SOCEMO: Surrogate Optimization of Computationally Expensive Multi-Objective Problems Code Manual

Juliane Müller
juliane.mueller2901@gmail.com

Center for Computational Science and Engineering
Lawrence Berkeley National Laboratory, Berkeley, CA, 94720, USA

Abstract

SOCEMO is an optimization algorithm for solving *computationally expensive, black-box, multi-objective optimization problems*. SOCEMO uses various surrogate models to approximate the computationally expensive objective functions. Hence, derivative information, which is generally unavailable for black-box simulation objective functions, is not needed. SOCEMO aims at solving problems that have continuous variables whose upper and lower bounds are known. Other constraints, such as computationally expensive black-box constraints or computationally cheap equality or inequality constraints cannot be handled by this version, but extensions of the algorithm to these problem classes are underway. This code manual describes the MATLAB code and how you can use SOCEMO to solve your multi-objective optimization problems.

1 Introduction

This documentation accompanies the MATLAB implementation of the algorithm SOCEMO (Surrogate Optimization of Computationally Expensive Multi-Objective problems). We implemented and tested SOCEMO in MATLAB 2012a [1] (for toolbox requirements, see below). SOCEMO is a derivative-free surrogate model algorithm that aims at solving multi-objective optimization problems:

$$\min_{\mathbf{x}\in\mathbb{R}^d} [f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_k(\mathbf{x})]^T \tag{1a}$$

$$-\infty < x_j^l \leq x_j \leq x_j^u < \infty, j = 1, \ldots, d \tag{1b}$$

where $\mathbf{f}(\mathbf{x}) = [f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_k(\mathbf{x})]^T : \mathbb{R}^d \mapsto \mathbb{R}^k, k \geq 2$, is the vector of objective functions (*objective vector*) that must be minimized simultaneously. We assume that the objective functions are deterministic, i.e., the function values are identical for identical input variable vectors $\mathbf{x}$. The variables $x_j \in \mathbb{R}$, $j = 1, \ldots, d$, are bounded from below and above by x_j^l and x_j^u, respectively. d is the problem dimension, and k is the number of objectives with $k \geq 2$. The optimization problem has the following characteristics:

- The objective functions are computationally expensive to compute. Evaluating all k objective functions is extremely time consuming and takes several minutes to hours. In SOCEMO, we assume that for every evaluation point $\mathbf{x}$, we compute *all* k objective function values.
- The objective functions are black boxes. Analytical descriptions of the functions are not available as it is the case, for example, when computer simulations are used when computing the objective function values. All k objective function values may be computed by the same black box simulation, or they may be the output of several simulations.
- Derivatives of $f_i(\mathbf{x}), i = 1, \ldots, k$, are not available.
- The objective functions are deterministic. The value of $f_i(\mathbf{x})$ is identical for identical inputs.
- The objective function is (most likely) multimodal. The domain scientists may have some idea of whether or not there are several basins of attraction. For black-box problems, it is not possible to tell a priori what the shape of the objective function landscape is, and in order to avoid becoming trapped in a local minimum, we assume that all $f_i(\mathbf{x})$ are multimodal, and thus we use a global search strategy in the optimization.
- The objective functions are conflicting, i.e., improving one objective will worsen at least one other.

In multi-objective optimization, there is in general no single solution that optimizes all objectives. Rather, the goal is to find trade-off (Pareto-optimal, non-dominated) solutions. Since a single function evaluation is computationally extremely expensive, we want to do only very few evaluations of $\mathbf{f}(\mathbf{x})$ in order to keep the optimization time acceptable. If all objective functions are computationally inexpensive (fractions of a second), SOCEMO will not be an efficient solver. We developed SOCEMO for problems whose function evaluation time does not allow us to do tens of thousands of evaluations.

In this code companion, we focus mostly on explaining the functionalities of the individual m-function and how to use SOCEMO for solving your multi-objective optimization problems. We recommend reading the paper "SOCEMO: Surrogate Optimization of Computationally Expensive Multi-Objective Problems" by J. Müller (2017, to appear in *INFORMS Journal on Computing*), for further explanations and references. Please cite this paper if you use SOCEMO in your work.

Required MATLAB version and toolboxes: MATLAB 2012(a) and newer (implemented and tested in 2012(a)); Optimization toolbox; Global optimization toolbox; Statistics toolbox.

Make sure that the SOCEMO code directory is known to the MATLAB search path. To test the algorithm, type in the MATLAB command window

```
testdriver
```

This runs a computationally cheap test problem and should finish successfully. Please note that SOCEMO solves several optimization subproblems when making iterative sampling decisions and thus this test run may take some 45-60 seconds to complete. However, compared to the computational cost of doing an actual simulation evaluation, we deem this overhead acceptable.

We organized this code manual as follows. Section 2 is a brief overview of how surrogate model algorithms work in general and radial basis functions (RBFs). The individual m-functions of the algorithm are described in Section 3. The output of the algorithm is described in Section 4. An example of how to define your own optimization problem and how to call the algorithm is given in Section 5. Section 6 shows the code structure for reference in case you are looking for specific function dependencies.

Should you encounter difficulties or bugs, please feel free to contact me at

juliane.mueller2901@gmail.com

Lastly, if you are interested in single-objective mixed-integer or continuous optimization of computationally expensive black-box problems, please feel free to browse through my website `https://ccse.lbl.gov/people/julianem/index.html`.

2 Surrogate Model Algorithm and Radial Basis Functions

Algorithm 1 gives a high-level overview of the steps in SOCEMO and Figure 1 illustrates the steps on a single objective function with one variable.

Algorithm 1 Overview of the Algorithm SOCEMO

1: Create an initial experimental design and do the k expensive objective function evaluations at each point in the initial design. Fit k surrogate models, one for each objective.
2: Use the information from the surrogate models to select the point $\mathbf{x}_{\text{new}}$ at which we do the next expensive function evaluations. Do the expensive evaluations at $\mathbf{x}_{\text{new}}$: $f_{i,\text{new}} = f_i(\mathbf{x}_{\text{new}}), i = 1, \ldots, k$.
3: Update the k surrogate models and go to Step 2.
4: Stop when the stopping criterion is satisfied and return the non-dominated points found.

First, we create an initial experimental design and evaluate the computationally expensive objective functions at the selected points. Note that we need all k function values for every evaluation point. In general, any initial design strategy may be used, but it has to be ensured that there are sufficiently many points to compute the parameters of the surrogate models. The objective function value predictions of the surrogate models at unsampled points are used when selecting the next evaluation points. After the expensive function values have been computed at the newly selected points, the surrogate models are updated if the stopping criterion has not been satisfied (for example, the budget of

function evaluations has not been exhausted) and a new point is selected for evaluation. Otherwise, the algorithm stops and returns the best non-dominated (Pareto-optimal) solutions found so far. As in single-objective black-box global optimization, we cannot guarantee that the returned solutions actually are the true Pareto-optimal solutions (due to the objective functions being black-box and the number of allowed function evaluations being generally too low to guarantee global convergence). However, for practical applications where the evaluation time of the objective functions restricts the number of evaluations that can be done, our approach of using derivative-free efficient surrogate-based sampling has been shown superior over other derivative-free methods such as genetic algorithms which generally need thousands of function evaluations to reach solutions that are close to those found by our surrogate model strategy.

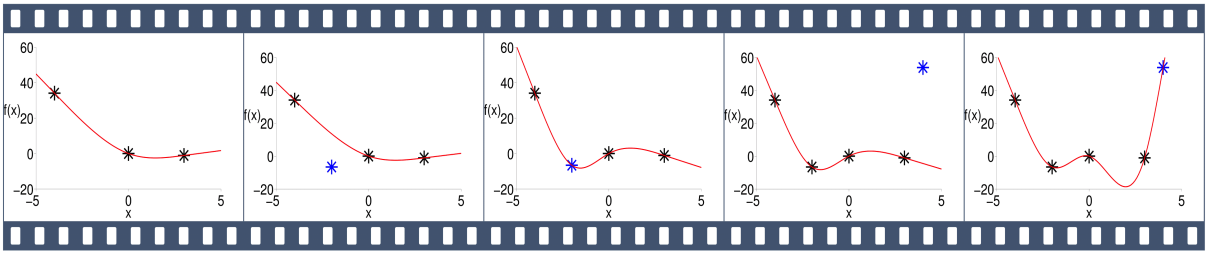


Figure 1: Illustration of the surrogate model algorithm steps described in Algorithm 1 for a one-dimensional problem and for a single objective.

SOCEMO uses radial basis function (RBF) surrogate models, in particular cubic RBF's (although the user can optionally use a linear or thin-plate spline RBF as well). The RBF interpolant is defined as

$$s(\mathbf{x}) = \sum_{\iota=1}^{n} \lambda_\iota \phi(\|\mathbf{x} - \mathbf{x}_\iota\|) + p(\mathbf{x}), \tag{2}$$

where $\phi(\cdot)$ is the radial basis function (types defined in Table 1), $\mathbf{x}_\iota, \iota = 1, \ldots, n$, denotes the points at which the objective function value is known (already evaluated points), and $p(\cdot)$ denotes the polynomial tail whose minimal order (Table 1) depends on the chosen RBF type (we use linear polynomial tails for all RBF types in our implementation). The parameters $\lambda_\iota \in \mathbb{R}, \iota = 1, \ldots, n$, and the parameters of the polynomial tail $\beta_0, \beta_1, \ldots \beta_d \in \mathbb{R}$ are determined by solving the following linear system of equations

$$\begin{bmatrix} \mathbf{\Phi} & \mathbf{P} \\ \mathbf{P}^T & \mathbf{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{\lambda} \\ \boldsymbol{\beta} \end{bmatrix} = \begin{bmatrix} \mathbf{F} \\ \mathbf{0} \end{bmatrix}, \tag{3}$$

where $\Phi_{\iota\nu} = \phi(\|\mathbf{x}_\iota - \mathbf{x}_\nu\|)$, $\iota, \nu = 1, \ldots, n$, $\mathbf{0}$ is a matrix with all entries 0 of appropriate dimension, and

$$\mathbf{P} = \begin{bmatrix} \mathbf{x}_1^T & 1 \\ \mathbf{x}_2^T & 1 \\ \vdots & \vdots \\ \mathbf{x}_n^T & 1 \end{bmatrix}, \quad \boldsymbol{\lambda} = \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \vdots \\ \lambda_n \end{bmatrix} \quad \boldsymbol{\beta} = \begin{bmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_d \\ \beta_0 \end{bmatrix}, \quad \mathbf{F} = \begin{bmatrix} f_i(\mathbf{x}_1) \\ f_i(\mathbf{x}_2) \\ \vdots \\ f_i(\mathbf{x}_n) \end{bmatrix}. \tag{4}$$

The entries of $\mathbf{F}$ in the right hand side of the equation are the function values of the ith objective function at the sampled points. Thus, we compute a separate set of parameters $\boldsymbol{\lambda}$ and $\boldsymbol{\beta}$ for each objective by using the function values from each objective as $\mathbf{F}$. The matrix in (3) is invertible if and only if $\text{rank}(\mathbf{P}) = d + 1$ [3].

Table 1: Radial basis function types and their corresponding minimal degree μ_p of $p(\mathbf{x})$. In the SOCEMO implementation, we use a linear polynomial tail for each kind of radial basis function type.

Name	$\phi(r) =$	μ_ρ
Linear	r	0
Cubic	r^3	1
Thin plate spline	$r^2 \log r$	1

3 Description of Individual m-functions

3.1 socemo.m

3.1.1 socemo.m inputs

The main function from which to run the algorithm is socemo.m. socemo.m takes three (3) input arguments shown in Table 2.

socemo(datafile, maxeval, setrandseed)

Table 2: Parameter inputs for socemo.m.

Input #	Name	Description
1	datafile	string, mandatory, name of the file containing the user's problem definition
2	maxeval	integer, mandatory, maximum number of allowed function evaluations; must be larger than dimension + 1
3	setrandseed	integer, mandatory, seed for random number generator

The first input argument (datafile), is a string with the name of the m-file in which your optimization problem is defined. We recommend using one of the examples that come with SOCEMO as template for writing your own datafile. The datafile must have as output argument a structure array Data (function call: Data = your_filename). In the datafile, all relevant problem information must be defined. We need the lower bounds (Data.xlow) and upper bounds (Data.xup) for *each* variable. You must define the problem dimension (Data.dim), the number of objective functions (Data.nr_obj), and the function handle for the objective functions (Data.objfunction). For example,

Data.xlow = [-4, -4];
Data.xup = [4, 4];
Data.dim = 2;
Data.nr_obj = 2
Data.objfunction = @(x) function_handle(x).

The objective function must be defined such that the input variable vector x is a row vector and it must return a row vector of k =Data.nr_obj objective function values, i.e., each variable input vector within the variable lower and upper bounds must be evaluable and every objective function value has to be in $\mathbb{R} \setminus \{\infty, -\infty\}$.

The input maxeval has to be an integer number. It defines the maximum number of allowable function evaluations (note that one function evaluation means that we get all k objective function values). In order to fit an RBF model, we need at least $d + 1$ function evaluations, and therefore maxeval has to be larger than $d + 1$. The number of allowable function evaluations depends on the user's time constraint and the computation time needed to get all k function values for one sample point. The algorithm stops after the maximum number of function evaluations has been reached.

The last input argument, setrandseed, sets the random number seed. Since the algorithm has a stochastic component when generating sample points, we can use the random number seed to do a re-run of the algorithm with the exact same random numbers. If you want to run the algorithm for the same problem more than once with different random numbers, simply use a different random number seed for each run.

3.1.2 socemo.m algorithm steps

socemo.m does the optimization search over the unit hypercube and rescales evaluation points to their true ranges for function evaluations. Hence, SOCEMO starts by setting the search space size to the unit hypercube. SOCEMO then generates an initial symmetric Latin hypercube design (slhd.m) with n_start=2(Data.dim+1) initial points. We check if the rank condition for matrix $\mathbf{P}$ in equation (4) is satisfied. If the condition is not satisfied, we add randomly generated points from the search domain to the initial design until the condition is satisfied. We then do the expensive function evaluations at the initial points.

Next, SOCEMO finds the set of strictly non-dominated solutions among the already evaluated points. We need at least k such solutions (since we will later fit a piecewise linear approximation function to the Pareto front). If we do not have sufficiently many linearly independent non-dominated solutions, we use MATLAB's multi-objective genetic algorithm to solve a surrogate problem in which we minimize the surrogate models of all objectives simultaneously:

$$\min_{\mathbf{x} \in \mathbb{R}^d} \; [s_1(\mathbf{x}), s_2(\mathbf{x}), \ldots, s_k(\mathbf{x})]^T \tag{5a}$$

$$-\infty < x_j^l \leq x_j \leq x_j^u < \infty, j = 1, \ldots, d \tag{5b}$$

SOCEMO then uses a variety of sampling methods to generate new points at which we evaluate the expensive objective functions. We cycle through five sample point selection strategies until we have exhausted the budget of allowable function evaluations. In sampling strategy 1, we generate a surrogate approximation of the current Pareto front. On this Pareto front, we find the point $\boldsymbol{\tau} = [\tau_1, \tau_2, \ldots, \tau_k]^T$ that maximizes the minimum distance to all other current Pareto front points. This point defines a target value for each objective function. We find the point $\mathbf{x}$ in the parameter space that solves the multi-objective auxiliary optimization problem

$$\min_{\mathbf{x}\in\mathbb{R}^d} \left[|s_1(\mathbf{x}) - \tau_1|, |s_2(\mathbf{x}) - \tau_2|, \ldots, |s_k(\mathbf{x}) - \tau_k|\right]^T \tag{6a}$$

$$-\infty < x_j^l \leq x_j \leq x_j^u < \infty, j = 1, \ldots, d. \tag{6b}$$

If this problem does not have a unique solution, we choose as solution $\arg\min_{\mathbf{x}\in\mathcal{G}} \sum_{i=1}^{k} |s_i(\mathbf{x}) - \tau_i|$, where $\mathcal{G}$ is the solution set of (6).
The second sampling strategy is based on adding random perturbations to the currently non-dominated sample points. We use as scoring criterion a weighted sum of function value predictions by the surrogate models and the distance of the perturbed points to already evaluated points.
The third sampling strategy is based on finding the minimum of each objective function's surrogate model. Thus, in this step we may generate up to k new sample points. This sampling strategy enables us to examine the extrema of the Pareto front.
In sampling strategy 4, we randomly generate points from the whole parameter domain and we use the same scoring criteria as in strategy 2 in order to select a new evaluation point.
Sampling strategy 5 uses MATLAB's multi-objective genetic algorithm to solve the multi-objective surrogate problem (5). We evaluate the expensive objective functions only at a subset of the Pareto solutions of the surrogate problem.
In between sampling strategies and whenever we do expensive evaluations, we update our set of currently non-dominated sample points and their function values. After the budget of function evaluations (maxeval) is exhausted, SOCEMO saves all problem and sample information as a data structure Data in results.mat.

3.2 slhd.m

slhd.m selects n_start points as initial experimental design by generating a symmetric Latin hypercube design [4]. The input parameter is the structure array Data that contains the information about how many starting points are needed and what the problem dimension is. The output is a matrix with d (dimension) columns and n_start rows.

3.3 find_targetvalues.m

find_targetvalues.m computes target values $\boldsymbol{\tau}$ for all objective functions. The function uses a piecewise linear approximation of the Pareto front as a constraint on which it searches for a vector $\boldsymbol{\tau}$ that maximizes the minimum distance to all other points that are currently on the Pareto front. The goal is to find a new sample point in the variable domain that assumes these target values and thus closes gaps on the current Pareto front.

3.4 rbf_minus_tv.m

rbf_minus_tv.m uses the target values computed by find_targetvalues.m and searches for the point $\mathbf{x}$ in the parameter space that assumes these target values. We solve the multi-objective surrogate problem (6) with MATLAB's multi-objective genetic algorithm, where s_i is the cubic RBF approximation of f_i. Ideally, the objectives are not conflicting, and we find one point that minimizes all objectives simultaneously. If this is not the case (and we find several trade-off solutions), we choose as new sample point the point that minimizes the sum of the individual objective functions of (6). Note that in rbf_minus_tv.m, we call MATLAB's gamultiobj.m function, and we adjust the number of generations and the population size. Both numbers can be increased to possibly improve the solution quality of (6), but this comes at a cost of computational overhead of the optimizer.

3.5 pert_sampling.m

pert_sampling.m generates new evaluation points by adding random perturbations to the currently non-dominated points. We generate 500 randomly perturbed points in the vicinity of each non-dominated point. We score the "quality" of the random points by predicting their objective function values with the surrogate models and by computing their distances to already evaluated points. Both criteria are weighted in compute_scores_mo.m and the points with the best scores are selected for evaluation with the expensive objectives.

3.6 compute_scores_mo.m

compute_scores_mo.m computes for every candidate sample point (which was created by the perturbation method pert_sampling.m) a weighted sum of two criteria, namely the predicted objective function values and the distance to already evaluated points. We scale the objective function values for each objective to [0,1] and use the mean over all objectives for each candidate as first criterion. We also scale the distance values to [0,1] to obtain the second criterion. We give a higher weight (and therefore a higher importance) to the objective function value criterion.

3.7 solve_surr_problem.m

solve_surr_problem.m solves a multi-objective auxiliary problem in which we simultaneously minimize the surrogate approximations of the objective functions (see equations (5)). Since evaluating the surrogate models is computationally inexpensive, we use again MATLAB's multi-objective genetic algorithm. Similar to rbf_minus_tv.m, one may adjust the population and generation numbers in order to possibly improve the solution quality (at a cost of computational overhead).

3.8 maximindist_decispace.m

maximindist_decispace.m finds a new sample point by maximizing the minimum distance to all already evaluated points.

3.9 check_distance.m

check_distance.m computes the distance of every point in a new set of potential sample points to the set of already evaluated points. Any new point that is closer than a given tolerance to the set of already evaluated points will be discarded and not considered for evaluation with the expensive functions.

3.10 fix_rank_surr.m

fix_rank_surr.m is used when we do not have sufficiently many non-dominated points to fit a surrogate surface for the Pareto front. In order to approximate the Pareto front, we require that

$$\operatorname{rank}\left(\begin{bmatrix} f_1(\tilde{\mathbf{x}}_\mathbf{1}) & f_2(\tilde{\mathbf{x}}_\mathbf{1}) & \dots & f_{k-1}(\tilde{\mathbf{x}}_\mathbf{1}) & 1 \\ f_1(\tilde{\mathbf{x}}_\mathbf{2}) & f_2(\tilde{\mathbf{x}}_\mathbf{2}) & \dots & f_{k-1}(\tilde{\mathbf{x}}_\mathbf{2}) & 1 \\ \vdots & \vdots & \ddots & \vdots & \\ f_1(\tilde{\mathbf{x}}_\mathbf{P}) & f_2(\tilde{\mathbf{x}}_\mathbf{P}) & \dots & f_{k-1}(\tilde{\mathbf{x}}_\mathbf{P}) & 1 \end{bmatrix}\right) = k, \tag{7}$$

where $\tilde{\mathbf{x}}_\mathbf{1}, \dots, \tilde{\mathbf{x}}_\mathbf{P}$ are the points in the set of non-dominated solutions. In order to increase this set and to satisfy the rank condition, we use MATLAB's multi-objective genetic algorithm and solve the surrogate problem (5).

3.11 check_dominance.m and check_dominance2.m

check_dominance.m finds the non-dominated points within a single set of points. check_dominance2.m finds the non-dominated points from two separate point sets. Both functions keep solutions that have the exact same function values.

3.12 check_strict_dominance.m and check_strict_dominance2.m

check_strict_dominance.m finds the strictly non-dominated points within a single set of points. check_strict_dominance2.m finds the strictly non-dominated points from two separate point sets. Both functions discard points that have the exact same function values.

3.13 rbf_params_mo.m

rbf_params_mo.m computes the RBF parameters for all objective functions.

3.14 rbf_prediction_mo.m

rbf_prediction_mo.m predicts for a sample point all objective function values using the RBF surrogate models.

3.15 rbf_prediction.m

rbf_prediction.m predicts for a sample point one objective function value at a time using the RBF surrogate model. This function is needed when we sample by using the minimum point of each RBF surrogate surface.

3.16 rbf_matrices.m

rbf_matrices.m assembles the matrices needed for solving the linear system (3).

3.17 rbfvalue.m

rbfvalue.m computes the value $\phi(r)$ (for the cubic RBF, $\phi(r) = r^3$). See Table 1 for other options.

3.18 datainput_mop1.m, datainput_mop2.m, datainput_mop3.m

datainput_mop1.m, datainput_mop2.m, and datainput_mop3.m are example datainput files. A datainput file is mandatory for socemo.m. Use one of these examples as template for defining your own problem. See Section 3.1.1 for further details.

3.19 pareto_plot.m

pareto_plot.m can be used to plot the Pareto front for problems that have two or three objective functions. This function can optionally be called from the command line after socemo.m has finished and the file results.mat has been generated.

3.20 testdriver.m

testdriver.m can be run directly from the command line. It solves a test problem and plots the Pareto front.

4 SOCEMO Output

socemo.m generates a file, results.mat that contains the complete sample history, the final non-dominated points, and all problem information supplied by the user. In order to access the data, type

```
load results.mat
```

into the command prompt in MATLAB (make sure that the results.mat file is located in a directory known to the MATLAB search path). A structure array Data will appear in the workspace. The fields of Data are described in Table 3.

Table 3: Fields of the structure array Data.

Field name	Description
dim	Scalar, problem dimension (d)
xlow	Vector, variable lower bounds ($x_j^l, j = 1, \ldots, d$)
xup	Vector, variable upper bounds ($x_j^u, j = 1, \ldots, d$)
nr_obj	Scalar, number of objective functions (k)
objfunction	Objective function handle
lb	Vector of zeros, sampling decisions are made in unit hypercube
ub	Vector of ones, sampling decisions are made in unit hypercube
maxeval	Maximum number of allowed function evaluations (1 evaluation = all k objective function values)
tol_same	Scalar, distance tolerance below which two points are considered equal
m	Scalar, number of function evaluations done
S	Matrix (m $\times$ dim) with evaluated points
Y	Matrix (m $\times$ nr_obj) with objective function values
S_nondom	Matrix ($P\times$ dim) with P non-dominated points
Y_nondom	Matrix ($P\times$ nr_obj) with function values of non-dominated points
S_for_tv	Matrix with strictly non-dominated points (submatrix of S_nondom)
Y_for_tv	Matrix with function values of strictly non-dominated points (submatrix of Y_nondom)
totaltime	Total time needed for optimization

5 Example

In this section, we show an example of how to define an optimization problem and use socemo.m to solve it. You must provide a data file (see Section 3.1.1 for the details). The data file contains all information about the optimization problem. SOCEMO comes with three test functions (datainput_mop1.m, datainput_mop2.m, datainput_mop3.m). We recommend using one of these files as template for defining your own problem. Section 3.1.1 shows the mandatory problem specifications that must be given for all problems.

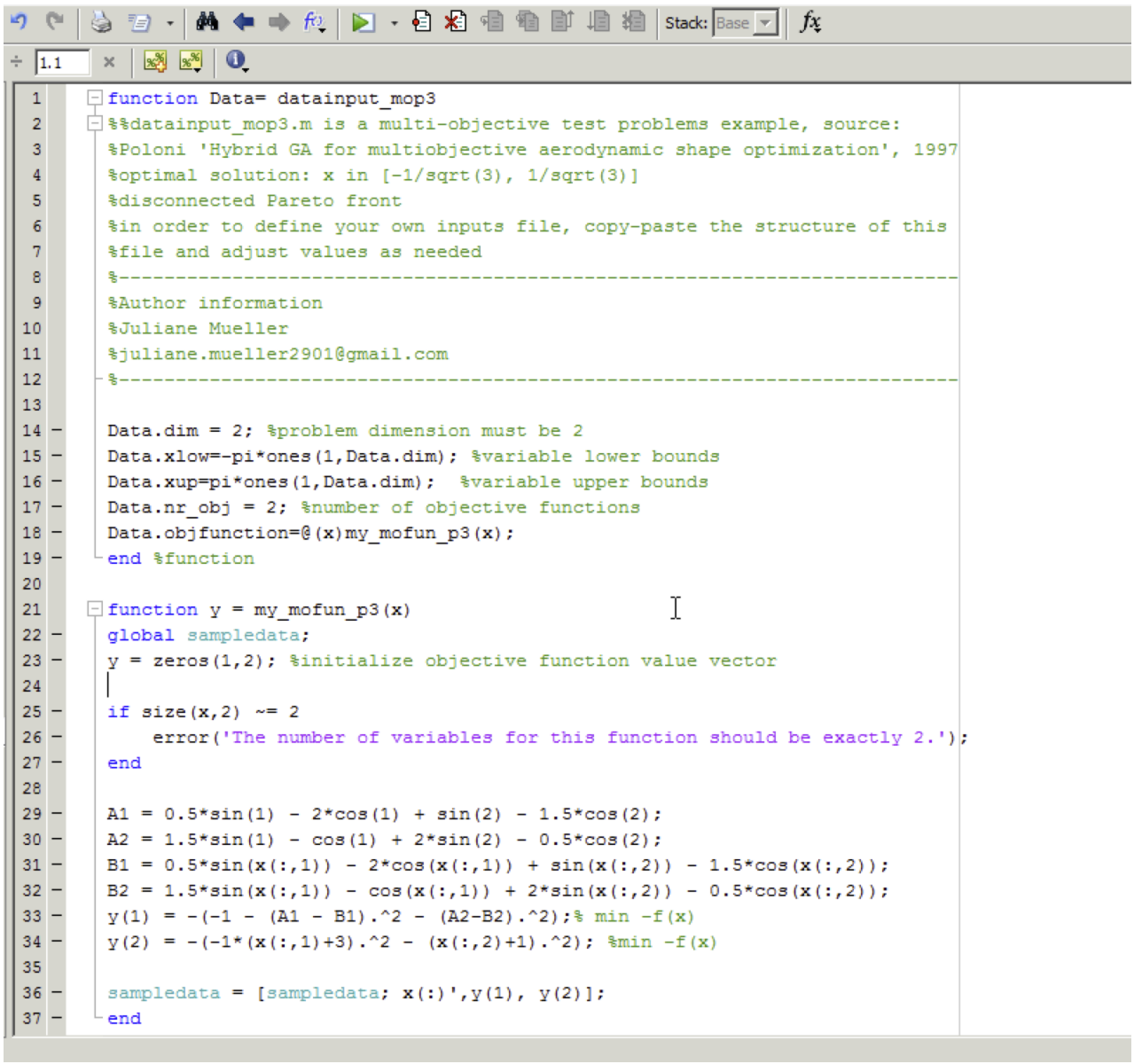
When defining the objective function, you have to include the command

```
global sampledata
```

This global variable collects the sample points and their corresponding function values. sampledata is a matrix with m rows and dim + nr_obj columns that gets dynamically increased whenever we evaluate a new point. Define your problem such that the output of your objective function definition is a row vector y. Last, collect the new data in the global variable sampledata by using the command

```
sampledata = [sampledata; x(:)', y];
```

Your data input file should look similar to the code shown in Figure 2.



```
function Data= datainput_mop3
%%datainput_mop3.m is a multi-objective test problems example, source:
%Poloni 'Hybrid GA for multiobjective aerodynamic shape optimization', 1997
%optimal solution: x in [-1/sqrt(3), 1/sqrt(3)]
%disconnected Pareto front
%in order to define your own inputs file, copy-paste the structure of this
%file and adjust values as needed
%--------------------------------------------------------------------------
%Author information
%Juliane Mueller
%juliane.mueller2901@gmail.com
%--------------------------------------------------------------------------

Data.dim = 2; %problem dimension must be 2
Data.xlow=-pi*ones(1,Data.dim); %variable lower bounds
Data.xup=pi*ones(1,Data.dim);  %variable upper bounds
Data.nr_obj = 2; %number of objective functions
Data.objfunction=@(x)my_mofun_p3(x);
end %function

function y = my_mofun_p3(x)
global sampledata;
y = zeros(1,2); %initialize objective function value vector

if size(x,2) ~= 2
    error('The number of variables for this function should be exactly 2.');
end

A1 = 0.5*sin(1) - 2*cos(1) + sin(2) - 1.5*cos(2);
A2 = 1.5*sin(1) - cos(1) + 2*sin(2) - 0.5*cos(2);
B1 = 0.5*sin(x(:,1)) - 2*cos(x(:,1)) + sin(x(:,2)) - 1.5*cos(x(:,2));
B2 = 1.5*sin(x(:,1)) - cos(x(:,1)) + 2*sin(x(:,2)) - 0.5*cos(x(:,2));
y(1) = -(-1 - (A1 - B1).^2 - (A2-B2).^2);% min -f(x)
y(2) = -(-1*(x(:,1)+3).^2 - (x(:,2)+1).^2); %min -f(x)

sampledata = [sampledata; x(:)',y(1), y(2)];
end
```

Figure 2: Example input datafile for the Poloni [2] test function.

In order to use socemo.m to solve your problem, you must know how many function evaluations (maxeval) you want to allow (here, one function evaluation corresponds to evaluating all objective functions at one point) and you must give your random number seed (setrandseed). Let's assume we want to solve the Poloni problem in Figure 2, we allow 200 function evaluations, and we use the random seed 1. That means, from the command line, we call

```
socemo('datainput_mop3', 200,1)
```

After the algorithm has finished, you will find the file results.mat in the current MATLAB directory (see Section 4 for details of the results file). If you have only two or three objective functions, you can plot the Pareto front by typing into the command window:

```
load results.mat;
pareto_plot(Data);
```

For the above example, the Pareto front is shown in Figure 3.

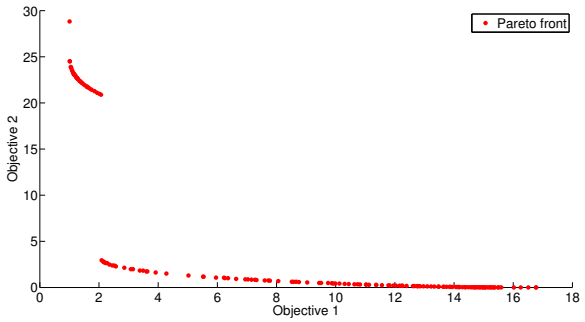
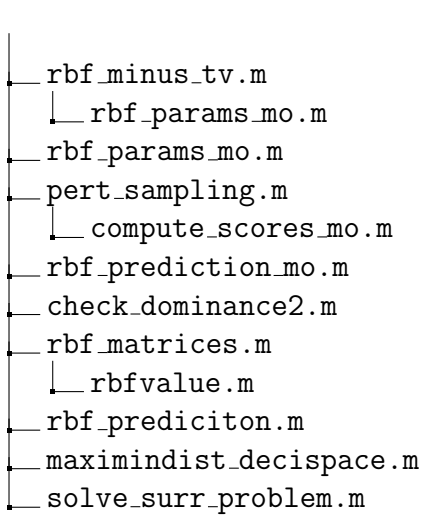


Figure 3: Pareto front obtained by SOCEMO for the Poloni [2] test function.

6 SOCEMO Code Structure

We outline the code structure here. Functions in subtrees indicate that they are called by a higher level function.

```
socemo.m
├── slhd.m
├── check_distance.m
├── check_dominance.m
├── check_strict_dominance.m
├── fix_rank_surr.m
│   ├── rbf_params_mo.m
│   ├── check_dominance.m
│   ├── check_distance.m
│   ├── check_strict_dominance.m
│   └── check_strict_dominance2.m
└── find_targetvalues.m
```



References

[1] MATLAB. *MATLAB R2012a*. The MathWorks Inc., Natick, Massachusetts, 2012.

[2] C. Poloni. HYBRID GA for multi-objective aerodynamic shape optimization. In G. Winter, J. Periaux, M. Galan, and P. Cuesta, editors, *Genetic Algorithms in Engineering and Computer Science*, pages 397–416. Wiley & Sons Chichester, UK, 1995.

[3] M.J.D. Powell. *The Theory of Radial Basis Function Approximation in 1990*. Advances in Numerical Analysis, vol. 2: wavelets, subdivision algorithms and radial basis functions. Oxford University Press, Oxford, pp. 105-210, 1992.

[4] K.Q. Ye, W. Li, and A. Sudjianto. Algorithmic construction of optimal symmetric Latin hypercube designs. *Journal of Statistical Planning and Inference*, 90:145–159, 2000.