MATSuMoTo Code Documentation

Juliane Müller

Cornell University
School of Civil and Environmental Engineering
Ithaca, NY, USA
email: juliane.mueller2901@gmail.com

March 28, 2014

1 Introduction

This documentation accompanies MATSuMoTo, the MATLAB Surrogate Model Toolbox for deterministic computationally expensive black-box global optimization problems. MATSuMoTo requires MATLAB version 2010b or newer. MATSuMoTo is intended for computationally expensive black-box global optimization problems with continuous, integer, or mixed-integer variables that are formulated as minimization problems. We refer with "computationally expensive" to optimization problems whose objective function evaluation takes a considerable amount of time (from several minutes to several hours or more). Such objective function evaluations may require, for example, running a computer simulation and hence the analytical description of the objective function is not available (black box). Furthermore, these objective functions are generally multimodal, i.e. there are several local minima and the goal is to find the global minimum.

MATSuMoTo contains ideas from the following published papers that should be cited when the toolbox is used:

1. J. Müller and R. Piché, 2011. "Mixture Surrogate Models Based on Dempster-Shafer Theory for Global Optimization Problems", Journal of Global Optimization, 51:79-104
2. J. Müller, C.A. Shoemaker, and R. Piché, 2013. "SO-MI: A Surrogate Model Algorithm for Computationally Expensive Nonlinear Mixed-Integer Black-Box Global Optimization Problems", Computers and Operations Research, 40(5):1383-1400
3. J. Müller, C.A. Shoemaker, and R. Piché, 2013. "SO-I: A Surrogate Model Algorithm for Expensive Nonlinear Integer Programming Problems Including Global Optimization Applications", Journal of Global Optimization, DOI 10.1007/s10898-013-0101-y
4. J. Müller and C.A. Shoemaker, 2014. "Influence of Ensemble Surrogate Models and Sampling Strategy on the Solution Quality of Algorithms for Computationally Expensive Black-Box Global Optimization Problems", Journal of Global Optimization, DOI 10.1007/s10898-014-0184-0

For better comprehension of the algorithmic concepts in MATSuMoTo, we recommend reading these papers. However, it is also possible to learn how to use MATSuMoTo "just as a tool" by going through the examples provided in this document and formulating own optimization problems by using these examples as templates. The code is thoroughly commented and we encourage the interested user to look also at the code for implementation details.

This document is structured as follows. In Section 2, we summarize the steps of a general surrogate model algorithm to give the user a broad idea of how MATSuMoTo works. The installation and software requirements are briefly described in Section 3. Section 4 describes how to use the test driver to check if the installation was successful. Section 5 describes the options for the user's input arguments. In Section 6, we briefly describe how parallelism is exploited. The output of MATSuMoTo is explained in Section 7. Section 8 contains few more examples of how to use MATSuMoTo. The dependencies of the single functions in the code are shown in Appendix A.

Finally, should you have any questions or encounter any bugs, please feel free to contact me at the email address `juliane.mueller2901@gmail.com`.

2 Surrogate Model Algorithms

Computer simulations are used to approximate complex physical behavior. These simulations are often computationally expensive, and thus during an optimization the goal is to find optimal solutions by using only very few of these expensive function evaluations. Surrogate models (also known as response surfaces or metamodels) are used to approximate the expensive simulation model [1]. During the optimization phase information from the surrogate model is used to guide the search for improved solutions. The predictions of the surrogate model at unsampled points are used to carefully select new points where the computationally expensive objective function will be evaluated. This approach reduces the computation time of the optimization considerably because fewer expensive function evaluations are required for finding the optimum. Most surrogate model algorithms consist of the steps shown in the algorithm below and illustrated in Figure 1.

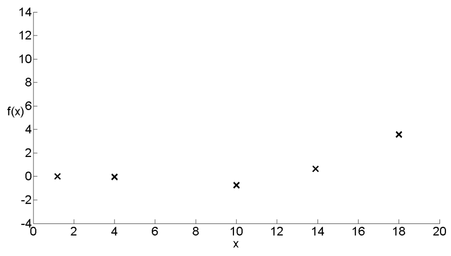
Algorithm ***General Surrogate Model Algorithm***

1. Generate an initial experimental design.
2. Do the costly function evaluations at the points generated in Step 1.
3. Fit a surrogate model to the data.
4. Use the surrogate model to predict the objective function values at unsampled points in the variable domain to decide at which point(s) to do the next expensive function evaluation(s).
5. Do the expensive function evaluation(s) at the point(s) selected in Step 4.
6. Check if the stopping criterion has been reached. If not, got to Step 3. If the stopping criterion has been met, stop.

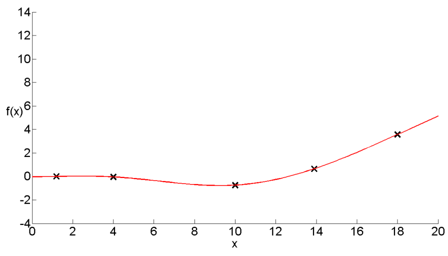
Surrogate model algorithms in the literature differ mainly with respect to

- the strategy for generating the initial experimental design;
- the chosen surrogate model;
- the strategy for selecting the sample point(s) in each iteration.

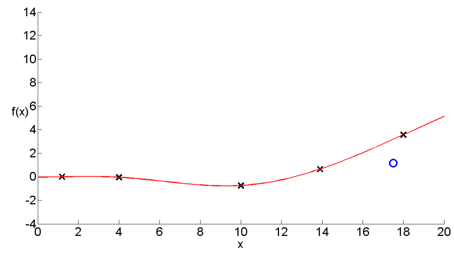
Typically used stopping criteria are a maximum number of allowed function evaluations (used in MATSuMoTo), a maximal allowed CPU time, or a maximum number of failed iterative improvement trials.



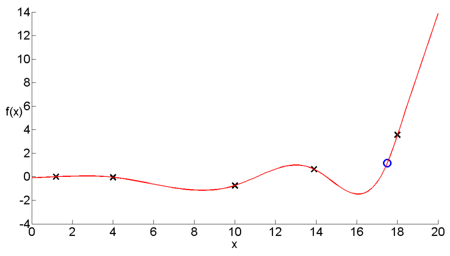
(a) Initial experimental design (black crosses).



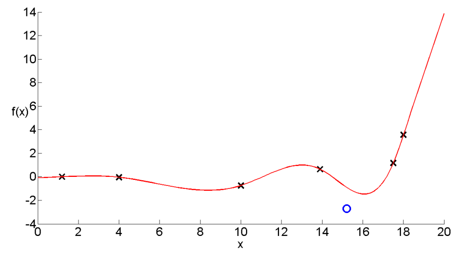
(b) Fit the surrogate model (red graph).



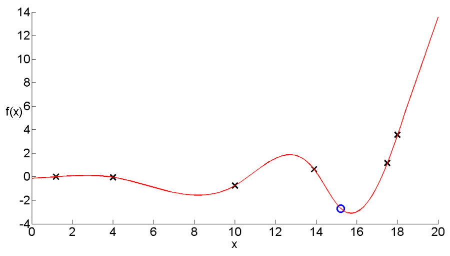
(c) Selected a new data point (blue circle).



(d) Update the surrogate model (red graph).



(e) Selected a new data point (blue circle).



(f) Update the surrogate model (red graph).

Figure 1: Illustration of the steps of the surrogate model algorithm with a one-dimensional problem. Black crosses denote points that have already been evaluated in previous iterations, blue circles denote newly chosen data points, the red graph illustrates the interpolating surrogate model.

3 Installation

MATSuMoTo requires MATLAB version 2010b or newer and the MATLAB Statistics Toolbox. To use MATSuMoTo, download the file MATSuMoTo.zip and unzip it in a location known to the MATLAB search path. Alternatively, you can add a new folder to the MATLAB search path by clicking in the MATLAB window on

Set Path... $\rightarrow$ Add with Subfolders

and browse to the location of the unzipped folder MATSuMoTo. Click on the folder, and click on Open. In the MATLAB Set Path window, click on Save and subsequently on Close.

Depending on the options selected, MATSuMoTo requires also the following MATLAB toolboxes:

- MATLAB Optimization Toolbox when using option SurfMin for continuous problems
- MATLAB Global Optimization Toolbox when using option SurfMin for pure integer and mixed-integer problems
- MATLAB Parallel Computing Toolbox when it is desired to evaluate several sample points simultaneously

MATSuMoTo can also be used without the optimization and parallel computing toolboxes, but the option of selecting the minimum point of the surrogate model as new sample point will be disabled (for options of selecting sample points, see Section 5.3).

4 Test Driver

You can try if your installation was successful by using the test driver included in the toolbox. Type

```
>> TestDriver
```

into the MATLAB command prompt. If you get an error similar to

```
??? Undefined function or method 'TestDriver' for input arguments  of type 'char'
```

then MATLAB does not know the folder in which to look for the files, indicating that something went wrong when adding the folder to MATLAB's search path. Should you get an error similar to

```
Error using feval
Undefined function or variable 'datainput_hartman3'
```

then not all subfolders were added to the MATLAB search path. Go back to Section 3, and select Add with Subfolders when setting the MATLAB search path.

If no error messages appear, the test driver will run the optimization of a continuous problem, a pure integer problem, and a mixed-integer problem. You will see messages appearing in the command prompt such as "Testing continuous problem", messages with information about the best objective function value found so far, and "Continuous problem finished successfully". Similar messages will appear when the pure integer and mixed-integer problems are tested. After the optimization of each problem has finished, a figure that shows a progress plot of the objective function value improvement will appear (your figures should look similar to Figures 2-4).

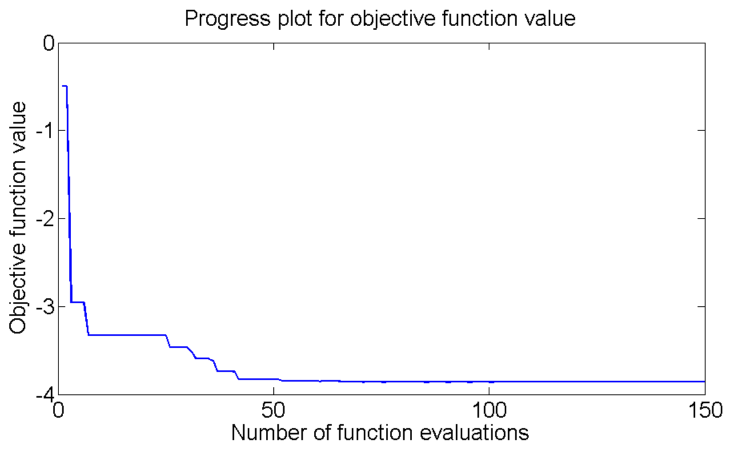


Figure 2: Progress plot for the continuous test problem.

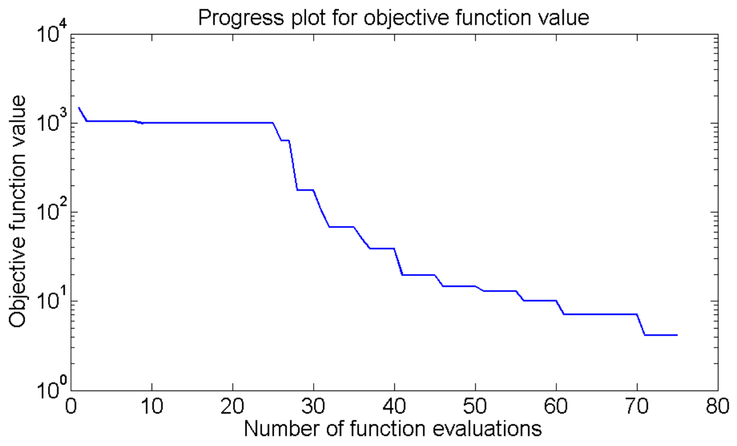


Figure 3: Progress plot for the integer test problem.

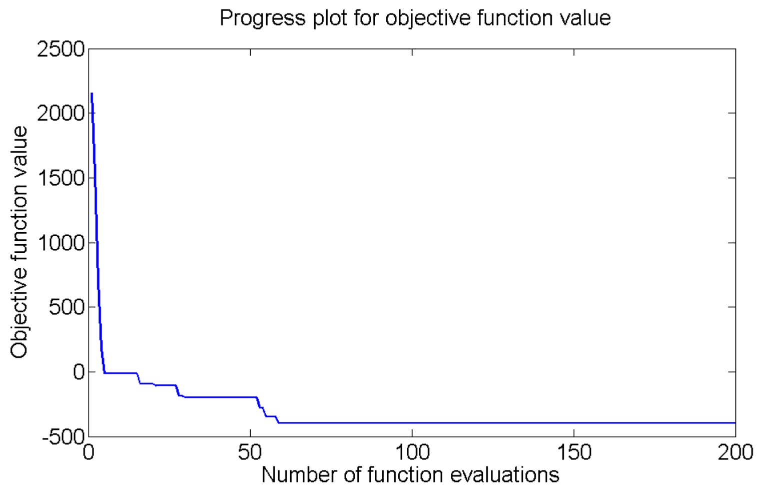


Figure 4: Progress plot for the mixed-integer test problem.

5 User Specified Input

The main file from which to start MATSuMoTo is MATSuMoTo.m. The user can define several input arguments (see the details later in this section). The algorithm starts by checking the user's input (function InputCheck.m) and assigns default values to options that have not been set by the user as input arguments. Then, depending on whether the optimization problem is continuous, integer, or mixed-integer, the corresponding optimization function is called (Optimization_continuous,Optimization_integer, or Optimization_mixedinteger). After the optimization routine has finished, the results are stored in the file Results.mat (see Section 7 for what is stored in this file).

MATSuMoTo has many different options that the user can set by supplying input arguments. Only the first input argument must be defined, which is the name of the MATLAB file in which the user's optimization problem is specified (see Section 5.1). Additional input arguments such as the maximal number of allowed function evaluations, the surrogate model to be used, the sampling strategy, the initial experimental design technique, the number of points in the initial experimental design, specific starting points, or the number of points to be selected in every iteration of the algorithm are optional.

The general function call is as follows:

```
>> [xbest, fbest] = MATSuMoTo(data_file,maxeval,surrogate_model,sampling_technique,...
    initial_design,number_startpoints,starting_point,NumberNewSamples);
```

The individual input arguments are summarized in Table 1 and are in more detail described in Sections 5.1-5.6.

Table 1: Input arguments, M = mandatory, O = optional, d = problem dimension

Input argument	M/O	Description
data_file	M	String with name of data file containing optimization problem data, see Section 5.1
maxeval	O	Positive integer defining maximum number of allowed function evaluations (default $20d$)
surrogate_model	O	String defining which surrogate model to use (default 'RBFcub'), see Section 5.2
sampling_technique	O	String defining the technique for selecting the next sample site (default 'CANDglob'), see Section 5.3
initial_design	O	String defining the initial experimental design technique (default 'LHS'), see Section 5.4
number_startpoints	O	Positive integer defining the number of initial starting points (default $2(d+1)$), see Section 5.5
starting_point	O	Matrix with specific points to be added to the initial experimental design (default []), see Section 5.5
NumberNewSamples	O	Integer determining the number of points to be evaluated in every iteration (default 1), see Section 5.6

5.1 Input Argument data_file

The data file has no input argument, and one output argument (the structure Data). The data structure has to contain all the necessary problem information shown in Table 2. For an example, open the file datainput_convex_MI.m in the folder ExampleMixedInteger. You should see the following definitions:

```
Data.xlow=-10*ones(1,8); %variable lower bounds
Data.xup=10*ones(1,8);  %variable upper bounds
Data.dim = 8; %problem dimension
Data.integer=(1:4); %indices of integer variables
Data.continuous=(5:8); %indices of continuous variables
Data.objfunction=@(x) 3.1*x(:,1).^2 + 7.6* x(:,2).^2 +6.9*x(:,3).^2 +0.004*x(:,4).^2 +...
    +19*x(:,5).^2 +3*x(:,6).^2 +x(:,7).^2  +4*x(:,8).^2 ; %objective function handle
```

Note that this is a computationally cheap test problem. For a computational expensive simulation, you must define a function handle for Data.objfunction that calls your simulation model.

Table 2: Elements of the data file defining the optimization problem

Structure element	Description
Data.xlow	Variable lower bounds, row vector with d (=dimension) entries
Data.xup	Variable upper bounds, row vector with d (=dimension) entries
Data.dim	Problem dimension, positive integer
Data.integer	Row vector containing indices of variables with integer constraints (Data.integer=[] if no integrality constraints)
Data.continuous	Row vector containing indices of continuous variables (Data.continuous=[] if no continuous variables)
Data.objfunction	Handle to objective function, must return a scalar value

5.2 Input Argument surrogate_model

There are several options for choosing the surrogate model. Surrogate models can be interpolating (for example, radial basis functions (RBF)) or non-interpolating (for example, polynomial regression models, multivariate adaptive regression splines (MARS)). MATSuMoTo allows the user to choose between different models and ensembles (see Table 3). The surrogate model to be used can be defined by the user as string in place of the input argument surrogate_model in the general function call described at the beginning of this section. When using the MARS model, the ARESLab toolbox [2] is used.

Table 3: Choices for surrogate models and ensembles

surrogate_model	Description
'RBFcub'	Cubic radial basis function interpolant, default
'RBFtps'	Thin-plate spline radial basis function interpolant
'RBFlin'	Linear radial basis function interpolant
'POLYlin'	Linear regression polynomial
'POLYquad'	Full quadratic regression polynomial
'POLYquadr'	Reduced quadratic regression polynomial
'POLYcub'	Full cubic regression polynomial
'POLYcubr'	Reduced cubic regression polynomial
'MARS'	Multivariate adaptive regression spline [2]
'MIX_RcM'	Ensemble of 'RBFcub' and 'MARS'
'MIX_RcPc'	Ensemble of 'RBFcub' and 'POLYcub'
'MIX_RcPcr'	Ensemble of 'RBFcub' and 'POLYcubr'
'MIX_RcPq'	Ensemble of 'RBFcub' and 'POLYquad'
'MIX_RcPqr'	Ensemble of 'RBFcub' and 'POLYquadr'
'MIX_RcPcM'	Ensemble of 'RBFcub', 'POLYcub', and 'MARS'

5.3 Input Argument sampling_technique

The user can select between two sampling strategies, namely a randomized strategy and a strategy that uses the minimum point of the surrogate model as new evaluation point (see Table 4 for options). The sampling strategy to be used can be defined by the user as string in place of the input argument sampling_technique in the general function call described at the beginning of this section.

Table 4: Options for sampling strategies

sampling_technique	Description
'CANDloc'	Randomized sampling approach by local perturbation of the best point found so far
'CANDglob'	Like 'CANDloc', but with additional points uniformly selected from the whole variable domain, default
'SurfMin'	Uses the minimum point of the surrogate model

For 'CANDloc' a set of candidate points is generated by adding random perturbations to the best point found so far. For each candidate its distance to already sampled points and its predicted objective function value are used to define a score. The point with the best score is selected as new sample site [4] .

'CANDglob' works in the same way as 'CANDloc', but in addition to the points created by randomly perturbing the best point found so far also a set of points that is uniformly selected from the whole variable domain is generated and the score is calculated over both sets of points. In contrast to 'CANDloc' the global fit of the surrogate model can be improved whenever a point is favored that is far away from already sampled points.

'SurfMin' uses a (local) minimum point of the surrogate model as new sample point. For continuous problems, MATLAB's fmincon is used starting from a randomly selected point of the variable domain. For problems with integrality constraints, MATLAB's genetic algorithm ga is used. Hence, MATLAB's Optimization Toolbox and Global Optimization Toolbox, respectively, are required. The default settings are used for fmincon and ga, respectively. If the found minimum point of the surrogate model has already been evaluated in a previous iteration or if it is too close to an already evaluated point, the point that maximizes the minimum distance to all already evaluated points is selected as new sample point. Hence, it is possible to improve the global fit of the surrogate model and new areas of the variable domain where the global optimum may be located can be detected.

5.4 Input Argument initial_design

The user can choose between the initial experimental design strategies shown in Table 5. The initial experimental design strategy to be used can be defined by the user as string in place of the input argument initial_design in the general function call described at the beginning of this section.

Table 5: Initial experimental design strategies

initial_design	Description
'SLHD'	Symmetric Latin hypercube design
'LHS'	MATLAB's built-in Latin hypercube design (default)
'CORNER'	Uses (subset) of corner points of variable domain plus center point

'SLHD' generates a symmetric Latin hypercube design, whereas 'LHS' uses MATLAB's Latin hypercube design with maximin option and 20 iterations. 'CORNER' uses (a subset of) the corner points and the midpoint of the hypercube defined by the variables' upper and lower bounds. There are restrictions as to how many points must be contained in the initial experimental design. In general, any design strategy may be used if the design contains sufficiently many points for fitting the desired surrogate model. In the initial user input check at the beginning, MATSuMoTo verifies whether the number of points in the initial experimental design suffice. If not, the necessary minimum number of points is used. The 'CORNER' option is recommended only for relatively small problems (15 or fewer dimensions, since otherwise the machine may run out of memory when generating the design sites).

5.5 Input Arguments number_startpoints and starting_point

The number of points in the initial experimental design can be defined by the user as positive integer in place of the input argument number_startpoints in the general function call described at the beginning of this section. Additionally, the user can define m point(s) s/he wants to add to the points in the initial experimental design. The points must be given in matrix form $(m \times d)$, where d denotes the problem dimension and these points are added at the beginning of the experimental design, i.e. the given points will be evaluated first. The desired points to be added to the initial experimental design can be defined by the user as $(m \times d)$-matrix in place of the input argument starting_point in the general function call described at the beginning of this section.

The points in the initial experimental design are counted towards the maximum number of allowed function evaluations, i.e. if the maximum number of allowed function evaluations is set to 100 and there are 20 points in the initial experimental design, then only 80 points will be evaluated during the optimization phase.

The minimum number of required points for the initial experimental design depends on the chosen surrogate model and the initial experimental design strategy. Following are some recommendations about the number of points required for different starting designs and surrogate model types. Although MATSuMoTo checks the user input for possibly conflicting options, not all possibilities may be covered.

- When using SLHD, at least $2d$ points should be used because otherwise there are linear dependencies in the design matrix.
- When using CORNER, at most $2^d + 1$ points may be used because there are only 2^d corners and one midpoint. This option is not recommended when the problem dimension is larger than 15.
- When using the full quadratic polynomial regression model (POLYquad) at least $1 + 2d + \binom{d}{2}$ points are needed. Otherwise the least squares problem is underdetermined.
- When using the reduced quadratic polynomial regression model (POLYquadr), at least $2d + 1$ points are needed. Otherwise the least squares problem is underdetermined.
- When using the full cubic polynomial regression model (POLYcub), at least $1 + 3d + \binom{d}{2} + \binom{d}{3}$ points are needed. Otherwise the least squares problem is underdetermined.
- When using the reduced cubic polynomial regression model (POLYcubr), at least $3d + 1$ points are needed. Otherwise the least squares problem is underdetermined.
- When using model ensembles, in general at least the minimum number of required points plus one additional point are needed because of the cross-validation. For example, if using MIX_RcPq, at least $\max(d+1, 1+2d+\binom{d}{2})+1 = 2+2d+\binom{d}{2}$ points are required (the full quadratic regression polynomial needs at least $1 + 2d + \binom{d}{2}$ points, the cubic RBF needs at least $d + 1$ points, and one additional point is needed because of the leave-one-out cross-validation).

5.6 Input Argument **NumberNewSamples**

NumberNewSamples determines the number of sample points to be evaluated in each iteration of the algorithm. The desired number of sample points can be defined by the user as positive integer in place of the input argument NumberNewSamples in the general function call described at the beginning of this section. In general, any positive integer can be given, but it is recommended to not exceed the number of available processors (if parallel computing is an option). If more than one sample point is selected in each iteration, all sample points are evaluated simultaneously if MATLAB's Parallel Computing Toolbox is installed (see Section 6).

5.7 Default Input Arguments

The default values for all input arguments are:

- maxeval: $20d$, d =dimension
- surrogate_model: 'RBFcub'
- sampling_technique: 'CANDglob'
- initial_design: 'LHS'
- number_startpoints: $2(d + 1)$
- NumberNewSamples:1

Note, however, that these settings are not guaranteed to perform best among all options. For many problems surrogate model ensembles are more successful. Also, depending on the maximum number of allowed function evaluations, the settings 'CANDloc' or 'SurfMin' may be more efficient with respect to finding improvements within a very limited number of function evaluations (see also [3]). The algorithm is implemented such that if several consecutive improvement trials are unsuccessful, the algorithm starts from scratch, i.e. all points evaluated so far are discarded (but saved) and a new initial experimental design is built from which the improvement trials start. If the problem dimension is large compared to the total number of allowed function evaluations, then the probability of restarting from scratch is low.

5.8 Input Example

The following example calls MATSuMoTo for finding the minimum of the three-dimensional Hartmann function defined in the file datainput_hartman3.m (to execute this example, type example_section_5_8 into the MATLAB command window or, alternatively, open the file in the MATLAB editor and click on Run). The maximum number of function evaluations is set to 300, the surrogate model to be used is the thin-plate spline radial basis function ('RBFtps'). The global randomized sampling strategy is used ('CANDglob'), and the initial experimental design is built with MATLAB's Latin hypercube design ('LHS') with 15 points. Specific user-defined starting points are not given and only one point is selected in every iteration. The user is encouraged to try the example. A progress plot will appear and the data is recorded in the file Results.mat which you can find in the current working directory. The value xbest is the best point found during the optimization and fbest is the corresponding best function value.

```
data_file = 'datainput_hartman3'; %name of data file
maxeval = 300; %maximum number of allowed function evaluations
surogate_model = 'RBFtps'; %selected surrogate model type
sampling_technique = 'CANDglob'; %global randomized sampling strategy
initial_design = 'LHS'; %MATLAB's lhsdesign.m as initial design
number_startpoints = 15; %15 points in the initial experimental design
starting_point = []; %no user-specified points to be added to the initial design
NumberNewSamples = 1; %1 new point is selected in each iteration
[xbest, fbest] = MATSuMoTo(data_file,maxeval,surogate_model,sampling_technique,...
    initial_design,number_startpoints,starting_point,NumberNewSamples);
```

6 Parallel Evaluations

The algorithm is implemented such that several computationally expensive function evaluations can be done in parallel. There are two steps in the algorithm where parallelism can reduce the total computation time, namely when evaluating the points in the initial experimental design and if more than one point is being selected in each iteration. MATLAB's Parallel Computing Toolbox is required for doing evaluations in parallel. MATSuMoTo checks if the Parallel Computing Toolbox is installed and if so, MATSuMoTo automatically does the function evaluations in parallel using parfor. Note that MATSuMoTo needs to open a pool of workers when using parallel evaluations. In this case a message similar to

```
Starting matlabpool using the 'local' profile ... connected to 12 labs.
```

will appear in the command window.

7 Results

The algorithm saves the results of the optimization to the file Results.mat. The elements shown in Table 6 are contained in the saved structure Data. You can load the results by typing

```
>>load Results.mat
```

into the MATLAB command window. The individual structure elements can be accessed by typing, for example, Data.Y for the objective function values, into the MATLAB command window. The element Data.fbest contains the best function value found during the optimization. This element may have several entries depending on whether or not the algorithm found a local minimum and restarted from scratch in which case the best result of every such trial is recorded. Similarly, Data.xbest contains the corresponding best point found so far in each trial.

Table 6: Saved data structure elements

Data.	Description
continuous	Vector with indices of continuous variables
dim	Problem dimension
EvalsEachTrial	Number of function evaluations in each trial
fbest	Best objective function values found
fevaltime	Vector with time for each function evaluation
initial_design	Name of the initial experimental design strategy
integer	Vector with indices of integer variables
maxeval	Maximum number of allowed function evaluations
number_startpoints	Number of points in the initial experimental design
NumberNewSamples	Number of points selected in each iteration for evaluation
parallel_eval	Indicator if parallelism was exploited (1) or not (0)
Problem	Name of the optimization problem data file
S	Matrix with sample sites
sampling_technique	Name of the technique used to select evaluation points in each iteration
starting_point	User-defined points that are added to the initial experimental design
surrogate_model	Name of the surrogate model used during the optimization
TotalTime	Total computation time needed by the algorithm
trial	Number of starts from scratch when finding in a local minimum
xbest	Best points found during the optimization
xlow	Vector with variables' lower bounds
xup	Vector with variables' upper bounds
Y	Vector with objective function values

8 Examples

This section contains examples of computationally cheap continuous, integer, and mixed-integer black-box optimization test problems. The goal is here to clarify input options rather than solving computationally expensive problems. The input data files are supplied in the code package. The examples have computationally cheap objective functions in order to reduce the computation time when experimenting with MATSuMoTo.

8.1 Examples for Continuous Problems

MATSuMoTo is applicable to box-constrained continuous problems. If the problem has additional constraints, they can be incorporated by the user with a penalty term in the objective function. There are several examples of input data files in the folder ExampleContinuous.

The following example uses the four-dimensional Shekel function (to execute this example, type example_section_8_1 into the MATLAB command window or, alternatively, open the file in the MATLAB editor and click on Run). The maximum number of allowed function evaluations is 200, the used surrogate model is an ensemble of the cubic radial basis function interpolant and a reduced cubic regression polynomial. The randomized global sampling strategy is used and the initial experimental design is generated by the symmetric Latin hypercube sampling strategy. The number of points in the initial experimental design is not defined (MATSuMoTo will assign the default value) and there are no user-specified points to be added to the initial experimental design. In every iteration one new point is evaluated.

```
data_file = 'datainput_Shekel7'; %name of data file
maxeval = 200; %maximum number of allowed function evaluations
surogate_model = 'MIX_RcPcr'; %selected response surface
sampling_technique = 'CANDglob'; %global randomized sampling strategy
initial_design = 'SLHD'; %symmetric Latin hypercube design as initial design
number_startpoints = []; %default number of points for initial experimental design
starting_point = []; %no user-specified points to be added to the initial design
NumberNewSamples = 1; %1 new point is selected in each iteration
[xbest,fbest] = MATSuMoTo(data_file,maxeval,surogate_model,sampling_technique,...
    initial_design,number_startpoints,starting_point,NumberNewSamples);
```

8.2 Examples for Integer Problems

MATSuMoTo is applicable to optimization problems where all variables have integer constraints. If the problem has additional constraints, the user has to incorporate them with a penalty term in the objective function. MATSuMoTo contains example data input files for integer problems in the folder ExampleInteger. It is not advised to use a polynomial regression model when the problem has binary variables because in that case the matrix for solving the least-squares problem will be ill-conditioned. If there are binary variables, using a radial basis function surrogate model is advised.

The following example uses an eight-dimensional convex test problem (to execute this example, type example_section_8_2 into the MATLAB command window or, alternatively, open the file in the MATLAB editor and click on Run). The maximum number of allowed function evaluations is 300 and the used surrogate model is the cubic radial basis function. The surface minimum sampling strategy is used and the initial experimental design is generated by MATLAB's built-in Latin hypercube sampling strategy. The number of points in the initial experimental design is 19 and there are no user-specified points to be added to the initial experimental design. In every iteration two new points are evaluated.

```
data_file = 'datainput_convex_I'; %name of data file
maxeval = 300; %maximum number of allowed function evaluations
surogate_model = 'RBFcub'; %selected surrogate model type
sampling_technique = 'SurfMin'; %surface minimum sampling strategy
initial_design = 'LHS'; %MATLAB's Latin hypercube design as initial design
number_startpoints = 19; %19 points for initial experimental design
starting_point = []; %no user-specified points to be added to the initial design
NumberNewSamples = 2; %2 new points are selected in each iteration
[xbest,fbest] = MATSuMoTo(data_file,maxeval,surogate_model,sampling_technique,...
    initial_design,number_startpoints,starting_point,NumberNewSamples);
```

8.3 Examples for Mixed-Integer Problems

MATSuMoTo is applicable for mixed-integer problems. If the problem has additional constraints, the user has to incorporate them with a penalty term in the objective function. The example data input files for mixed-integer problems are contained in the folder ExampleMixedInteger. It is not advised to use a polynomial regression model when the problem has binary variables because in that case the matrix for solving the least-squares problem will be ill-conditioned. If there are binary variables, using a radial basis function surrogate model is advised.

The following example uses a five-dimensional test problem (to execute this example, type example_section_8_3 into the MATLAB command window or, alternatively, open the file in the MATLAB editor and click on Run). The maximum number of allowed function evaluations is 300, the used surrogate model is the thin-plate spline radial basis function. The global randomized sampling strategy is used and the initial experimental design is generated by MATLAB's built-in Latin hypercube sampling strategy. The number of points in the initial experimental design is 20 and one point is added to the initial experimental design. In every iteration four new points are evaluated.

```
data_file = 'datainput_ex1221_MI'; %name of data file
maxeval = 300; %maximum number of allowed function evaluations
surogate_model = 'RBFtps'; %selected surrogate model type
sampling_technique = 'CANDglob'; %global randomized sampling strategy
initial_design = 'LHS'; %MATLAB's Latin hypercube design as initial design
number_startpoints = 20; %20 points for initial experimental design
starting_point = [0, 1, 0.5604, 0.9019, 0.8903]; %user-specified feasible point to
%be added to the initial design
NumberNewSamples = 4; %4 new points are selected in each iteration
[xbest,fbest] = MATSuMoTo(data_file,maxeval,surogate_model,sampling_technique,...
    initial_design,number_startpoints,starting_point,NumberNewSamples);
```

A Code Structure

The structure of the code is outlined here. Depending on if the problem is purely integer, continuous, or mixed-integer, option (a), (b), or (c) is used, respectively. The user can choose between two sampling strategies (options (i) an (ii)). The tree-structure symbolizes which file calls other files.

```
MATSuMoTo.m
|__InputCheck.m
|__(a) Optimization_integer.m
|   |__StartingDesign.m
|   |   |__SLHD.m/lhsdesign.m/cornerpoints.m
|   |__SOI_OP.m
|       |__FitSurrogateModel.m
|       |   |__RBF.m/POLY.m/aresbuild.m
|       |   |__DempsterFor2models.m
|       |   |   |__RBF.m/RBF_eval.m/POLY.m/POLY_eval.m/aresbuild.m/arespredict.m
|       |   |   |__cc_calc.m/RMSE_calc.m/MAE_calc.m/MAD_calc.m
|       |   |   |__dempster_rule.m
|       |   |   |__Dempster_belpl.m
|       |   |__DempsterFor3models.m
|       |       |__RBF.m/RBF_eval.m/POLY.m/POLY_eval.m/aresbuild.m/arespredict.m
|       |       |__cc_calc.m/RMSE_calc.m/MAE_calc.m/MAD_calc.m
|       |       |__dempster_rule.m
|       |       |__Dempster_belpl.m
|       |       |__model2combi2_3mod.m
|       |       |__weights_in_combi.m
|       |__(i) Perturbtation_SOI.m
|       |__(i) SamplePointSelection.m
|       |   |__PredictFunctionValues.m
|       |       |__RBF_eval.m/POLY_eval.m/arespredict.m
|       |__(ii) SurfMin.m
|           |__RBF_eval.m/POLY_eval.m/arespredict.m
|           |__ga.m/MaximinD_i.m
|__(b) Optimization_continuous.m
    |__StartingDesign.m
        |__SLHD.m/lhsdesign.m/cornerpoints.m
        |__FitSurrogateModel.m
        |   |__RBF.m/POLY.m/aresbuild.m
        |   |__DempsterFor2models.m
        |   |   |__RBF.m/RBF_eval.m/POLY.m/POLY_eval.m/aresbuild.m/arespredict.m
        |   |   |__cc_calc.m/RMSE_calc.m/MAE_calc.m/MAD_calc.m
        |   |   |__dempster_rule.m
        |   |   |__Dempster_belpl.m
        |   |__DempsterFor3models.m
        |       |__RBF.m/RBF_eval.m/POLY.m/POLY_eval.m/aresbuild.m/arespredict.m
        |       |__cc_calc.m/RMSE_calc.m/MAE_calc.m/MAD_calc.m
        |       |__dempster_rule.m
        |       |__Dempster_belpl.m
        |       |__model2combi2_3mod.m
        |       |__weights_in_combi.m
        |__(i) Perturbation.m
        |__(i) SamplePointSelection.m
        |   |__PredictFunctionValues.m
        |       |__RBF_eval.m/POLY_eval.m/arespredict.m
        |__(ii) SurfMin.m
            |__RBF_eval.m/POLY_eval.m/arespredict.m
            |__fmincon.m/MaximinD_c.m
```

```
│
│
│
└── (c) Optimization_mixedinteger.m
    ├── StartingDesign.m
    │   ├── SLHD.m/lhsdesign.m/cornerpoints.m
    │   └── FitSurrogateModel.m
    │       ├── RBF.m/POLY.m/aresbuild.m
    │       ├── DempsterFor2models.m
    │       │   ├── RBF.m/RBF_eval.m/POLY.m/POLY_eval.m/aresbuild.m/arespredict.m
    │       │   ├── cc_calc.m/RMSE_calc.m/MAE_calc.m/MAD_calc.m
    │       │   ├── dempster_rule.m
    │       │   └── Dempster_belpl.m
    │       └── DempsterFor3models.m
    │           ├── RBF.m/RBF_eval.m/POLY.m/POLY_eval.m/aresbuild.m/arespredict.m
    │           ├── cc_calc.m/RMSE_calc.m/MAE_calc.m/MAD_calc.m
    │           ├── dempster_rule.m
    │           ├── Dempster_belpl.m
    │           ├── model2combi2_3mod.m
    │           └── weights_in_combi.m
    ├── (i) Perturbation_SOMI.m
    ├── (i) SamplePointSelection.m
    │   └── PredictFunctionValues.m
    │       └── RBF_eval.m/POLY_eval.m/arespredict.m
    └── (ii) SurfMin.m
        ├── RBF_eval.m/POLY_eval.m/arespredict.m
        └── ga.m/MaximinD_i.m
```

References

[1] A.J. Booker, J.E. Dennis Jr, P.D. Frank, D.B. Serafini, V. Torczon, and M.W. Trosset. A rigorous framework for optimization of expensive functions by surrogates. *Structural Multidisciplinary Optimization*, 17:1–13, 1999.

[2] G. Jekabsons. ARESLab: Adaptive Regression Splines toolbox for Matlab. *available at http://www.cs.rtu.lv/jekabsons/*, 2010.

[3] J. Müller and C.A. Shoemaker. Influence of ensemble surrogate models and sampling strategy on the solution quality of algorithms for computationally expensive black-box global optimization problems. *Journal of Global Optimization*, 2014.

[4] R.G. Regis and C.A. Shoemaker. A stochastic radial basis function method for the global optimization of expensive functions. *INFORMS Journal on Computing*, 19:497–509, 2007.