MOPS: Documentation

Leo Warnow

March 6, 2019

1 Introduction

This MATLAB algorithm deals with **multiobjective optimization problems**. Therefore, we consider

$$\min_{x \in S} \begin{pmatrix} f_1(x) \\ \vdots \\ f_m(x) \end{pmatrix} \tag{MOP}$$

with $S \subseteq \mathbb{R}^m$ being a non-empty set and $f_i : \mathbb{R}^n \longrightarrow \mathbb{R}$ continously differentiable functions for all $i \in \{1, \dots, m\}$ $(n, m \in \mathbb{N}, m \geq 2)$. Our aim is to find so called **efficient points**, that is $\bar{x} \in S$ so that there is no other $x \in S$ with

$$f_i(x) \leq f_i(\bar{x}) \text{ for all } i \in \{1, \dots, m\} \text{ and}$$
$$f_j(x) < f_j(\bar{x}) \text{ for at least one } j \in \{1, \dots, m\}.$$

In general there are infinitely many of those points. The goal of this algorithm is to find a finite selection which somehow well represents this (infinite) set. Sometimes you want to be a little bit less restrictive and search for **weakly efficient points**, that is $\bar{x} \in S$ so that there is no $x \in S$ with

$$f_i(x) < f_i(\bar{x}) \text{ for } i \in \{1, \dots, m\}.$$

As every efficient point is also a weakly efficient point, this gives a superset of the optimal solution set of the original problem. The main idea is to use a scalarization approach presented in [4] by Pascoletti and Serafini. As this scalarization only leads to one weakly efficient point, the algorithm varies the scalarization's parameters to generate a (finite) set of them. So it is our goal to obtain a representation of the set $\{f(x) \in \mathbb{R}^m \mid x \in S \text{ efficient}\}$. For $m = 2$ it is possible to generate a nearly equidistant representation by using the adaptive mode (see section 2.3.2). These ideas were introduced by Eichfelder in [1] and [2].
The algorithm is entirely written in MATLAB and was tested in MATLAB R2015b and R2018a. As it uses `fmincon`, it requires the Optimization Toolbox for MATLAB. If you use this algorithm in publications, please provide reference to [2].

2 Working with the algorithm

In the following sections you will be shown how to use this algorithm to work with your problems and how to edit certain properties. There will also be some advice on how to deal with certain problems that may occur.

2.1 Getting started

This section will guide you through the very first steps including unpacking the solver archive. There will also be a walkthrough concerning (basic) usage of the algorithm.

2.1.1 Preparation

After you have downloaded the zip-archive containing the solver, please unpack it. You can do this at whatever place on your computer you want, but make sure that you have both read and write access to this specific directory.
If you now open the folder which contains all the files you will see three folders (problems, parameters and solver) as well as two MATLAB files named **`callSolver.m`** and **`UserFile.m`**. Now simply open **`UserFile.m`** in MATLAB and you are ready to go.

2.1.2 Solving your first problem

Once you have extracted all files from the zip-archive and opened **`UserFile.m`** you can solve a first problem to get familiar with the solver. As there are some prepacked examples, you can just run this file and get your first results. If you want to solve your own problems, here are three options to be set:

```
25 %% Please enter your parameters below
26 % Your Problem
27 problem = 'problem_KimDeWeck';
28
29 % Choose a parameter file
30 parameter = 'ps_param_auto';
31
32 % Should the result be plotted (m = 2 and m = 3 only) [1 == yes, 0 == no]
33 plot_result = 1;
```

The first parameter to set is **`problem`** which contains the filename of the problem you want to solve. Please make sure that there is a file with this name in the problems subfolder. There are also some prepacked examples you can use. As the algorithm needs some parameters you will also have to provide the name of a parameter file in the variable **`parameter`**. This file has to be in the parameters subfolder. For image dimension $m = 2$ and $m = 3$ the algorithm can also plot its result. This is exactly what the parameter **`plot_result`** is used for. This variable will be automatically ignored if $m > 3$, so you don't have to set it to 0 in this case.

2.2 Providing a problem

To make this algorithm work with a certain problem you will at least have to create a MATLAB function file with all problem relevant data to be used by the solver.
First of all make sure that you are in the **`problems`** subfolder. Now you can either create a new MATLAB function file or simply duplicate one of the existing problems and then modify the copy you created. In the following we will always take **`problem_ball.m`** as an example problem. This is

$$\min_{x \in S} x$$

with $n = m = 3$ and $S = \{x \in \mathbb{R}^3 \mid \|x\| = \sqrt{x_1^2 + x_2^2 + x_3^2} \leq 1,\ -2 \leq x_i \leq 2\ \forall i \in \{1, 2, 3\}\}$.

2.2.1 Problem header

As every other function a problem has to start with a header. Let's take a look at our ball problem's header:

```
1 function [n,m,fun,x0,Aineq,bineq,Aeq,beq,lb,ub,nonlcon,options] =
      problem_ball()
2 %PROBLEM_BALL A test instance with m=3 for our solver
3 %   Try to compute the pareto front of an euclidean ball with its center at
4 %   [0;0;0] and radius 1.
```

As you can see there are no input arguments. This is because the algorithm only needs this function to get all problem relevant information such as the objective function or constraints, so this is kind of a read only file. Each problem file needs to have exactly the 12 specified output arguments you can see in this example and they have to be in the exact same order as well. While the algorithm doesn't care what names you choose for those 12 variables, it is strongly recommended to stay with the names as you can see them here. Details concerning all those parameters will appear in the following sections.
As you can see, there is also a short description of our problem in lines two to four. You don't have to include a description for your problem, but it may help you and others later. Finally keep in mind that the name of your function matches the file name (especially when creating a new problem by duplicating another).

2.2.2 Dimensions and objective function

The first three output parameters are the dimensions of domain and image space as well as a function handler for the objective function. Let's take a look at the ball problem again:

```
6  %Dimension of domain and image space
7  n = 3;
8  m = 3;
9
10 % Objective function
11 fun = @(x) [x(1);x(2);x(3)];
```

For this specific problem we have $n = m$ but of course these two dimensions can be different. Remember that n is the dimension of the domain, m the dimension of the image space and $f\colon \mathbb{R}^n \longrightarrow \mathbb{R}^m$ is the objective function. Please ensure that the objective function returns a column vector (not a row vector) of size m.

2.2.3 Starting point

As all scalar optimization problems are performed by **fmincon** you have to provide a starting point. For the ball problem this is $x_0 = (0, 0, 0)^\top$.

```
13 % Starting point for optimizer
14 x0 = zeros(n,1);
```

Please make sure that the starting point you provide is a column vector of size n. It is recommended to choose a point $x_0 \in S$. However in most cases some point outside of S will work as well.

2.2.4 Linear constraints and bounds

As already mentioned there is some set of feasible points $S \subseteq \mathbb{R}^m$. This set may be (partwise) given by some linear equations or inequalities as well as some bounds for $x \in \mathbb{R}^n$. Let's take a look at the ball problem again:

```
16 % Linear constraints (Aineq*x <= bineq, Aeq*x = beq)
17 Aineq = [];
18 bineq = [];
19 Aeq = [];
20 beq = [];
21
22 % Lower and upper bounds (lb <= x <= ub)
23 lb = ones(n,1).*(-2);
24 ub = ones(n,1).*2;
```

As you see there are no linear constraints for this problem. However, and this is really important, you have to provide all four variables **Aineq**, **bineq**, **Aeq**, **beq** because as mentioned earlier you need to provide all 12 output parameters.

The same holds for the lower and upper bounds: If your problem doesn't have any bounds you still have to provide those variables. In this case you can simply set `lb = []` and `ub = []`.

2.2.5 Nonlinear constraints

Of course your set S can also be given by some nonlinear constraints. Those are provided by another function handler. Let's take a look at the ball example:

```
26 % Non-linear constraints (c(x) <= 0, ceq(x) = 0)
27 function [c,ceq] = nonlcon_fun(x)
28     c = (x(1)^2+x(2)^2+x(3)^2)-1;
29     ceq = [];
30 end
31 nonlcon = @nonlcon_fun;
```

Now there are several things to consider. First of all you always have to provide a function handler for nonlinear constraints. The function itself has to have exactly one input argument which is $x \in S$. There also have to be exactly two output vectors: one for inequalities and one for equalities. These can really be vectors (of different sizes) as you have multiple constraints there. Please keep in mind that (as all vectors) these have to be column vectors.

2.2.6 Solver options

Finally you can provide options for **`fmincon`** to use, e.g. the algorithm to use, maximum number of iterations, optimality tolerance and so on. However, the ball example doesn't provide any options there.

```
33 % Options for solver
34 options = [];
```

If you don't provide any options, the algorithm will load some default ones. These consist of two parameters: The **`Display`** option is set to **`none`** and the default **`Algorithm`** is set to **`interior-point`**. The last one is mainly to surpress compatibility warnings.

2.3 Setting the parameters

First of all it is not necessary to provide your own parameter file to make the algorithm work with your own problems. You can just use one of the default files. Nevertheless, the following sections will demonstrate how a parameter file works. Therefore, please make sure you are in the **`parameters`** subfolder. As well as problem files, all parameter files are MATLAB functions. To create a new parameter file you can either create a copy of an existing one and modify it or build it from scratch. As an example we will take a look at **`ps_param_01.m`**.

2.3.1 Parameter header

As the parameter file is a function file it starts with a function header. It will look like this:

```
1 function [adapt,alpha,beta,epsilon,b,r] = ps_param_01(~)
2 %PS_PARAM_01 Some default parameters for ps_standard to load
3 %   A demonstration of how a parameter file could look like. You can use
4 %   this together with problem_01 to see how the non-adaptive algorithm
5 %   works for m = 2.
```

A parameter file needs to provide exactly 6 output arguments. These have to be in the exact same order as shown above. As you can see the parameter file has one input argument which is ignored in this specific example. This input argument has to be a function handler for a problem file. It is (only) needed, if you want your parameters to be computed based on the problem. One reason to this could be that you want your parameter file to work for any dimension $m \in \mathbb{N}$. In this case you will have to extract the dimension parameter of the problem file.

If you want to see an example of how to do this, take a look at **ps_param_auto.m**. If you don't need any data from the problem file, replace this input argument by ~ as shown in the example.

2.3.2 Adaptive mode

After the problem header you will find the parameter to choose adaptive or non-adaptive mode:

```
% Should Pascoletti Serafini use adaptive mode? (1 == yes, 0 == no)
adapt = 0;
```

This parameter is only important for problems with $m = 2$. In any other case it will automatically be ignored (but you have to provide it). If the parameter is set to 1 then the algorithm will call solver **ps_adapt.m** which uses an adaptive step size. Otherwise **ps_standard.m** will be used (with fixed step size). If $m > 2$ then the solver **ps_general.m** is called no matter what this parameter looks like.

2.3.3 Pascoletti Serafini parameters

Finally there are the different parameters for the Pascoletti Serafini scalarization and two more parameters to control the step size:

```
% Parameter for Pascoletti Serafini
alpha=0.10; % This will be ignored if adapt==0
beta=1;
epsilon=0.05;
b=[1;1];
r=[1;1];
```

The number of iterations for the solvers is controlled by parameter **epsilon** (ϵ). For $m = 2$ there are two cases: If the non-adaptive method is used, the number of iterations is $\lfloor 1/\epsilon \rfloor$. If the adaptive method is used then an additional parameter **alpha** modifies the step size based on the Lagrange multiplier of the scalarization problem. If you want to know how exactly this works, please take a look at the solver files, [1] or [2]. For $m > 2$ the solver creates an $(m-1)$-dimensional grid with $1/\epsilon$ points per dimension. In this case you will have $(1/\epsilon)^{m-1}$ iterations.

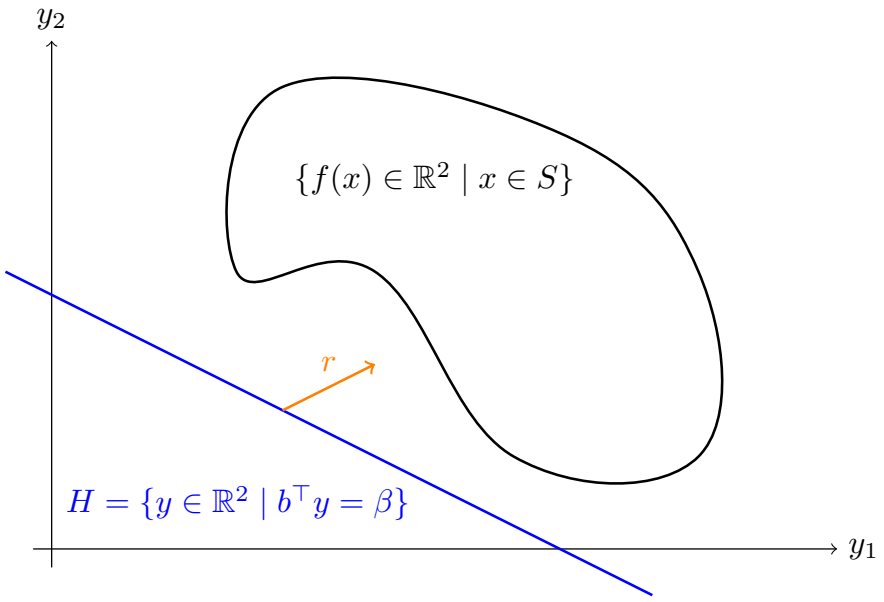


Figure 1: Image of S under the objective function, hyperplane H and direction r

The other parameters are used to define a hyperplane $H = \{y \in \mathbb{R}^m \mid b^\top y = \beta\}$ and a direction $r \in \mathbb{R}^m$. You can find a schematic illustration in Figure 1. Please make sure that **b** and **r** match the dimension m of your problem and $b^\top r \neq 0$. If you want to know how those parameters are used in the algorithm, take a look at [1] or [2].

2.4 Implement a problem on your own

Now you know how to provide problems and parameters for the algorithm by using MATLAB function files. You are also familiar with using the UserFile to choose a problem and a parameter file to work with. So it is time to start working with all those files.

2.4.1 Problem description

For this whole section we will take a closer look at a problem which was also used in [1]. This is

$$\min_{x \in \mathbb{R}^2} \begin{pmatrix} \sqrt{1 + x_1^2} \\ x_1^2 - 4x_1 + x_2 + 5 \end{pmatrix} \text{ s.t. } x_1^2 - 4x_1 + x_2 + 5 \leq 3.5, \\ x_1 \geq 0, x_2 \geq 0.$$

This problem has three inequality constraints. How would you provide these in a problem file? Which of them are linear / non-linear constraints and which are lower and upper bounds?
If you have found an answer to all those questions you can find this problem in the file **problem_02.m** in the **problems** folder and check how we provided the constraints there. But you could also try to implement the whole problem on your own. This would be a good exercise.

2.4.2 Choosing a set of parameters

Now that you have provided the problem in a problem file, you will need some parameters to start with. The easiest way would be to use **ps_param_auto** to get a first idea of what the set $\{f(x) \in \mathbb{R}^m \mid x \in S \text{ efficient}\}$ looks like. In [1] the following parameters are used:

$$H = \{y \in \mathbb{R}^2 \mid (1,1)y = 2.5\}, r = (1,0)^\top \text{ and } \alpha = 0.2.$$

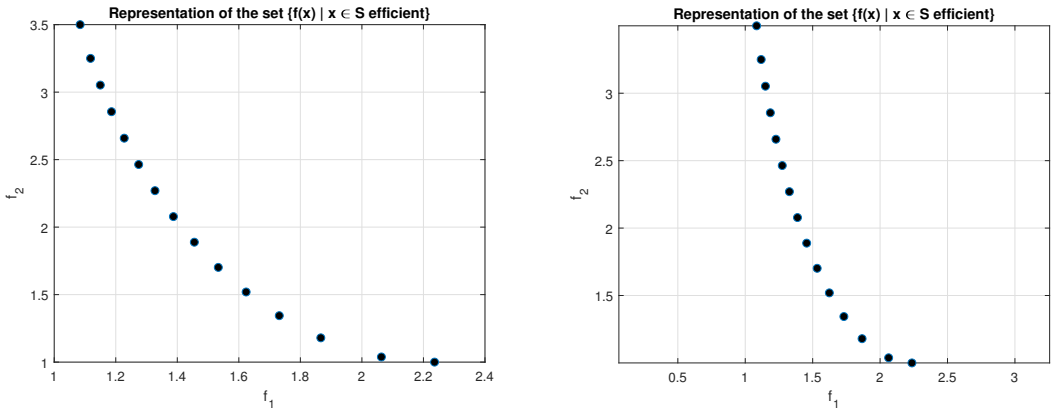
The values for r and α are obvious, but what about b and β? Maybe you can try to provide a parameter file that realizes all the things above. If you want to check your results, take a look at the prepacked parameter file **ps_param_02.m**. In this file you will see that we decided to use the adaptive mode and chose $\epsilon = 0.1$. Of course this is not specified above, so your parameters can be totally different here.

2.4.3 A first run

As you have a problem and a parameter file, you are ready to get the first results. Everything you have to do is to provide them in the **UserFile.m**. Maybe you want to go back to section 2.1.2 to lookup how to do this. If you want to use the prepacked files, it would look like this:

```
25 %% Please enter your parameters below
26 % Your Problem
27 problem = 'problem_02';
28
29 % Choose a parameter file
30 parameter = 'ps_param_02';
31
32 % Should the result be plotted (m = 2 and m = 3 only) [1 == yes, 0 == no]
33 plot_result = 1;
```

Now just press Run in MATLAB (or F5 on your keyboard) to get a first result.



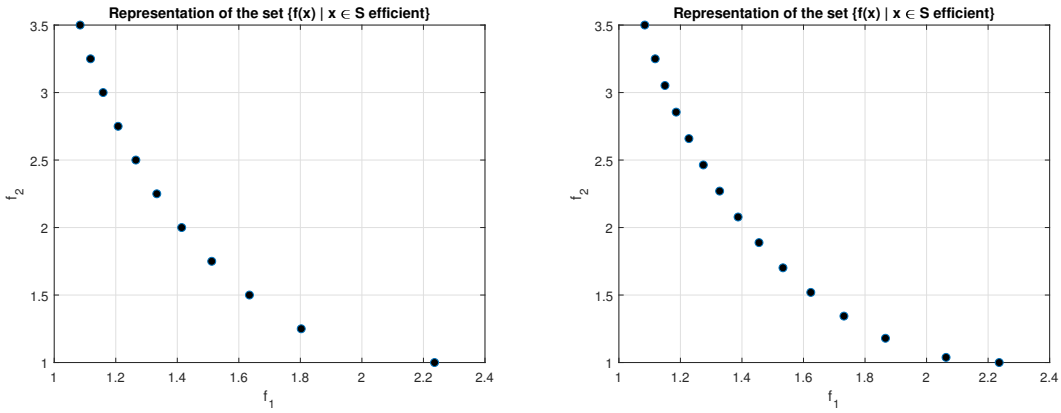
(a) Default result (b) Result for equal axes

Figure 2: Results for a first run with the prepacked parameters

In Figure 2 you can see what your results should look like. Maybe you expected a more "quadratic" look of the representation. Therefore, you will have to make sure that the axes are scaled equally by using the command **`axis equal`** from the MATLAB command line. The corresponding result is also shown in Figure 2.

2.4.4 Changing the parameters

For a better understanding of the algorithm, we recommend to vary the parameters and take a closer look at the results. For example set $b = r = (1, 0)^\top$ and choose non-adaptive mode by setting **`adapt = 0`**. What do you expect to see?
Now run the UserFile again to check the results. Is everything as expected? If you take a look at the generated image, you may notice that the points are not equally distributed along the quadratic curve as before, but they are equally distributed concerning f_2. To verify this, just take a look at the entries in **`fres`**. They start at 1.00 and go up to 3.50 by equal steps of 0.25. The image can also be seen in Figure 3.

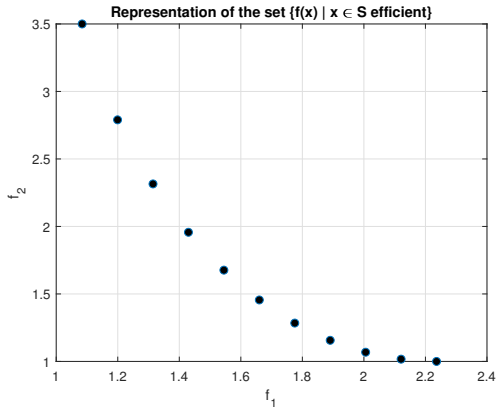


(a) Result for non-adaptive mode (b) Result for adaptive mode

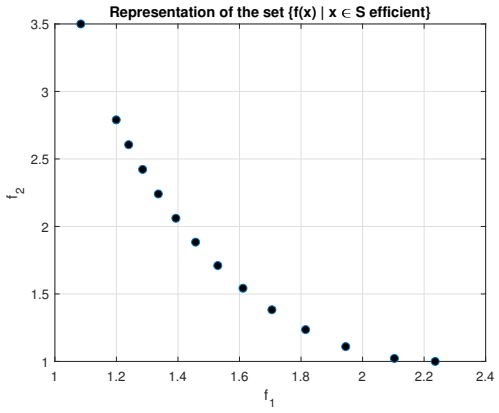
Figure 3: Results for $b = r = (1, 0)^\top$

As already mentioned in section 2.3.2 we expect a more equally distributed result along the quadratic curve when using adaptive mode. So what happens if you go to the parameter file and set **`adapt = 1`**? Just take a look at Figure 3. You can see how the gap in the lower right section of the image is no longer present when using adaptive mode.

However, adaptive mode can't guarantee better results for every set of parameters. For example set $b = r = (0,1)^\top$. This will probably return images as shown in Figure 4. As you can see the gap (which is now in the upper left corner) did not disappear. But nevertheless the distribution between all the other points seems now to be a little bit more equal.



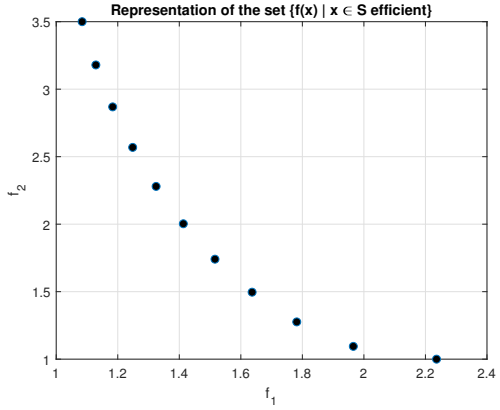
(a) Result for non-adaptive mode



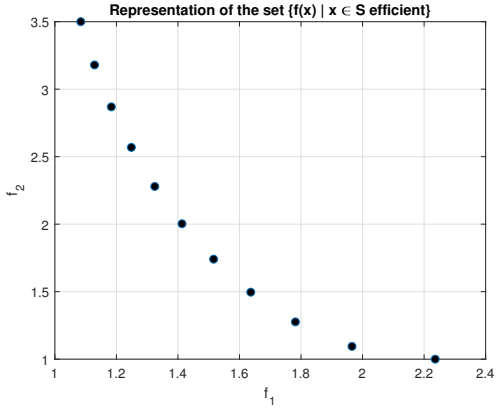
(b) Result for adaptive mode

Figure 4: Results for $b = r = (0,1)^\top$

Finally we would like to point out that the impact of the hyperplane is somehow limited by the direction r. Therefore, please adjust the parameters again and set $r = (1,1)^\top$ and make sure that you have still $b = (0,1)^\top$ and adaptive mode is turned off.



(a) Result for $b = (0,1)^\top$



(b) Result for $b = (1,0)^\top$

Figure 5: Results for $r = (1,1)^\top$ and different vectors $b \in \mathbb{R}^2$

The results you will get now are already distributed quite equally along the quadratic curve. You can see the corresponding image in Figure 5. There you can also see the results for the same direction $r = (1,1)^\top$ but another hyperplane given by $b = (1,0)^\top$. As you can see there is no noticeable difference between those two results. One may now easily underrate the impact of the hyperplane on the result and only care about the direction r. But those two (hyperplane and direction) always belong together. Just think of the condition $b^\top r \neq 0$. What happens if you choose some b and r with $b^\top r$ beeing very close to 0?

2.4.5 Further ideas

Now you have seen some ways to change the different parameters and what you can expect when doing so. But of course there are many more combinations one could think of and try out. For example you could adjust β and ϵ and see what will happen. We would also recommend to take a look at another problem like `problem_01.m`. How do all the different choices of parameters from above affect the results for this example? There are endless possibilities and things to try out.

3 Troubleshooting and advanced adjustments

Finally we will discuss some possible issues that can occur when using this algorithm. There will also be some advice on how to further improve the algorithm for your needs.

3.1 Finding global solutions

For solving any kind of (scalar) optimization problems, this algorithm uses the MATLAB function `fmincon`. As this is a local solver it will only lead to locally optimal solutions. It is also important to know that `fmincon` is a gradient-based method. Therefore, not only the objective functions but also the constraint functions (given by either equalities or inequalities) should be continously differentiable. For convex problems and the prepacked test instances this is not a problem. However, you may experience some issues with that for more complex problems.

3.1.1 An easy multistart approach

One approach to increase the chance of finding a global minimum could be to use multistarts. Therefore, you can easily embed the `fmincon` operations in a for loop and change the starting point within every iteration. Than you just select the minimum of all those solutions. This will not necessarily lead to a global optimal solution but it is easy to implement and definitely worth a try. Please keep in mind that this approach will (depending on the number of multistarts) increase computation time.

3.1.2 Using a global solver

Another idea is to replace `fmincon` by a global solver. In case of more complex non-convex objective functions and a set S given by box and convex constraints but without equality constraints, we recommend to use the algorithm proposed by Niebling and Eichfelder in [3]. Depending on what kind of solver you choose, you will maybe have to adjust the problem parameters to the solver's needs. In other words, you may have to do some more advanced modifications not only in the problem files but also in the solver files. Therefore, we recommend to create a new copy of your problems, so that you can easily switch back to the non-modified versions in case things don't work out as well as expected.

3.2 Increasing speed

For some problems you may experience a longer computation time. This section will present some ideas how to speed up the algorithm.

3.2.1 Decreasing the number of iterations

The easiest way to reduce computation time is to decrease the number of iterations by increasing the parameter ϵ. Especially for problems with a large dimension m this can have a huge impact. Just remember that the number of iterations in this case is $(1/\epsilon)^{m-1}$.

3.2.2 Parallelization

For this section we assume that you use a non-adaptive algorithm. An easy approach to speedup the solver is to replace the main for loop by a parfor loop. You might ask why we haven't set this to be the default option. The answer is that for the prepacked examples starting the parallel pool takes more time than actually solving the problem. Please also keep in mind that you need the Parallel Computing Toolbox to use parfor (if the Toolbox is not activated MATLAB will do a usual for loop instead). Of course you can use all other tools of this toolbox as well (e.g. using your GPU instead of or in addition to the CPU).

3.3 Providing suitable problems and parameters

Finally we want to give you some advice on how to provide your problem and choosing a suitable hyperplane (by adjusting its parameters in a parameter file).

3.3.1 Scaling the objective functions

Sometimes one of your objective functions has a very small or huge range (in image space) compared to another. This can lead to various problems including numerical issues and an unequally distributed result. Therefore, we recommend to scale your objective functions before providing them to the algorithm. Of course you have to be aware that the backscaling can lead to some numerical errors in the image space as well. However, your optimal solutions (in the domain space) are not affected.

3.3.2 Choosing suitable parameters

If you use the non-adaptive method your choice of the parameters b, r and β will have quite a large impact on your result, especially concerning the distribution of the result in the image space. Now take a look at Figure 1 again. It seems to be a good approach to choose a hyperplane H that (in some way) represents the curvature of the lower left boundary of the set $\{f(x) \in \mathbb{R}^m \mid x \in S \text{ efficient}\}$. Of course, in general this curvature is not known. However, it is important to keep in mind that the steepness of H may have a huge impact on the results. For better understanding you may want to take a look at the results for `problem_02.m` when using $b = r = (0, 1)^\top$ as well as using $b = r = (1, 0)^\top$.

References

[1] G. Eichfelder, Adaptive scalarization methods in multiobjective optimization, Springer, 2008.

[2] G. Eichfelder, An adaptive scalarization method in multiobjective optimization, SIAM Journal on Optimization, 19 (2009), pp. 1694–1718.

[3] J. Niebling and G. Eichfelder, A branch-and-bound based algorithm for nonconvex multiobjective problems. accepted for publication in SIAM Journal on Optimization, 2019.

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