MOBO: Documentation

Carolin Kästner, Ernest Quintana-Aparicio

December 16, 2020

1 Introduction

The goal of the MOBO-Algorithm is to get an approximation of the minimal solutions of **multiobjective bilevel optimization problems** of the form

$$\begin{aligned} \min_{y} \quad & F(x,y) \\ \text{s.t.} \quad & x \in \Psi(y) \\ & \tilde{G}(y) \leq 0 \end{aligned} \tag{ULP}$$

with Ψ being the set-valued mapping defined by

$$\Psi(y) = \operatorname*{argmin}_{x} \{ f(x,y) \,|\, G(x,y) \leq 0 \}, \tag{LLP(y)}$$

$x \in \mathbb{R}^{n_1}, y \in \mathbb{R}^{n_2}, f : \mathbb{R}^{n_1 \times n_2} \to \mathbb{R}^{m_1}, F : \mathbb{R}^{n_1 \times n_2} \to \mathbb{R}^{m_2}, G : \mathbb{R}^{n_1 \times n_2} \to \mathbb{R}^{p}, \tilde{G} : \mathbb{R}^{n_2} \to \mathbb{R}^{\tilde{p}}$, where f and G should be two times continuously differentiable.
A *minimial solution* $(\bar{x}, \bar{y})$ in the sense of multiobjective optimization means that $(\bar{x}, \bar{y})$ is feasible and there exists no other feasible point (x, y) with

$$F(x,y) \leq F(\bar{x}, \bar{y}) \text{ and } F(x,y) \neq F(\bar{x}, \bar{y}),$$

where $\leq$ is meant component-wise.

The problem (ULP) is also called *upper level problem*, (LLP(y)) *lower level problem*; therefore x is the *lower level variable* and y is the *upper level variable*.

This bilevel optimization problem is restricted to functions $\tilde{G}$ that are independent of x. For using the program it is also necessary to declare a box that contains all y satisfying $\tilde{G}(y) \leq 0$ — this is discussed more precisely in Section 2.2.2.

The underlying algorithm of MOBO was introduced by Eichfelder in [1]. All references in the code refer to this paper.
The basic idea is to approximate the feasible set of (ULP) and to find the minimal points of these approximation points with a modified Jahn-Graef-Younes method. In the following steps the approximation of the feasible set of (ULP) is refined regarding the surrounding of the minimal points and then the minimal points of this approximation are calculated. This procedure is repeated multiple times.

MOBO generalizes the algorithm in [1] as proposed by Eichfelder: y is not restricted to be one dimensional, the set $\{y \,|\, \tilde{G}(y) \leq 0\}$ does not need to be a box (but still bounded) and f can be more than two dimensional or even one dimensional.
In addition a modified Graef-Younes method is used to determine the minimal points w.r.t. (ULP). In this modified method one presorts the function values of F with the help of an increasing function φ. MOBO uses $\varphi(x) = \mathbb{1}^\top x$ with $\mathbb{1}$ being the all-one-vector. For further clarification see Algorithm 4 in [2].

MOBO is written in MATLAB. Since `fmincon` is used, running MOBO requires the Optimization Toolbox.

1.1 Assumptions on (LLP(y))

As stated above, f and G are supposed to be two times differentiable. Still, there are some further assumptions on those functions in order that the theoretical results of [1] can be applied.

As the algorithm solves multiple times the Pascoletti-Serafini scalarization (see [3]) of (LLP(y)), $\tilde{a} = y$:

$$\begin{aligned} \min_{t,\,x} \quad & t \\ \text{s.t.} \quad & -(a + tr - f(x, \tilde{a})) \leq 0 \\ & G(x, \tilde{a}) \leq 0 \\ & t \in \mathbb{R}, \end{aligned} \tag{SP($a, r, \tilde{a}$)}$$

assumptions regarding the constraints and their Lagrange multipliers $\mu \in \mathbb{R}^{m_1}$, $\nu \in \mathbb{R}^p$ occur. For each $\tilde{a} \in \mathbb{R}^{n_2}$ that satisfies $\tilde{G}(\tilde{a}) \leq 0$ and the minimal solution (t^0, x^0) of (SP($a, r, \tilde{a}$)) with parameters $a \in \mathbb{R}^{m_1}$ and $r = e_{m_1} \in \mathbb{R}^{m_1}$ (the m_1-th unit vector), the following conditions must be fulfilled:

- $\forall\, i \in \{1, \ldots, m_1\} : a_i + tr_i - f_i(x, \tilde{a}) > 0$ or $\mu_i > 0$ (non-degeneracy),
- $\forall\, j \in \{1, \ldots, p\} : G_j(x, \tilde{a}) < 0$ or $\nu_j > 0$ (non-degeneracy),
- the vectors $\begin{pmatrix} r_i \\ -\nabla_x f_i(x^0, \tilde{a}) \end{pmatrix}$, $i \in I^+ := \{i \in \{1, \ldots, m_1\} \,|\, \mu_i > 0\}$ and $\begin{pmatrix} 0 \\ -\nabla_x G_j(x^0, \tilde{a}) \end{pmatrix}$, $j \in J^+ := \{j \in \{1, \ldots, p\} \,|\, \nu_j > 0\}$ are linearly independent,
- $\exists\, \xi > 0 : (t, x^\top)(\mathbf{H}\mathscr{L})_{t,x}(t^0, x^0, \mu, \nu, a, r) \begin{pmatrix} t \\ x \end{pmatrix} \geq \xi \left|\left| \begin{pmatrix} t \\ x \end{pmatrix} \right|\right|^2$
  $\forall\, (t, x) \in \{(t, x) \in \mathbb{R}^{1+n_1} \,|\, r_i t = \nabla_x f_i(x^0, \tilde{a})^\top x \;\forall i \in I^+,\; \nabla_x G_j(x^0, \tilde{a})^\top x = 0 \;\forall j \in J^+\}$
  where $(\mathbf{H}\mathscr{L})_{t,x}$ is the Hessian matrix of the Lagrange function of (SP($a, r, \tilde{a}$)) w.r.t. t and x.

Eventually, one last an assumption concerning scalar optimization problems is required:

- $\min_x\{f_i(x, y) \,|\, G(x, y) \leq 0\}$ is solvable for arbitrary but fixed $y \in \{y \,|\, \tilde{G}(y) \leq 0\}$ and all $i \in \{1, \ldots, m_1\}$. This holds, in particular, if $\{(x, y) \in \mathbb{R}^{n_1+n_2} \,|\, G(x, y) \leq 0\}$ is non-empty and compact.

2 Using the program

2.1 Declaring the model

In order to declare a problem the user must create a `.m`-file where all functions F, f, G and $\tilde{G}$ are saved as column vectors. Also, the Jacobian and Hessian matrix of f and G should be provided as defined below (analogue for derivatives with respect to y and the function G). It is not necessary to put in the derivatives of F and $\tilde{G}$.

$$f(x,y) = \begin{pmatrix} f_1(x,y) \\ \vdots \\ f_{m_1}(x,y) \end{pmatrix} \in \mathbb{R}^{m_1}$$

$$(\mathbf{J}_f(x,y))_x = \begin{pmatrix} \frac{\partial}{\partial x_1} f_1(x,y) & \cdots & \frac{\partial}{\partial x_{n_1}} f_1(x,y) \\ \vdots & \ddots & \vdots \\ \frac{\partial}{\partial x_1} f_{m_1}(x,y) & \cdots & \frac{\partial}{\partial x_{n_1}} f_{m_1}(x,y) \end{pmatrix} \in \mathbb{R}^{m_1 \times n_1}$$

$$(\mathbf{H}f(x,y))_{xx} = \begin{pmatrix} \frac{\partial^2}{\partial x_1 \partial x_1} f_1(x,y) & \cdots & \frac{\partial^2}{\partial x_{n_1} \partial x_1} f_1(x,y) \\ \vdots & \ddots & \vdots \\ \frac{\partial^2}{\partial x_1 \partial x_{n_1}} f_1(x,y) & \cdots & \frac{\partial^2}{\partial x_{n_1} \partial x_{n_1}} f_1(x,y) \\ \frac{\partial^2}{\partial x_1 \partial x_1} f_2(x,y) & \cdots & \frac{\partial^2}{\partial x_{n_1} \partial x_1} f_2(x,y) \\ \vdots & \ddots & \vdots \\ \frac{\partial^2}{\partial x_1 \partial x_{n_1}} f_{m_1}(x,y) & \cdots & \frac{\partial^2}{\partial x_{n_1} \partial x_{n_1}} f_{m_1}(x,y) \end{pmatrix} \in \mathbb{R}^{m_1 n_1 \times n_1}$$

$$(\mathbf{H}f(x,y))_{xy} = \begin{pmatrix} \frac{\partial^2}{\partial x_1 \partial y_1} f_1(x,y) & \cdots & \frac{\partial^2}{\partial x_{n_1} \partial y_1} f_1(x,y) \\ \vdots & \ddots & \vdots \\ \frac{\partial^2}{\partial x_1 \partial y_{n_2}} f_1(x,y) & \cdots & \frac{\partial^2}{\partial x_{n_1} \partial y_{n_2}} f_1(x,y) \\ \frac{\partial^2}{\partial x_1 \partial y_1} f_2(x,y) & \cdots & \frac{\partial^2}{\partial x_{n_1} \partial y_1} f_2(x,y) \\ \vdots & \ddots & \vdots \\ \frac{\partial^2}{\partial x_1 \partial y_{n_2}} f_{m_1}(x,y) & \cdots & \frac{\partial^2}{\partial x_{n_1} \partial y_{n_2}} f_{m_1}(x,y) \end{pmatrix} \in \mathbb{R}^{m_1 n_2 \times n_1}$$

These definitions follow the style proposed in the bilevel optimization library [4]. Unfortunately, a different notation for (ULP) and (LLP(y)) is used there and only one dimensional objective functions of the optimization problems are valid for which they provide the gradients, not the Jacobian matrix. For this purpose, a transformation function (`transformBOLIB.m`) is given, so that the user can still try those examples. Just set `BOLIB = true` in `UserFile.m`.
Note that, in [4] problems are allowed, where the function $\tilde{G}$ depends on x. Such problems are not solvable with MOBO. The program tests this before running the solver; if $\tilde{G}$ depends on x, an error is returned.
Another restriction of MOBO in contrast to [4] is that $n_2 \geq 1$ and $\tilde{p} \geq 1$ are required. The reason for this is explained in Section 2.2.2. If this condition is violated, an error is issued.

The `.m`-file for the model should contain a function depending on x, y, a key for the function name (e.g. "keyf", taking values `'f'`, `'F'`, `'G'`, `'G_tilde'`) and an optional key for the derivative (e.g. "keyxy" taking values `'x'` (for $(\mathbf{J}_f)_x$), `'y'`, `'xx'`(for $(\mathbf{H}f)_{xx}$),

'xy', 'yy'). It is recommended to complete the `model_dummy.m`-file with the values of the individual problem.

Example

Consider the following multiobjective bilevel problem

$$\begin{aligned} \min_{y} \quad & \begin{pmatrix} x_1 + y_1 \\ y_2 \end{pmatrix} \\ \text{s.t.} \quad & x \in \operatorname*{argmin}_{x} \{ x_1^2 + x_2^2 y_2 - \frac{1}{3} y_1^3 \,|\, x_1^2 + y_1 \leq 0,\ y_1 y_2 \leq 0 \} \\ & 0 \leq y_1 \leq 10 \\ & 0 \leq y_2 \leq 10 \end{aligned}$$

Here, the functions should be saved as

```
function val=model_example(x,y,keyf,keyxy)
if nargin<4 || isempty(keyxy)
    switch keyf
    case 'f'; val = [x(1)^2 + x(2)^2*y(2) - 1/3 * y(1)^3];
    case 'F'; val = [x(1) + y(1); ...
                     y(2)];
    case 'G'; val = [x(1)^2 + y(1); ...
                     y(1) * y(2)];
    case 'G_tilde'; val = [y(1) - 10; ...
                           0 - y(1); ...
                           y(2) - 10; ...
                           0 - y(2)];
    end
else
    switch keyf
    case 'f'
        switch keyxy
        case 'x' ; val = [2*x(1), 2*x(2)*y(2)];
        case 'y' ; val = [-y(1)^2, x(2)^2];
        case 'xx'; val = [2, 0; ...
                          0, 2*y(2)];
        case 'xy'; val = [0, 0; ...
                          0, 2*x(2)];
        case 'yy'; val = [-2*y(1), 0; ...
                          0,       0];
        end
    case 'G'
        switch keyxy
        case 'x' ; val = [2*x(1), 0; ...
                          0,      0];
        case 'y' ; val = [1,    0; ...
                          y(2), y(1)];
        case 'xx'; val = [2, 0; ...
                          0, 0; ...
```

```
                                0, 0; ...
                                0, 0];
        case 'xy'; val = zeros(4, 2);
        case 'yy'; val = [0, 0; ...
                                0, 0; ...
                                0, 1; ...
                                1, 0];
        end
    end
end
```

MOBO infers the dimensions m_1, m_2, p and $\tilde{p}$ from the model `.m`-file. If there is a mistake regarding the dimension of a derivative, the program returns an error. Afterwards the user is asked to check (with the help of several warnings) whether the mistake occurs in the definition of the function or in the definition of the derivative.

2.2 Determining the parameters for the algorithm

Some parameters need to be declared by the user in a `.m`-file. Again, it is highly recommended to complete `parameter_dummy.m` with the desired values.

2.2.1 Dimensions

At first, the dimensions of the lower and upper level variables n_1, n_2 should be given.

n_1 is the dimension of the lower level variable x,

n_2 is the dimension of the upper level variable y.

Those two variables are defined in the parameter file and not in the model file so that the user does not need to change the model file of a problem extracted from the bilevel optimization library [4].

2.2.2 Box for $\tilde{G}$

It is also required to declare some parameters for the algorithm itself. As stated in Section 1, the set $\{y \,|\, \tilde{G}(y) \leq 0\}$ does not have to be a box, but at least needs to be bounded. This is due to the fact that the algorithm needs a box containing the set $\{y \,|\, \tilde{G}(y) \leq 0\}$, namely $[c, d] = \{y \,|\, c \leq y \leq d\}$ with $c, d \in \mathbb{R}^{n_2}$ and $\leq$ meant component-wise.
This box will be discretized for the first approximation of the feasible set of (ULP) by stepsize $\beta \in \mathbb{R}^{n_2}$ such that $y^k = c + (k_1\beta_1, \ldots, k_{n_2}\beta_{n_2})^\top$, $k = (k_1, \ldots, k_{n_2}) \in \mathbb{N}_0 \times \cdots \times \mathbb{N}_0$ as long as $y^k \leq d$. Therefore the parameter β should be declared in the parameter file or computed by specifing how many steps should lie between c and d (then $\beta_i = \frac{1}{steps_{(i)}}(d_i - c_i)$, $i \in \{1, \ldots, n_2\}$, where *steps* can be a n_2-dimensional vector or a scalar).
This discretization is also the reason why $n_2 \geq 1$ and $\tilde{p} \geq 1$ are required.
When providing β one should keep in mind that for this first approximation of the feasible set of (ULP), MOBO solves at least $m_1 \cdot \prod_{i=1}^{n_2}(steps_i + 1)$ times a minimization problem.

```
% parameters for the box containing G_tilde (see explanation
% after Algorithm 1):
c = [];                  % n_2 dimensional vector, lower bound
                         % of G_tilde
d = [];                  % n_2 dimensional vector, upper bound
                         % of G_tilde
% discretization parameter for G_tilde:
steps = 10;              % with how many steps G_tilde should
                         % be discretized (can be n_2 dimen-
                         % sional or scalar)
beta = (d-c)./steps;     % n_2 dimensional vector, stepsize in
                         % every direction
```

2.2.3 Parameters for step 2

For calculating a first approximation of the feasible set of (ULP) (step 2 of Algorithm 1 in [1]) one needs to specify some values regarding the update of the parameter $a \in \mathbb{R}^{m_1}$ in (SP$(a, r, \tilde{a})$) with $r = e_{m_1} = (0, \dots, 0, 1)^\top \in \mathbb{R}^{m_1}$ and $\tilde{a} = y^k$.

The parameter $\alpha > 0$ describes how close two consecutive minimal solutions x^i, x^{i+1} of the first approximation of $\Psi(y^k)$ lie (see (5.1) in [1]).
Since in the algorithm, (LLP(y)) is solved by (SP$(a, r, \tilde{a})$), one regulates this by varying $a^{l+1} = a^l + \frac{\alpha}{||(M^{-1}N\tilde{v})_{|x}||} v^a$ with $v^a = e_j \in \mathbb{R}^{m_1}$ (the j-th unit vector) for each $j \in \{1, \dots, m_1 - 1\}$ and $\tilde{v} = ({v^a}^\top, 0_{m_1}^\top)^\top \in \mathbb{R}^{2m_1}$ (for further explanations see (5.2) in [1]).
Thus, if one wants to have a finer first approximation of the feasible set of (LLP(y)), one might reduce α. Or, conversely, if the first approximation (step 2) takes too much time one can increase α . The effects of changing α in contrast to reducing or increasing β differ for each problem.

For numerical reasons, the algorithm sets $a^2 = a^1 + \delta v^a$ with $a^1 = (f_{1:m_1-1}(\bar{x}^j, y^k)^\top, 0)^\top$, $\bar{x}^j \in \operatorname{argmin}\{f_j(x, y^k) \,|\, G(x, y^k) \leq 0_p\}$, $j \in \{1, \dots, m_1 - 1\}$. That is why the user has to declare a small $\delta > 0$, too (for further explanations see after Algorithm 1 in [1]). It is recommended to use $\delta = 0.1$.

```
% parameters for step 2:
% distance for nearly equidistant approximation of the set
% of feasible points of the upper level problem (see (5.1)):
alpha = 0.1;
% small number for slight mutation of the initial a for
% numerical reasons (see step 2a and explanation below
% Algorithm 1):
delta = 0.1;
```

2.2.4 Parameters for step 4

After computing a first approximation of the feasible points of (ULP), namely A^0, MOBO extracts the minimal points $M(A^0)$ (step 3) and refines the approximation around those points. This refinement happens repeatedly in step 4 of Algorithm 2 in [1] producing sets A^i and their minimal points $M(A^i)$ (computed in step 5) for $i \in \{1, \dots,$ `rep_step4`$\}$.

To do so, MOBO expects the user to set the parameter `rep_step4` to the desired number of repetitions of step 4 (and the consecutive step 5).

To refine the approximation set $A^i (i \in \{0, \dots,$ `rep_step4`$-1\})$, the program solves (SP$(a, r, \tilde{a})$) for a variation of $\hat{a} = (a^\top, \tilde{a}^\top)^\top \in \mathbb{R}^{m_1+n_2}$. This variation depends on a parameter γ and a parameter n^D:

$$\hat{a} = \hat{a}^0 + l^1 s^1 v^1 + l^2 s^2 v^2 \text{ for each } l^1, l^2 \in \{-n^D, \dots, n^D\},\ (l^1, l^2) \neq (0, 0),$$

$v^1 = e_j \in \mathbb{R}^{m_1+n_2}$, $j \in \{1, \dots, m_1 - 1\}$, $v^2 = e_{m_1+k} \in \mathbb{R}^{m_1+n_2}$, $k \in \{1, \dots, n_2\}$ and

$$s^i = \frac{\gamma}{||(\hat{M}^{-1}\hat{N}v^i)|_{(x,y)}||},\ i \in \{1, 2\}$$

(for further explanations see before (5.5) in [1]).
For the sake of simplicity n^D is the same for every repetition of step 4, but γ should be provided as a vector of size `rep_step4` in order to change the value of γ in every repetition of step 4.
If step 4 takes too long, consider reducing n^D (therby reducing the number of produced points in A^i) or, conversely, if it is desired to explore more points in the surrounding of $\hat{a}$, one might increase n^D.
It is recommended to decrease γ with each repetition of step 4 because the more the approximation of the feasible set of (ULP) is refined, the closer one wants to search in a surrounding of $\hat{a}$. So γ influences how far the surrounding of a point in A^i is explored.

If it is desired to plot the points resulting from the algorithm `color` should be set as a vector of length `rep_step4`$+1$, otherwise set `color = []`. If `color` is not empty (and if possible), then the f and F values of A^i (the approximation of the feasible set of (ULP)) are plotted (with `'x'`) and the points, where F is minimal on the current A^i are circled (`'o'`) (see examplary figures below for `rep_step4 = 2`, `color = ['b', 'r', 'g']`, $m_1 = 2$, $m_2 = 2$, $n_2 = 1$).

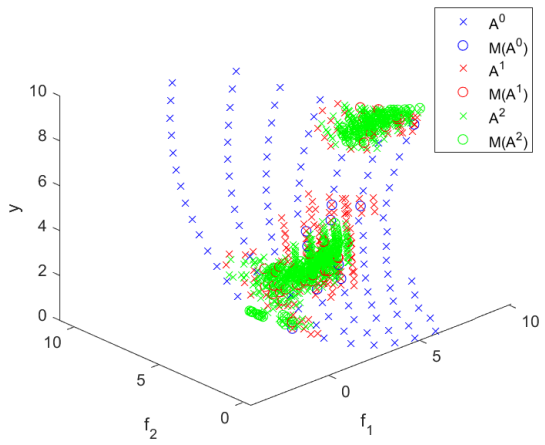

Figure 1: Image space of (LLP(y))

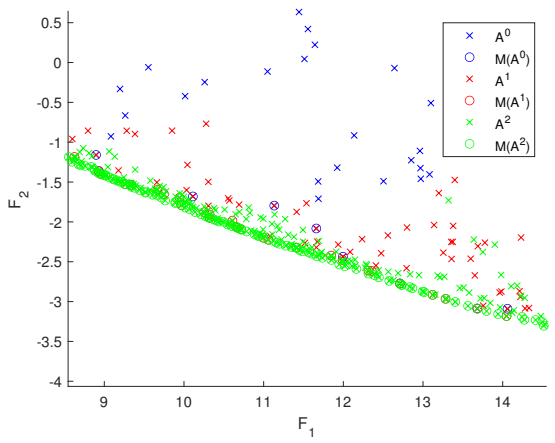

Figure 2: Image space of (ULP) (zoomed in)

```
% parameters for step 4:
rep_step4 = 2;              % how often step 4 should be
                            % repeated (A^i refined)
nD = 2;                     % number of desired new discreti-
                            % zation points (see (5.5))
% vector of length rep_step4 defining how large the variation
% of a_hat in step 4 should be (see before (5.5)):
gamma = [0.3, 0.15];
% vector of length rep_step4 + 1 containing colors for
% plotting A^i after step 4 resp. M(A^i) after step 5,
% if empty no plot will be produced:
color = ['b', 'r', 'g'];
```

2.2.5 Parameters for fmincon

Some parameters of `fmincon` should also be handed over to the parameter `.m`-file. For further clarification see the documentation of `optimoptions`.

```
% parameters for fmincon:
MaxIterations = 1000;
MaxFunctionEvaluations = 3000;
OptimalityTolerance = 1e-06;
```

The parameter `.m`-file should now contain a function with no input parameters and the return values `c`, `d`, `beta`, `alpha`, `delta`, `nD`, `rep_step4`, `gamma`, `color`, `MaxIterations`, `MaxFunctionEvaluations`, `OptimalityTolerance`. The dimensions of the optimization values `n_1`, `n_2` need to be set to `global`.

2.3 Running a model

After creating the model `.m`-file and the parameter `.m`-file the user has to complete `UserFile.m` with the names of those files and to specify whether the model is extracted from the bilevel optimization library [4].

```
%% +++++ Please enter your functions and parameters +++++++++
global Model n_1 n_2 m_1 m_2 p

% name of file holding functions:
Model = 'model_dummy';
% is it a model from BOLIBv2 ?
BOLIB = false;
% name of file holding parameters for the algorithm:
parameter = 'parameter_dummy';
```

After this one last complement one can run MOBO by running `UserFile.m`.

2.4 The Output

While and after running `UserFile.m` MOBO will produce an output into the command window.
On one hand the output issued while running marks when MOBO finishes a step of the Algorithm, so that one can estimate how long each step takes and probably adjust the parameters.

```
step 1 finished
step 2 finished
step 3 finished
step 4 finished for i=0
step 5 finished for i=0
step 4 finished for i=1
step 5 finished for i=1
step 4 finished for i=2
step 5 finished for i=2
program finished
```

On the other hand the output produced at the end of the program presents the F-, x- and y-values of $M(A^{\texttt{rep\_step4}})$. It also indicates with which exitflag of `fmincon` the point (x, y) was generated. Be suspicious if it is 0, because that means that the point was produced in step 2 whilst computing the Lagrange multipliers of $(\mathrm{SP}(a, r, \tilde{a}))$ with $a = (f_{1:m_1-1}(\bar{x}^j, y^k)^\top, 0)^\top$, $\bar{x}^j \in \operatorname{argmin}\{f_j(x, y^k) \,|\, G(x, y^k) \leq 0_p\}$ for one $j \in \{1, \ldots, m_1 - 1\}$ and $\tilde{a} = y^k$, but x may not be a minimal point of $(\mathrm{LLP}(y))$. Normally, this won't happen.

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
   F values    |    x values    |    y values    | exitflag
0.0000 0.0000 |0.0000 1.0000 |0.0000 0.0000 |1
-0.0000 1.0000 |-0.0000 -0.0000 |0.0000 1.0000 |1
-0.0000 6.0000 |-0.0000 -0.0000 |0.0000 6.0000 |1
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

This is the output created for the model discussed in the example of Section 2.1. One can easily verify, that the real solution is $(x_1, x_2, y_1, y_2) = (0, \xi, 0, 0)$ with $\xi \in \mathbb{R}$ and minimal value $F(x, y) = (0, 0)^\top$. In this output of MOBO one can see, that there are numerical difficulties such that the other two points seem to be minimal as well.

References

[1] G. Eichfelder: *Multiobjective bilevel optimization.* Mathematical Programming, 123(2):419–449, 2010.

[2] C. Günther and N. Popovici: *New algorithms for discrete vector optimization based on the Graef-Younes method and cone-monotone sorting functions.* Optimization, 67(7):975–1003, 2018.

[3] A. Pascoletti and P. Serafini: *Scalarizing vector optimization problems.* Journal of Optimization Theory and Applications, 42(4):499–524, 1984.

[4] S. Zhou, A.B. Zemkoho, and A. Tin: *Bolib: Bilevel optimization library of test problems.* In *Bilevel Optimization*, pp. 563–580. Springer, 2020.