter.

A standard way is to construct the m-file that is similar to `DempeDutta2012Ex24_ver1.m` in Table 13, or `DempeDutta2012Ex24_ver2.m` in Table 15 or `DempeDutta2012Ex24_ver3.m` in Table 18. But c should be given inside, see Table 21.

```
function w = HenrionEtal2011(x,y,keyf,keyxy)
c = 1;
if nargin<4 || isempty(keyxy)
    switch keyf
    case 'F'; w = x^2 + c*y;
    case 'G'; w = [];
    case 'f'; w = 0.5*y^2-x*y;
    case 'g'; w = [];
```

```
    end
end
```

Table 21: Function description of `HenrionSurowiec2011.m`.

e) The fifth way. However, it is quite inconvenient that changing the value of c because every time you need open the file `HenrionEtal2011.m` to alter c. An easy operation is to construct the m-file as in Table 22.

```
function w = HenrionEtal2011_ver5(x,y,keyf,keyxy,c)
if nargin<4 || isempty(keyxy)
    switch keyf
    case 'F'; w = x^2 + c*y;
    case 'G'; w = [];
    case 'f'; w = 0.5*y^2-x*y;
    case 'g'; w = [];
    end
end
```

Table 22: Function description of `HenrionSurowiec2011_ver5.m`.

Then, to solve Example 5.3, type (or copy) the following codes in a new command window and run it (Or run `demon5thway.m` to see one of solvers to solve Example 5.3). Line 3 redefines the function `HenrionEtal2011_ver5` with 5 inputs as a new function `func` with 4 inputs. The given data is also integrated into `pars` by line 6. Clearly, in this way, altering c becomes very easy and convenient.

```
clc; clear; close all;                                                    % line 1
c          = 1;                                                           % line 2
func = @(x,y,keyf,keyxy)HenrionEtal2011_ver5(x,y,keyf,keyxy,c);           % line 3
dim        = [1 1 0 0];                                                   % line 4
pars.check = 1;                                                           % line 5
pars.data  = c;                                                           % line 6
Out        = SNLLVF(func, dim,  pars);                                    % line 7
```

The second example involves amounts of data. To make the function construction more clear, it is highly suggested that one separate the data and the function.

Example 5.4 *An et al. 2009 [2] defined one example as follows,*

$$\begin{array}{rcl} F(x,y) & := & \frac{1}{2}z^\top Hz + c^\top z \\ G(x,y) & := & \begin{bmatrix} -x \\ -y \\ Ax+By+d \end{bmatrix} \\ f(x,y) & := & y^\top Px + \frac{1}{2}y^\top Qy + q^\top y \\ g(x,y) & := & Dx + Ey + b \end{array}$$

where $z = (x^\top, y^\top)^\top$ *and* H, c, A, B, d, P, Q, q, D, E, b *respectively are given data.*

First, we create a file `AnEtal2009Data.m` to store all data: H, c, A, B, d, P, Q, q, D, E, b. The corresponding codes are presented in Table 24.

```
function Data = AnEtal2009_data
Data.H = [-3.8  4.4  1.2 -2.2;  4.4 -2.2  0.6  1.8;
           1.2  0.6  0.0  0.4; -2.2  1.8  0.4  0.0];
Data.c = [935.74474; 87.53654; 121.96196; 299.24825];
Data.A = [0.00000 3.88889;-2.00000 8.77778];
Data.B = [4.88889 7.44444; -5.11111 0.88889];
Data.d = [-61.57778; -0.80000];
Data.P = [-17.85000 6.57500; 30.32500 30.32500];
Data.Q = [21.10204,11.81633;11.81633,-14.44898];
Data.q = [-18.21053;13.05263];
Data.D = [5.00000 7.44444; -8.33333 3.00000; -8.66667 -8.55556; 6.44444 -5.11111];
Data.E = [3.88889 1.77778;  6.88889 6.11111; -5.33333 -7.00000; 1.44444  4.44444];
Data.b = [-39.62222;-60.00000;72.37778;-17.28889];
end
```

Table 24: Function description of `AnEtal2009_data.m`.

Then create the function file `AnEtal2009_ver5.m` to define the Example 5.4 as in Table 25.

```
function w=AnEtal2009_ver5(x,y,keyf,keyxy,data)
if nargin<4 || isempty(keyxy)
    switch keyf
    case 'F';  w = [x' y']*(Data.H*[x; y]/2+Data.c);
    case 'G';  w = [-x;-y;Data.A*x+Data.B*y+Data.d];
    case 'f';  w = y'*(Data.P*x+Data.q)+y'*Data.Q*y/2;
    case 'g';  w = Data.D*x+Data.E*y+Data.b;
    end
else
    switch keyf
    case 'F'
        z    = Data.H*[x; y]+Data.c;
        switch keyxy
        case 'x' ; w = z(1:2,:);
        case 'y' ; w = z(3:4,:);
        case 'xx'; w = Data.H(1:2,1:2);
        case 'xy'; w = Data.H(3:4,1:2);
        case 'yy'; w = Data.H(3:4,3:4);
        end
    case 'G'
        switch keyxy
        case 'x' ; w = [-eye(2);zeros(2);Data.A];
        case 'y' ; w = [zeros(2);-eye(2);Data.B];
        case 'xx'; w = zeros(12,2);
        case 'xy'; w = zeros(12,2);
        case 'yy'; w = zeros(12,2);
        end
 case 'f'
        switch keyxy
        case 'x' ;  w = Data.P'*y;
        case 'y' ;  w = Data.P*x+Data.q+(Data.Q+Data.Q')*(y/2);
        case 'xx';  w = zeros(2);
```

```
        case 'xy';  w = Data.P;
        case 'yy';  w = (Data.Q+Data.Q')/2;
        case 'yxx'; w = [];
        case 'yxy'; w = [];
        case 'yyy'; w = [];
        end
 case 'g'
        switch keyxy
        case 'x' ;  w = Data.D;
        case 'y' ;  w = Data.E;
        case 'xx';  w = zeros(8,2);
        case 'xy';  w = zeros(8,2);
        case 'yy';  w = zeros(8,2);
        case 'yxx'; w = [];
        case 'yxy'; w = [];
        case 'yyy'; w = [];
        end
     end
end
end
```

Table 25: Function description of `AnEtal2009_data.m`.

Finally, to solve Example 5.4, type (or copy) the following codes in a new command window and run it (Or run `demon5thway.m` to see one of solvers to solve Example 5.4). Line 3 redefines the function `AnEtal2009_ver5` with 5 inputs as a new function `func` with 4 inputs. The given data is also integrated into `pars` by line 7. Clearly, in this way, altering `data` becomes very convenient.

```
clc; clear; close all;                                                  % line 1
data       = AnEtal2009_data;                                           % line 2
func = @(x,y,keyf,keyxy)AnEtal2009_ver5(x,y,keyf,keyxy,data);           % line 3
dim        = [2 2 6 4];                                                 % line 4
pars.xy    = [1;1;1;1];                                                 % line 5
pars.lam   = 1;                                                         % line 6
pars.data  = data;                                                      % line 7
Out        = SNLLVF(func, dim, pars);                                   % line 8
```

5.4 Importance of `pars.xy` and `pars.lam`

As presented in Table 11, two parameters: the starting points `pars.xy` and the penalty parameter `pars.lam` would have an influence on the performance of `SNLLVF`, `SNKKT` and `SNQVI`. The default one for the starting points is zero, namely, `pars.xy = 0`. While, since bilevel optimisation is often non-convex and the three solvers are based on semi-smooth Newton method whose performance generally relies on the starting points, it is better to set a good `pars.xy` rather than always being 0.

In addition, according to [5, 12, 11], the construction of `SNLLVF`, `SNKKT` and `SNQVI` involves an important penalty parameter, which somewhat decides the performance of those solvers. For instance, we apply them into solving Example `Colson2002BIPA4` (defined in [3]) from

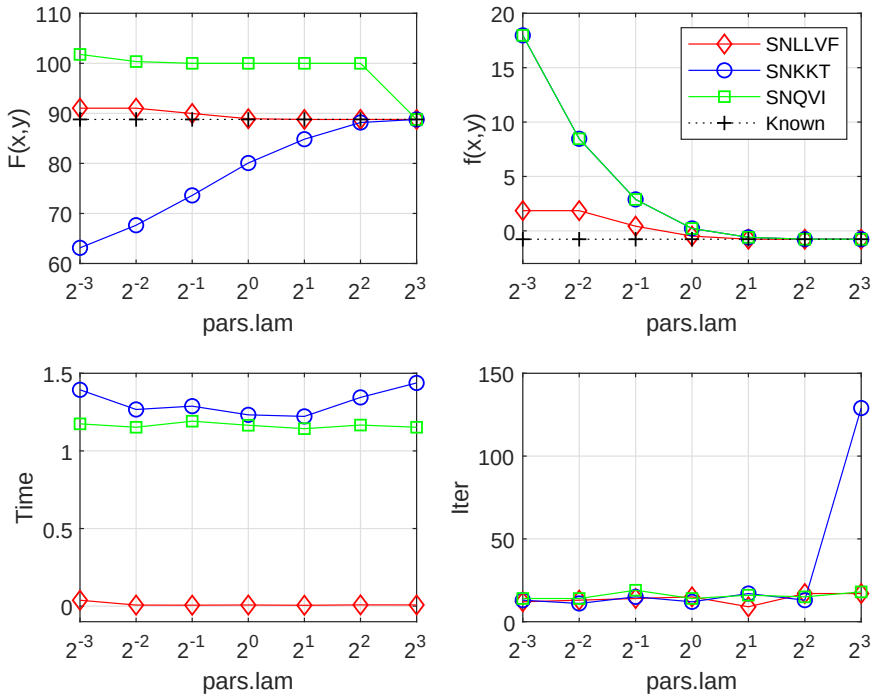

Figure 4: Performance of `SNLLVF`, `SNKKT` and `SNQVI` on solving example `Colson2002BIPA4`.

`Nonlinear` in BOLIB under different `pars.lam`. Results are reported in figure 4. Obviously, `pars.lam` more or less has an impact on their performance.

5.5 Summary

To end this section, we would like to summarize some key rules of using `BiOpt` solvers.

1) `SNLLVF` generally performs faster than `SNKKT` and `SNQVI` because it only needs 1st and 2nd order derivatives of involved functions, while the latter two still need 3rd order derivatives of f or g.

2) Even though it is very flexible to create the m-file of an example, such as ways in Tables 13, 15, 18 or 19, it is highly suggested to use the way as in Table 18 where all order derivatives described in Table 12 are given. Because in this way, each solver do not need to check the completeness and thus will save computational time.

3) When users definitely ensure that the created m-file covers all derivatives described in Table 12, it could set `pars.check = 0`. However, for this case, setting `pars.check = 1` actually will not take much time to check the completeness because it is already complete, which means always setting `pars.check = 1` is a good choice or just in case there are some derivatives are missing. This would make solvers solving problems more stably, namely, with less errors.

4) Adjusting the parameter `pars.lam` or choosing proper starting points `pars.xy` would render solvers better performance in terms of computational time or quality of solutions.

References

[1] E. Aiyoshi and K. Shimizu, A solution method for the static constrained Stackelberg problem via penalty method, IEEE Transactions on Automatic Control, 29, 1111-1114, 1984.

[2] L.T.H. An, P.D. Tao, N.N. Canh and N.V. Thoai, DC programming techniques for solving a class of nonlinear bilevel programs, Journal of Global Optimization, 44(3), 313-337, 2009.

[3] B. Colson, BIPA (Bilevel Programming with Approximation Methods): Software guide and test problems, Technical report, 2002.

[4] S. Dempe and J. Dutta, Is bilevel programming a special case of a mathematical program with complementarity constraints? Mathematical programming, 131(1-2), 37-48, 2012.

[5] A. Fischer A.B. Zemkoho and S. Zhou, Semismooth Newton-type method for bilevel optimization: Global convergence and extensive numerical experiments, Technical Report, 2019.

[6] S. Franke, P. Mehlitz and M. Pilecka, Optimality conditions for the simple convex bilevel programming problem in Banach spaces, Optimization, 67:2, 237-268, 2018.

[7] R. Henrion and T. Surowiec, On calmness conditions in convex bilevel programming, Applicable Analysis, 90(6), 951-970, 2011.

[8] Z.C. Lu, K. Deb, and A. Sinha, Robust and reliable solutions in bilevel optimization problems under uncertainties, COIN Report 2016026, Retrived on 19 November 2017 from http://www.egr.msu.edu/ kdeb/papers/c2016026.pdf.

[9] K. Shimizu, Y. Ishizuka and J.F. Bard, Nondifferentiable and two-level mathematical programming, Dordrecht: Kluwer Academic Publishers, 1997.

[10] S. Zhou, A.B. Zemkoho, and A. Tin. BOLIB 2019: Bilevel Optimization Library of Test Problems Version 2. available at https://www.researchgate.net/publication/325120369, 2019.

[11] S. Zhou, and A.B. Zemkoho. Value Function Approach to Quasi-variational Inequalities with Applications to Lipschitzian Stability and Numerical Methods for Optimization Problems. Technical report, 2019.

[12] S. Zhou, and A.B. Zemkoho. Theoretical and numerical comparison of the Karush-Kuhn-Tucker and value function reformulations in bilevel optimization. Technical report, 2019.