User guide for Modularized Surrogate Model Toolbox

Juliane Müller

Tampere University of Technology
Department of Mathematics

email: juliane.mueller2901@gmail.com

October 2, 2012

1 Introduction

This user guide accompanies the surrogate model toolbox for global optimization problems. The toolbox is made for computationally expensive black-box global optimization problems with continuous, integer, or mixed-integer variables. Problems where several or all variables have to be integers may also have black-box constraints, whereas purely continuous problems may only have box constraints.

For problems with computationally cheap function evaluations the toolbox may not be very efficient. Surrogate models are intended to be used when function evaluations take from several minutes to several hours or more. When reading this manual it is recommended to simultaneously take a look at the code.

The code is set up such that the user only has to define his/her optimization problem in a Matlab file (see Section 6.1). Additional input such as the surrogate model to be used, the sampling strategy, or starting points are optional (see Section 6).

This document is structured as follows. In Section 2 the general structure of a surrogate model algorithm is summarized. The installation is described in Section 3. The dependencies of the single functions in the code are shown in Section 4. Section 5 briefly summarizes how the surrogate model algorithm works in general. Section 6 describes the options for the input of the main function. In Section 7 the input and output of the single subfunctions of the algorithm are described. Examples for using the surrogate model algorithm are given in Section 8. In Section 9 it is explained how the user can define an own (mixture) surrogate model and an example is given. The elements of the saved results are described in Section 10.

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

2 Surrogate Model Algorithms

Surrogate models (also known as response surfaces, metamodels) are used during the optimization phase to approximate expensive simulation models [1]. During the optimization phase information from the surrogate model is used in order to guide the search for improved solutions. Using the surrogate model instead of the true simulation model reduces the computation time considerably. Most surrogate model algorithms consist of the same steps as shown in the algorithm below.

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 response surface to the data generated in Steps 1 and 2.
4. Use the response surface to predict the objective function values at unsampled points in the variable domain to decide at which point to do the next expensive function evaluation.
5. Do the expensive function evaluation at the point selected in Step 4.
6. Use the new data point to update the surrogate model.
7. Iterate through Steps 4 to 6 until the stopping criterion has been met.

Surrogate model algorithms in the literature differ mainly with respect to

- the generation of 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, a maximum allowed CPU time, or a maximum number of failed iterative improvement trials.

3 Installation

Download the file SurrogateOptimizationModule.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

File → Set Path... → Add with Subfolders → Save

You can try if the algorithm works by typing

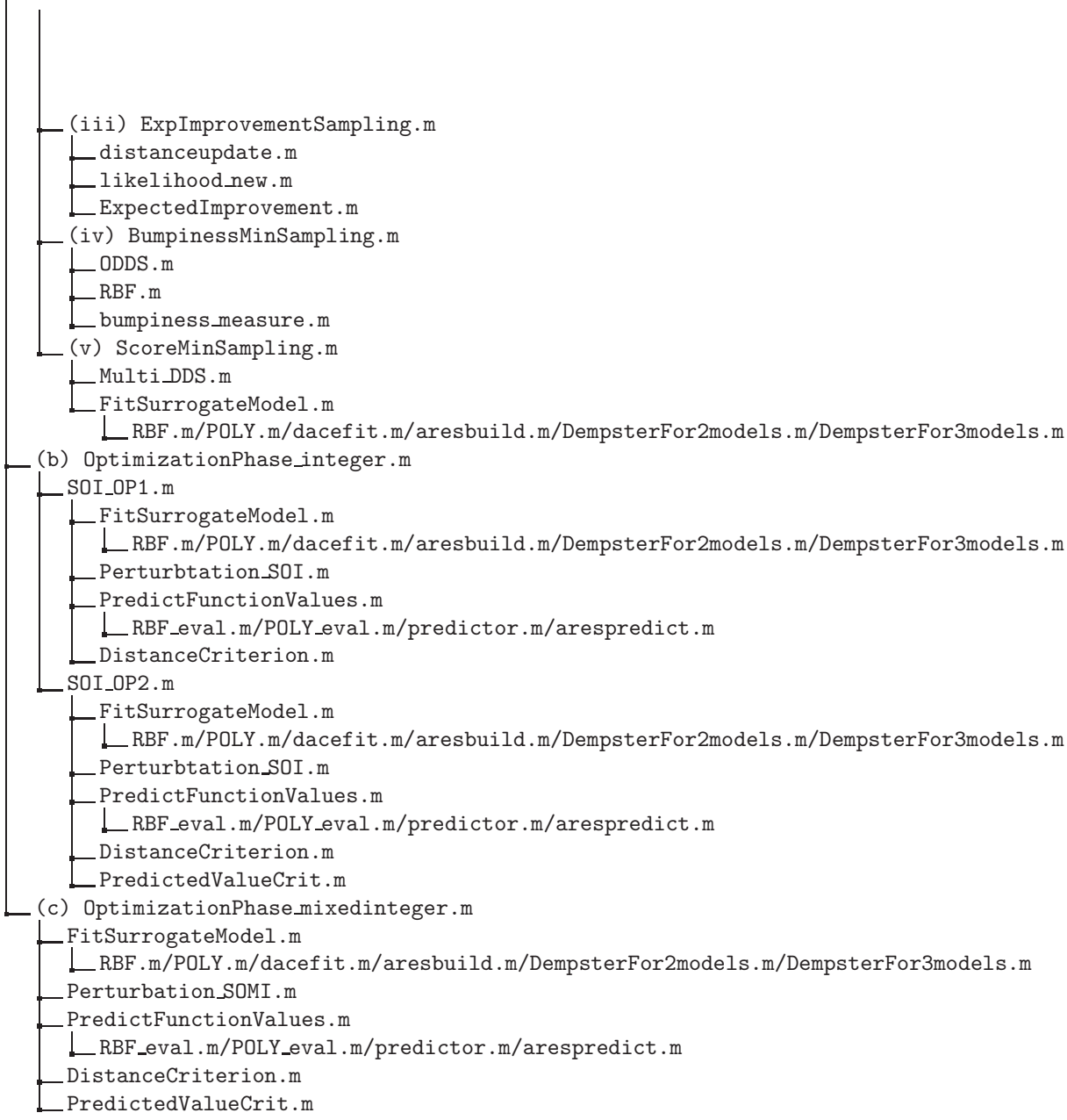
```
>> SurrogateModelModule_v1('datainput_hartman6',300,'MIX_RcKg', 'CAND', 'SLHD', 15);
```

into the command prompt.

4 Code Structure

The structure of the code is outlined here. Depending on if there are integrality constraints in the problem either option (a), (b), or (c) is used. If the problem has only continuous variables, the user can choose different sampling strategies (options (i)-(v)). For problems with integer constraints only the candidate point sampling strategy can be used at this moment.

```
SurrogateModelModule_v1.m
├─ StartingDesign.m
│  └─ SLHD.m/lhsdesign.m/bestlh.m/cornerpoints.m
└─ (a) OptimizationPhase_continuous.m
   ├─ FitSurrogateModel.m
   │  └─ RBF.m/POLY.m/dacefit.m/aresbuild.m/DempsterFor2models.m/DempsterFor3models.m
   ├─ (i) CandidatePointSampling.m
   │  ├─ Perturbation.m
   │  ├─ PredictFunctionValues.m
   │  │  └─ RBF_eval.m/POLY_eval.m/predictor.m/arespredict.m
   │  ├─ DistanceCriterion.m
   │  └─ PredictedValueCrit.m
   └─ (ii) SurfaceMinSampling.m
      ├─ ODDS.m
      └─ FitSurrogateModel.m
         └─ RBF.m/POLY.m/dacefit.m/aresbuild.m/DempsterFor2models.m/DempsterFor3models.m
```



5 The Main File SurrogateModelModule_v1.m

The file from which to run the surrogate model toolbox is SurrogateModelModule_v1.m. The file expects several inputs (see Section 6), of which only the first is mandatory. The algorithm starts by checking the input and assigns default values to variables that have not been set by the user. An initial experimental design is created (StartingDesign.m). After evaluating the expensive function evaluations at the selected points in the initial experimental design one of the three (continuous, integer, mixed-integer) optimization routines is called.

6 Input

The main file SurrogateModelModule_v1.m requires several input arguments (see Table 1), out of which only the first is mandatory to run the algorithm. If no input is given for the remaining arguments, default values are used.

Table 1: Input parameters

Input	Description
data_file	string with name of data file containing optimization problem data (mandatory!)
maxeval	integer defining maximum number of allowed function evaluations (default 400)
surrogate_model	string defining which surrogate model to use (default 'RBFcub')
sampling_technique	string defining the technique for picking the next sample site (default 'CAND')
initial_design	string defining which initial experimental design should be used (default 'SLHD')
number_startpoints	integer defining the number of initial starting points (default $2(d+1)$, d=dimension)
starting_point	matrix with points added to initial experimental design

6.1 Input data_file

The data file contains all the necessary problem information. See for example the file datainput_hartman6.m. The data file has no input argument, and one output argument (the structure Data). The data structure has to contain the information shown on Table 2.

Table 2: Contents data file

Variable	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, 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
Data.constraint	cell array with handles to constraint functions; only for integer and mixed-integer problems; optional

6.2 Input surrogate_model

There are several options for choosing the surrogate model. Surrogate models can be interpolating (for example radial basis functions (RBF), kriging) or non-interpolating (polynomial regression models, multivariate adaptive regression splines (MARS)). The toolbox allows the user to choose between different (mixture) models (see Tables 3, 4, 5, 6). The surrogate model to be used can be defined by the user as string as function input surrogate_model. Numerical experiments showed that if nothing about the behavior of the black-box objective function is known, a mixture surrogate model containing a radial basis function should be used. Mixture models prevent accidentally selecting the worst surrogate model.

Table 3: Radial basis function surrogate models

surrogate_model	Description
'RBFcub'	cubic radial basis function interpolant
'RBFtps'	thin plate spline radial basis function interpolant
'RBFlin'	linear radial basis function interpolant

Table 4: Kriging surrogate models

surrogate_model	Description
'KRIGexp0'	Kriging model with exponential correlation function and 0-order regression polynomial
'KRIGexp1'	Kriging model with exponential correlation function and first order regression polynomial
'KRIGexp2'	Kriging model with exponential correlation function and second order regression polynomial
'KRIGgexp0'	Kriging model with generalized exponential correlation function and 0-order regression polynomial
'KRIGgexp1'	Kriging model with generalized exponential correlation function and first order regression polynomial
'KRIGgexp2'	Kriging model with generalized exponential correlation function and second order regression polynomial
'KRIGgauss0'	Kriging model with Gaussian correlation function and 0-order regression polynomial
'KRIGgauss1'	Kriging model with Gaussian correlation function and first order regression polynomial
'KRIGgauss2'	Kriging model with Gaussian correlation function and second order regression polynomial
'KRIGlin0'	Kriging model with linear correlation function and 0-order regression polynomial
'KRIGlin1'	Kriging model with linear correlation function and first order regression polynomial
'KRIGlin2'	Kriging model with linear correlation function and second order regression polynomial
'KRIGspline0'	Kriging model with spline correlation function and 0-order regression polynomial
'KRIGspline1'	Kriging model with spline correlation function and first order regression polynomial
'KRIGspline2'	Kriging model with spline correlation function and second order regression polynomial
'KRIGsphere0'	Kriging model with spherical correlation function and 0-order regression polynomial
'KRIGsphere1'	Kriging model with spherical correlation function and first order regression polynomial
'KRIGsphere2'	Kriging model with spherical correlation function and second order regression polynomial
'KRIGcub0'	Kriging model with cubic correlation function and 0-order regression polynomial
'KRIGcub1'	Kriging model with cubic correlation function and first order regression polynomial
'KRIGcub2'	Kriging model with cubic correlation function and second order regression polynomial

Table 5: Non-interpolating surrogate models

surrogate_model	Description
'POLYlin'	linear regression polynomial
'POLYquad'	quadratic regression polynomial
'POLYquadr'	reduced quadratic regression polynomial
'POLYcub'	cubic regression polynomial
'POLYcubr'	reduced cubic regression polynomial
'MARS'	multivariate adaptive regression spline

Table 6: Mixture surrogate models

surrogate_model	Description
'MIX_RcKg'	mixture of cubic radial basis function interpolant and kriging model with Gaussian correlation function (first order regression polynomial)
'MIX_RcM'	mixture of cubic radial basis function interpolant and multivariate adaptive regression spline
'MIX_RcPc'	mixture of cubic radial basis function interpolant and full cubic regression polynomial
'MIX_KgM'	mixture of kriging model with Gaussian correlation function (first order regression polynomial) and multivariate adaptive regression spline
'MIX_KgPc'	mixture of kriging model with Gaussian correlation function (first order regression polynomial) and cubic regression polynomial
'MIX_KgPqr'	mixture of kriging model with Gaussian correlation function (first order regression polynomial) and reduced quadratic regression polynomial
'MIX_KgPcr'	mixture of kriging model with Gaussian correlation function (first order regression polynomial) and reduced cubic regression polynomial
'MIX_RcKgM'	mixture of cubic radial basis function interpolant, kriging model with Gaussian correlation function (first order regression polynomial) and multivariate adaptive regression spline

6.3 Input SampleStrategy

For problems with only continuous variables the user can determine the strategy for selecting the sample point in each iteration (see Table 7, input sampling_technique).

In the candidate point approach [10] (CAND) two groups of candidates for the next sample point are created. The points in the first group are generated by perturbing the best point found so far. The points in the second group are generated by uniformly selecting points from the whole box-constrained variable domain. The response surface is used to predict the objective function values at the candidate points (response surface criterion). The distance of each candidate point to the set of already sampled points is computed (distance criterion), and a weighted sum of both criteria is used to determine the best candidate point.

When using the minimum of the response surface ('SURFmin') as sampling strategy, the dynamically dimensioned search (DDS) algorithm [12] is used to find the surface minimum. Other algorithms (such as Matlab's fmincon) could be used as well. Note that it is not necessary to find the global minimum of the surface. Rather, response surfaces that are able to model multimodalities are preferable because sample points are likely to be derived from local optima in different regions of the variable domain, which in turn makes the search more global (see also [11]).

Using the point that maximizes the expected improvement ('EImax') has been introduced by Jones et al. [6] (EGO algorithm). When using this sampling strategy, the kriging model must be used

as response surface because it returns an error estimate together with the objective function value prediction. Thus, if the maximum expected improvement is the sampling strategy of choice, the surrogate model cannot be set by the user.

The strategy of using the point that minimizes the scoring criterion ('SCOREmin') corresponds to finding the point that minimizes the weighted score defined by the response surface and the distance criterion. In contrast to the candidate point approach (where the best candidate from a limited number of points is selected) the actual minimum is tried to be found. When minimizing the scoring function the DDS algorithm is used.

Minimizing the bumpiness measure [3] ('BUMPmin') aims at finding the point in the variable domain where it seems most "reasonable" that the objective function takes on a given target value, i.e. it is more reasonable that the objective function takes on a very low value in regions of the variable domain where already low function values have been encountered, rather than in regions where large objective function values have been observed and very steep valleys would result. The sampling strategy can only be used in connection with a radial basis function interpolant. During the optimization phase the algorithm cycles through several different target values as described by Holmström [4].

Computational experiments showed that the candidate point approach ('CAND'), maximizing the expected improvement ('EImax'), or using the minimum of the response surface ('SURFmin') lead in general to the best results.

Table 7: Options for sampling techniques

sampling_technique	Description
'CAND'	candidate point approach (default)
'SURFmin'	uses the minimum point of response surface
'EImax'	uses the point that maximizes the expected improvement (currently working only for kriging with generalized exponential correlation function)
'SCOREmin'	uses the scoring criteria from the candidate point approach, but tries to find the best point in the whole domain with respect to these criteria instead of choosing the best point among a limited number of points as done in 'CAND')
'BUMPmin'	minimizes bumpiness measure (see [3]), only for cubic, linear and thin-plate spline radial basis function interpolant

6.4 Input initial_design

The user can choose between the initial experimental design strategies shown in Table 8.

Table 8: Initial experimental design strategies

sampling_technique	Description
'SLHD'	symmetric Latin hypercube design
'LHS'	Matlab's built-in Latin hypercube design (default)
'CORNER'	uses (subset) of corner points of variable domain
'SPACEFIL'	space filling design (as in EGO, see [2])

When choosing the SLHD option, it is advised to use at least $2d$ (d=problem dimension) starting points, due to the possibility of linear dependencies in the design matrix. When many points are required for the initial experimental design, using the 'SPACEFIL' strategy becomes computationally

more expensive because of the optimization routine when creating the design. Also note that if more points are required for the initial experimental than there are corner points, the corner point sampling strategy as initial experimental design fails.

6.5 Inputs number_startpoints and starting_point

The user can define the number of points s/he wishes to have in the initial experimental design. The minimum number of required points depends on the surrogate model and initial design strategy.

- When using the symmetric Latin hypercube sampling (SLHD), at least $2d$ (d=dimension) points should be used due to the possibility of linear dependencies in the design matrix.
- When using the kriging model together with the second order regression polynomial (KRIGexp2, KRIGgexp2, KRIGgauss2, KRIGlin2, KRIGspline2, KRIGsphere2, KRIGcub2), at least $1 + 2d + \binom{d}{2}$ points are needed, otherwise the DACE toolbox fails.
- 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 mixture models in general at least the minimum number of required points plus one additional point are needed because of the leave-one-out cross-validation. For example, if using MIX_RcKg for a d-dimensional problem, at least $d + 2$ points are required (first order regression polynomial needs at least $d + 1$ points, and one additional point is needed because of the leave-one-out cross-validation).
- When using the mixture 'MIX_RcPc' or 'MIX_KgPc', at least $2+3d+\binom{d}{2}+\binom{d}{3}$ points are needed. Otherwise the least squares problem is underdetermined.
- When using the mixture 'MIX_KgPqr' at least $2 + 2d$ points are needed. Otherwise the least squares problem is underdetermined.
- When using the mixture 'MIX_KgPcr' or 'MIX_KgPc', at least $2+3d$ points are needed. Otherwise the least squares problem is underdetermined.

In numerical experiments it was found that $2(d + 1)$ points (which is twice the minimum number of points needed to fit an RBF model of dimension d) work in general well. If the total number of allowed function evaluations is very low, it might however be better to use fewer points in the starting design, and spend more evaluations in the iterative improvement. If number_startpoints is not given, the default value $2(d + 1)$ is used.

Additionally, the user can define m point(s) s/he wants to add to the initial experimental design. The points must be given in matrix form ($m \times d$), and the points are added at the beginning of the starting design.

6.6 Input Example

The following example calls the surrogate model toolbox for finding the minimum of the six-dimensional Hartmann function defined in the file datainput_hartman6.m. The maximum number of function evaluations is set to 300, the surrogate model to be used is a mixture of the cubic radial basis function interpolant and the kriging model with Gaussian correlation function ('MIX_RcKg'). The candidate point sampling strategy is used ('CAND'), and the initial experimental design is built with a symmetric Latin hypercube design ('SLHD') with 15 points. Starting points are not given.

The user is encouraged to try out the example by typing into the Matlab command window (make sure the location of the files is known to Matlab's search path):

```
>> SurrogateModelModule_v1('datainput_hartman6',300,'MIX_RcKg', 'CAND', 'SLHD', 15);
```

7 File Description

This section contains the input and output variables of the single functions.

7.1 SurrogateModelModule_v1.m

Input: See Table 1.

Output: None. The algorithm saves the results in the structure Data in the file Results.mat.

This is the main file from which the algorithm is started. At first the user input is checked for correctness. If all input is correct, the initial experimental design is generated (file StartingDesign.m). If there are integrality constraints for the variables, the corresponding values of the variables in the initial design are rounded to the closest integers. The computationally expensive function evaluations are done at the points in the initial experimental design[1]. Depending on whether there are only continuous, only integer, or mixed-integer variables, the corresponding optimization function is called (OptimizationPhase_continuous.m, OptimizationPhase_integer.m, OptimizationPhase_mixedinteger.m).

7.2 StartingDesign.m

Input

Variable	Description
initial_design	string, name of design strategy (see Table 8)
number_startpoints	integer, number of points desired for the initial experimental design
Data	structure, contains all problem information

Output: Matrix with points in initial experimental design:

$$S = \begin{bmatrix} \mathbf{x}_1^T \\ \mathbf{x}_2^T \\ \vdots \\ \mathbf{x}_m^T \end{bmatrix} = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1d} \\ x_{21} & x_{22} & \cdots & x_{2d} \\ \vdots & \vdots & & \vdots \\ x_{m1} & x_{m2} & \cdots & x_{md} \end{bmatrix},$$

where x_{ij} is the jth component of the ith sample point. StartingDesign.m calls either one of the functions

Function	Description
lhsdesign.m	Matlab's built-in latin hypercube design
SLHD.m	symmetric Latin hypercube design [13]
bestlh.m	space-filling design, implementation by [2]
cornerpoints.m	corner point strategy

[1] Note that this may take a while depending on the time required for one function evaluation. Alternatively, the evaluations can be done in parallel. Uncomment the parfor loop in that case as indicated in the code.

7.3 OptimizationPhase_continuous.m

This function is executed when the optimization problem has only continuous variables.

Input:

Variable	Description
Data	structure array, contains the optimization problem information
Surrogate	string, determines which surrogate model to be used
SampleStrategy	string, determines strategy for selecting the next sample site
maxeval	integer, maximum number of allowed function evaluations

Output: updated structure array Data containing all information about the problem and solution

The function starts by assigning the parameter tolerance, which is used to decide when the distance of two points is so low that they are considered equal. The value can be changed by the user as desired. Note however that ill-conditioning of the design matrix may become a problem if the value is reduced. The best point and the corresponding function value in the initial experimental design are determined and stored in the variables Data.xbest and Data.fbest, respectively. These values are being updated throughout the optimization. Large function values are replaced by the median of all function values obtained so far in order to prevent the response surface from oscillating wildly.

The desired surrogate model is fit to the data (Data.S and Data.Ymed). For fitting the kriging models the Matlab toolbox DACE [7] has been used, and the user is referred to the DACE manual for instructions of use and parameter settings. Note that due to ill-conditioning of the sample site matrix the DACE toolbox may fail, in which case the algorithm fails. The DACE toolbox issues an error and the user may want to use a different surrogate model. For fitting the multivariate adaptive regression spline (MARS), the Matlab toolbox ARESLab [5] is used. The user is referred to the ARESlab manual for further details. If mixture models are to be used, several surrogate models have to be fit to the data. Dempster-Shafer theory is used to determine the influence each model should have in the mixture [8]. The user may define other (mixture) surrogate models where indicated in the code.

The surrogate models for which implementations exist are summarized in Tables 3, 4, 5, and 6, respectively.

7.3.1 FitSurrogateModel.m

This function fits the response surface(s) to the given data.

Input:

Variable	Description
Data	structure, contains all information about the problem
Surrogate	string, containing surrogate model to be used

Output:

Variable	Description
lambda, gamma	vectors, parameters of RBF model; lambda=gamma=[] if RBF model not used
dmodel	structure, parameters for kriging model; dmodel=[] if kriging model not used
mmodel	structure, parameters for MARS model; mmodel=[] if MARS model not used
beta	vector, parameters of polynomial regression model; beta=[] if polynomial model not used
w_m	vector, contains the weights for the models in mixtures; w_m=[] if no mixture model used

RBF.m *and* RBF_eval.m
The parameters of the radial basis function (RBF) interpolant are computed by the function RBF.m, and RBF_eval.m predicts the objective function values using the RBF surrogate model.

Input RBF.m:

Variable	Description
S	matrix, contains the sample points
Y	column vector, contains function values corresponding to points in S
flag	string, determines what type of RBF interpolant is used (see Table 3)

Output RBF.m:

Variable	Description
lambda	column vector, contains multipliers of radial basis functions
gamma	column vector, contains parameters for polynomial tail

Input RBF_eval.m:

Variable	Description
X	matrix, contains points where objective function value will be predicted
S	matrix, contains the sample points already evaluated
lambda	column vector, contains multipliers of radial basis functions
gamma	column vector, contains parameters for polynomial tail
flag	string, determines what type of RBF interpolant is used (see Table 3)

Output RBF_eval.m: vector Yest containing predicted objective function values

POLY.m ***and*** **POLY_eval.m**
The parameters of the polynomial regression model are computed by the function POLY.m, and POLY_eval.m predicts the objective function values.

Input POLY.m:

Variable	Description
S	matrix, contains the sample points
Y	column vector, contains function values corresponding to points in S
flag	string, determines the order of the polynomial (lin, quad, quadr, cub, cubr)

Output POLY.m: vector b containing the parameters of the polynomial regression model

Input POLY_eval.m:

Variable	Description
X	matrix, contains points where objective function value will be predicted
b	column vector, contains parameters
flag	string, determines the order of the polynomial (lin, quad, quadr, cub, cubr)

Output POLY_eval.m: vector Yest containing predicted objective function values

dacefit.m ***and*** **predictor.m**
See the tutorial for the Matlab toolbox DACE [7] for input and output arguments, and parameter settings. The output is stored in the structure variable dmodel.

aresbuild.m ***and*** **arespredict.m**
See the tutorial for the Matlab toolbox ARESLab [5] for input and output arguments, and parameter settings. The output is stored in the structure variable mmodel.

7.3.2 CandidatePointSampling.m

The function is called when the candidate point sampling strategy (setting 'CAND') is used. The function creates candidates for the next sample site by perturbing the best point found so far, and by uniformly selecting points from the whole variable domain (function Perturbation.m). The objective function values of the candidates are predicted using the response surface, and the distances of the candidate points to the set of already sampled points are computed. Based on these two criteria the best candidate point is selected for doing the next objective function evaluation.

Input:

Variable	Description
Data	structure, contains all information about the optimization problem
maxeval	integer, maximum number of allowed function evaluations
Surrogate	string, surrogate model type to be used
lambda,gamma	vectors, parameters of RBF model; lambda=gamma=[] if RBF model not used
dmodel	structure, parameters for kriging model; dmodel=[] if kriging model not used
mmodel	structure, parameters for MARS model; mmodel=[] if MARS model not used
beta	vector, parameters of polynomial regression model; beta=[] if polynomial model not used
w_m	vector, contains the weights for the models in mixtures; w_m=[] if no mixture model used
tolerance	scalar, distance when two points are considered equal

Output: updated structure Data with all information about the problem.

Perturbation.m
The function generates candidate points for the next sample site by randomly adding or subtracting perturbations to randomly chosen variables of the best point found so far. Furthermore, candidates that are uniformly selected from the whole variable domain are generated.

Input:

Variable	Description
Data	structure, contains all information about the problem
NCandPoint	integer, number of points for each candidate point group
sigma_stdev	vector, contains perturbation ranges (small, medium, large)
P	scalar, perturbation probability for each variable

Output: matrix CandPoint of size (2·NCandPoint$\times d$), where d is the problem dimension.

PredictFunctionValues.m
The function calls RBF_eval., POLY_eval., predictor.m, and/or arespredict.m, respectively for doing objective function value predictions at the candidate points.
Input:

Variable	Description
Data	structure, contains all information about the optimization problem
Surrogate	string, surrogate model type to be used
CandPoint	matrix, contains points for doing the objective function value predictions
lambda,gamma	vectors, parameters of RBF model; lambda=gamma=[] if RBF model not used
dmodel	structure, parameters for kriging model; dmodel=[] if kriging model not used
mmodel	structure, parameters for MARS model; mmodel=[] if MARS model not used
beta	vector, parameters of polynomial regression model; beta=[] if polynomial model not used
w_m	vector, contains the weights for the models in mixtures; w_m=[] if no mixture model used

Output: vector CandValue with predicted objective function value for each point in CandPoint.

Distancecriterion.m
The function computes the distance of each candidate point to the set of already sampled points, and scales the values to the interval [0,1], where the largest distance is scaled to 0 and the smallest distance is scaled to 1.

Input:

Variable	Description
S	matrix, contains all already sampled points
CandPoint	matrix, contains all candidate points for next function evaluation

Output:

Variable	Description
ScaledDist	vector, contains distances of candidate points to the set of already sampled points, scaled to [0,1]
Dist	vector, contains distances of candidate points to the set of already sampled points

PredictedValueCrit.m
Scales the predicted objective function values at the candidate points to the interval [0,1], where the lowest predicted objective function value is scaled to 0, and the largest predicted value is scaled to 1.

Input: Vector CandValue containing the predicted objective function values of all candidate points.

Output: Vector ScaledCandValue containing the predicted objective function values scaled to the interval [0,1].

7.3.3 SurfaceMinSampling.m

The function is called when the minimum site of the response surface is used as sampling strategy (setting 'SURFmin'). The dynamically dimensioned search algorithm [12] is used for finding the minimum of the surface.

Input:

Variable	Description
Data	structure, contains all information about the optimization problem
maxeval	integer, maximum number of allowed function evaluations
Surrogate	string, surrogate model type to be used
lambda,gamma	vectors, parameters of RBF model; lambda=gamma=[] if RBF model not used
dmodel	structure, parameters for kriging model; dmodel=[] if kriging model not used
mmodel	structure, parameters for MARS model; mmodel=[] if MARS model not used
beta	vector, parameters of polynomial regression model; beta=[] if polynomial model not used
w_m	vector, contains the weights for the models in mixtures; w_m=[] if no mixture model used
tolerance	scalar, distance when two points are considered equal

Output: updated structure Data with all information about the problem.

ODDS.m
Ordinary dynamically dimensioned search algorithm [12] for finding the minimum of the response surface.

Input:

Variable	Description
Data	structure, contains all information about the optimization problem
x0	vector, starting guess for search
objective	string, surrogate model type to be used
lambda,gamma	vectors, parameters of RBF model; lambda=gamma=[] if RBF model not used
dmodel	structure, parameters for kriging model; dmodel=[] if kriging model not used
mmodel	structure, parameters for MARS model; mmodel=[] if MARS model not used
beta	vector, parameters of polynomial regression model; beta=[] if polynomial model not used
w_m	vector, contains the weights for the models in mixtures; w_m=[] if no mixture model used
tolerance	scalar, distance when two points are considered equal

Output: vector xbest, best point found during the optimization routine.

7.3.4 ExpImprovementSampling.m

The function is called when the point that maximizes the expected improvement is used as sampling strategy (setting 'EImax'). Matlab's genetic algorithm is used to maximize the expected improvement. Parts of the implementation are from [2].

Input:

Variable	Description
Data	structure, contains all information about the optimization problem
maxeval	integer, maximum number of allowed function evaluations

Output: updated structure Data with all information about the problem.

distanceupdate.m
Computes the pairwise distances of all sample points.

Input: Structure Data containing all information about the problem

Output:

Variable	Description
D	matrix, contains pairwise distances between variable values in each dimension
ij	matrix, contains indices of points for which pairwise distances computed

likelihood_new.m
Implementation by A.I.J. Forrester [2]. See the book *Engineering Design via Surrogate Modelling - A Practical Guide* by [2] for usage and file description.

ExpectedImprovement.m
Computes the expected improvement at a given point n the variable domain.

Input:

Variable	Description
Data	structure, contains information about the problem
x	vector, the point in the variable domain at which the expected improvement is computed

Output: scalar ExpImp, the expected improvement at the point x. See the book *Engineering Design via Surrogate Modelling - A Practical Guide* by [2] for usage and file description.

7.3.5 BumpinessMinSampling.m

The function is called when the minimum site of the bumpiness measure is used as sampling strategy (setting 'BUMPmin').

Input:

Variable	Description
Data	structure, contains all information about the optimization problem
maxeval	integer, maximum number of allowed function evaluations
Surrogate	string, surrogate model type to be used
lambda,gamma	vectors, parameters of RBF model; lambda=gamma=[] if RBF model not used
tolerance	scalar, distance when two points are considered equal

Output: updated structures array with all information about the problem.

bumpiness_measure.m
The function is minimized in order to find the point where it is most reasonable that the objective function takes on the given target value.

Input:

Variable	Description
x	vector at which the bumpiness measure is being computed
Data	structure, contains all information about the optimization problem
flag	string, containing information of the type of RBF model to be used
target	scalar, target value for objective function
tolerance	scalar, distance when two points are considered equal
lambda,gamma	vectors, parameters of RBF model

Output: bumpiness measure value hn.

7.3.6 ScoreMinSampling.m

The function is called when the minimum site of the score function is used as sampling strategy (setting 'SCOREmin').

Input:

Variable	Description
Data	structure, contains all information about the optimization problem
maxeval	integer, maximum number of allowed function evaluations
Surrogate	string, surrogate model type to be used
lambda,gamma	vectors, parameters of RBF model; lambda=gamma=[] if RBF model not used
dmodel	structure, parameters for kriging model; dmodel=[] if kriging model not used
mmodel	structure, parameters for MARS model; mmodel=[] if MARS model not used
beta	vector, parameters of polynomial regression model; beta=[] if polynomial model not used
w_m	vector, contains the weights for the models in mixtures; w_m=[] if no mixture model used
tolerance	scalar, distance when two points are considered equal

Output: updated structure Data with all information about the problem.

Multi_DDS.m
Implementation of the DDS algorithm [12] for finding the minimum of the scoring function.

Input:

Variable	Description
Data	structure, contains all information about the optimization problem
valueweight	scalar, weight for the response surface criterion
mindistweight	scalar, weight for the distance criterion
tolerance	scalar, distance when two points are considered equal
myfun	function handle to the (mixture) surrogate model

Output: best point found during optimization xbest

7.4 OptimizationPhase_integer.m

This function is executed when the optimization problem has only integer variables. The algorithm is able to solve problems that have in addition to the integer and box constraints also black-box constraints. The constraints must be defined in form of function handles in the Data_file as variables Data.constraint{1}, Data.constraint{2}, etc. See the file datainput_G4_I.m for an example. If there are black-box constraints, and if there is no feasible point contained in the starting design, then the function SOI_OP1.m is executed. After a first feasible point has been found (or the problem does not have any black-box constraints), SOI_OP2.m is executed. If the problem has integer constraints for all variables, only the candidate point sampling strategy can be used.

7.4.1 SOI_OP1.m

This function tries to find a first feasible point of an optimization problem with additional black-box constraints by minimizing a constraint violation function. The candidate point approach is used during the optimization. SOI_OP1.m stops after the first feasible point has been found, or the maximum number of allowed function evaluations has been reached.

Input:

Variable	Description
Data	structure, contains all information about the optimization problem
maxeval	integer, maximum number of allowed function evaluations
P	scalar, perturbation probability for creating candidates
stdev_int	vector of integers, perturbation ranges
Surrogate	string, contains the name of the (mixture) surrogate model to be used for predictions

Output: Updated structure Data with all problem information.

Perturbation_SOI.m
This function generates the candidate points for the next expensive function evaluation. The best point found so far is perturbed for generating the candidates in the first group. For candidates in the second group are uniformly selected integer points form the whole variable domain.

Input:

Variable	Description
Data	structure, contains all information about the optimization problem
NCandidates	integer, number of candidates
stdev_int	vector of integers, perturbation ranges
P	scalar, perturbation probability for creating candidates

Output: Matrix CandPoint with candidate points.

7.4.2 SOI_OP2.m

This function is used after a first feasible point of an optimization problem with additional black-box constraints has been found (or if the problem does not have any additional constraints). The candidate point approach is used during the optimization.

Input:

Variable	Description
Data	structure, contains all information about the optimization problem
maxeval	integer, maximum number of allowed function evaluations
P	scalar, perturbation probability for creating candidates
stdev_int	vector of integers, perturbation ranges
Surrogate	string, contains the name of the (mixture) surrogate model to be used for predictions

Output: Updated structure Data with all problem information.

7.5 OptimizationPhase_mixedinteger.m

This function is called when mixed-integer black-box problems are considered. If the mixed-integer problem has additional black-box constraints, the user must supply at least one feasible point in the variable starting_point in the input of SurrogateModelModule_v1.m. For mixed-integer problems only the candidate point sampling approach can be used.

Input:

Variable	Description
Data	structure, contains all information about the optimization problem
Surrogate	string, contains the name of the (mixture) surrogate model to be used for predictions
maxeval	integer, maximum number of allowed function evaluations

Output: Updated structure Data with all problem information.

7.5.1 Perturbation_SOMI.m

This function is used to create candidate points by uniformly selecting points from the whole variable domain, and by perturbing the best feasible point found so far as follows

- randomly add or subtract small, medium, or large perturbations only to continuous variables
- randomly add or subtract small, medium, or large perturbations only to integer variables
- randomly add or subtract small, medium, or large perturbations to continuous and integer variables

Input:

Variable	Description
Data	structure, contains all information about the optimization problem
NCandidates	integer, number of candidates
stdev_int	vector of integers, perturbation ranges for integer variables
stdev_cont	vector, perturbation ranges for continuous variables
P	scalar, perturbation probability for creating candidates

Output: Matrix CandPoint with candidate points.

8 Examples

This section contains several examples for continuos, integer, and mixed-integer black-box optimization problems. The input data files are supplied, and the user is encouraged to take a look at these files when defining data files for own problems. The examples have computationally cheap objective functions in order to reduce the computation time when experimenting with the code.

8.1 Examples for Continuous Problems

The algorithm is applicable to box-constrained continuous problems. If the problem has additional constraints, they should be incorporated with a penalty term in the objective function. The input data files below are all for continuous problems.

- datainput_Ackley15.m
- datainput_Ackley30.m
- datainput_Branin.m
- datainput_DixonPrice15.m
- datainput_hartman3.m
- datainput_hartman6.m
- datainput_Levy20.m
- datainput_Michalewicz25.m
- datainput_Powell24.m
- datainput_Rastrigin12.m
- datainput_Rastrigin30.m
- datainput_Rosenbrock10.m
- datainput_Rosenbrock20.m
- datainput_Schoen_10_4_3.m
- datainput_Schoen_17_2_5.m
- datainput_Shekel5.m
- datainput_Shekel7.m
- datainput_Shekel10.m
- datainput_Shubert.m

- datainput_Sphere27.m
- datainput_Zakharov.m

For the six-dimensional Hartmann function, type

```
>> SurrogateModelModule_v1('datainput_hartman6',200,'RBFtps','BUMPmin','LHS',7);
```

With this input the following settings are used:

- 200 function evaluations are allowed
- the thin-plat spline radial basis function model is used as response surface (RBFtps)
- the bumpiness measure (BUMPmin) is used as sampling strategy
- the Latin hypercube sampling strategy (Matlab's built-in Latin hypercube sampling) is used (LHS) as experimental design strategy
- 7 points are used in the initial starting design

For the 4-dimensional Schoen function with 7 local minima, type

```
>> SurrogateModelModule_v1('datainput_Shekel7',400,'MIX_RcPc','CAND','SLHD',24);
```

With this input the following settings are used:

- 400 function evaluations are allowed
- the mixture model of cubic RBF and full cubic polynomial model is used as response surface (MIX_RcPc)
- the candidate point sampling strategy (CAND) is used as sampling strategy
- the symmetric Latin hypercube sampling strategy (SLHD) is used as experimental design strategy
- 24 points are used in the initial starting design

8.2 Examples for Integer Problems

The algorithm is applicable to optimization problems where all variables have integer constraints, and may have additional black-box constraints. The black-box constraints are treated with a penalty approach. Currently only the candidate point sampling strategy can be used with purely integer problems. The example data input files for integer problems all end in _I.m:

- datainput_convex_I.m
- datainput_ex_I.m
- datainput_ex1221_I.m
- datainput_G1_I.m
- datainput_G2_I.m
- datainput_G4_I.m
- datainput_G6_I.m
- datainput_G9_I.m
- datainput_GWSS20_I.m
- datainput_hmittelmann_I.m
- datainput_hydropower_1plant1_I.m

- datainput_hydropower_1plant2_I.m
- datainput_hydropower_1plant3_I.m
- datainput_hydropower_2plant1_I.m
- datainput_hydropower_2plant2_I.m
- datainput_hydropower_2plant3_I.m
- datainput_linearproblem_I.m
- datainput_nvs09_I.m
- datainput_nvs09alt_I.m
- datainput_Rastrigin12_I.m
- datainput_Rastrigin30_I.m
- datainput_throughput_I.m
- datainput_throughput_small_I.m

For the convex problem, type for example

```
>> SurrogateModelModule_v1('datainput_convex_I',300,'MARS','CAND','LHS',10);
```

With this input the following settings are used:

- 300 function evaluations are allowed
- the MARS model is used as response surface (MARS)
- the candidate point sampling strategy (CAND) is used as sampling strategy (this is the only sampling strategy that can be used with integer problems)
- Matlab's built-in Latin hypercube sampling strategy (LHS) is used as experimental design strategy
- 10 points are used in the initial starting design

For the small throughput problem, type for example

```
>> SurrogateModelModule_v1('datainput_throughput_small_I',500,'KRIGgauss2','CAND','LHS',21);
```

With this input the following settings are used:

- 500 function evaluations are allowed
- the kriging model with second order regression polynomial and Gaussian correlation is used as response surface (KRIGgauss2)
- the candidate point sampling strategy (CAND) is used as sampling strategy (this is the only sampling strategy that can be used with integer problems)
- Matlab's built-in Latin hypercube sampling strategy (LHS) is used as experimental design strategy
- 21 points are used in the initial starting design

8.3 Examples for Mixed-Integer Problems

The algorithm can be applied to mixed-integer problems that may have black-box constraints in addition to the variable upper and lower bounds. For constrained problems the user must supply at least one feasible starting point. During the computational experiments described in [9] the starting points given in the files StartPoints_1.mat, StartPoints_2.mat, etc. in Table 9 have been used. Each file contains 30 feasible points, and for each algorithm trial one of these points has been used. The data input files for mixed integer problems all end in _MI.m. For mixed-integer problem currently only the candidate point sampling strategy works. For testing the surrogate model algorithm for test problem datainput_G4_MI.m, for example, type

```
>> load('StartPoints_7.mat');
>> x0=StartPoints(1,:);
>> SurrogateModelModule_v1('datainput_G4_MI', [],[],[],[],[],x0)
```

In this example only the problem defining data file is given and the starting point. For all other input parameters the default values are used.

Table 9: Problem files and feasible starting points for mixed-integer problems

Test Problem File	Feasible Starting Points
datainput_BermanAshrafi_MI.m	StartPoints_1.mat
datainput_convex_MI.m	StartPoints_2.mat
datainput_ex_MI.m	StartPoints_13.mat
datainput_ex1221_MI.m	StartPoints_3.mat
datainput_Floudas_MI.m	StartPoints_4.mat
datainput_G2_MI.m	StartPoints_6.mat
datainput_G4_MI.m	StartPoints_7.mat
datainput_G6_MI.m	StartPoints_8.mat
datainput_G9_MI.m	StartPoints_9.mat
datainput_linearproblem_MI.m	StartPoints_11.mat
datainput_nvs09_MI.m	StartPoints_14.mat
datainput_nvs09alt_MI.m	StartPoints_15.mat
datainput_Rastrigin12_MI.m	StartPoints_16.mat
datainput_Rastrigin12alt_MI.m	StartPoints_12.mat
datainput_Rastrigin30_MI.m	StartPoints_10.mat
datainput_Yuan_MI.m	StartPoints_5.mat

For the problem G2, type for example

```
>> load('StartPoints_6.mat');
>> x0=StartPoints(11,:);
>> SurrogateModelModule_v1('datainput_G2_MI',300,'RBFtps','CAND','SLHD',50,x0);
```

With this input the following settings are used:

- 300 function evaluations are allowed
- the radial basis function with thin-plate spline is used as response surface (RBFtps)
- the candidate point sampling strategy (CAND) is used as sampling strategy (this is the only sampling strategy that can be used with mixed-integer problems)
- the symmetric Latin hypercube sampling strategy (SLHD) is used as experimental design strategy
- 50 points are used in the initial starting design

For the Yuan problem, type for example

```
>> load('StartPoints_5.mat');
>> x0=StartPoints(19,:);
>> SurrogateModelModule_v1('datainput_Yuan_MI',300,'MIX_RcKgM','CAND','SLHD',22,x0);
```

With this input the following settings are used:

- 300 function evaluations are allowed
- the mixture of cubic radial basis function, kriging with gaussian correlation function, and MARS is used as response surface (MIX_RcKgM)
- the candidate point sampling strategy (CAND) is used as sampling strategy (this is the only sampling strategy that can be used with mixed-integer problems)
- the symmetric Latin hypercube sampling strategy (SLHD) is used as experimental design strategy
- 22 points are used in the initial starting design

9 So you want to define your own surrogate model?

The files that need to be altered when defining an own (mixture) surrogate model are

- SurrogateModelModule_v1.m
- FitSurrogateModel.m
- DempsterFor2models.m or DempsterFor3models.m
- PredictFunctionValues.m
- SurfaceMinSampling.m (if necessary)
- ScoreMinSampling.m (if necessary)

9.1 Example

Suppose you want to add a mixture model consisting of the thin-plate spline RBF model and a reduced quadratic polynomial model. Let's abbreviate the model by MIX_RBFtpsPqr. We have to alter the file SurrogateModelModule_v1.m at the indicated position (see the code) as follows:

```
if isempty(surrogate_model)
      display('Using default surrogate model.')
      surrogate_model='RBFcub'; %if no surrogate model defined, use cubic RBF
else %add own surrogate model to the list
     if ~any(strcmp(surrogate_model,{'RBFcub', 'RBFtps', 'RBFlin','KRIGexp0',...
        'KRIGexp1','KRIGexp2','KRIGgexp0', 'KRIGgexp1', 'KRIGgexp2',...
        'KRIGgauss0', 'KRIGgauss1', 'KRIGgauss2','KRIGlin0','KRIGlin1',...
        'KRIGlin2','KRIGspline0','KRIGspline1','KRIGspline2','KRIGsphere0',...
        'KRIGsphere1','KRIGsphere2','KRIGcub0','KRIGcub1','KRIGcub2',...
        'POLYlin', 'POLYquad', 'POLYquadr', 'POLYcub', 'POLYcubr',...
        'MARS', 'MIX_RcKg','MIX_RcM','MIX_RcPc', 'MIX_KgM',...
        'MIX_KgPc', 'MIX_KgPqr', 'MIX_KgPcr','MIX_RcKgM','MIX_RBFtpsPqr'}))
          error('The surrogate model you want to use is not contained in the toolbox. Check spelling.')
      end
end
```

In the file FitSurrogateModel.m the following branch in the if-elseif tree must be added under the comment

% add more 2-model combinations here if necessary. Update function "DempsterFor2models" accordingly:

```
elseif strcmp(Surrogate,'MIX_RBFtpsPqr')
    [lambda, gamma]=RBF(Data.S,Data.Ymed,'TPS'); % RBF-tps
    beta=POLY(Data.S,Data.Ymed,'quadr'); %reduced quadratic polynomial
    w_m=DempsterFor2models(Data, Surrogate); %uses dempster shafer theory to adjust model weights
```

In the file DempsterFor2models.m the following branch in the if-elseif tree must be added under the comment

%other 2-model mixtures here:

```
%Mixture model: thin-plate spline RBF & reduced quadratic polynomial
elseif strcmp(Surrogate,'MIX_RBFtpsPqr')
    if ~forcekfold %leave-one-out cross-validation
        for jj=1:m
            [lambda, gamma]=RBF([Data.S(1:jj-1,:);Data.S(jj+1:end,:)],...
              [Data.Ymed(1:jj-1,:);Data.Ymed(jj+1:end,:)],'TPS'); % RBF
            yPred_all(jj,1)=RBF_eval(Data.S(jj,:),[Data.S(1:jj-1,:);...
              Data.S(jj+1:end,:)],lambda,gamma,'TPS'); %repredict jj-th objective function value
            reduced quadratic polynomial model prediction
            beta=POLY([Data.S(1:jj-1,:);Data.S(jj+1:end,:)],...
              [Data.Ymed(1:jj-1,:);Data.Ymed(jj+1:end,:)],'quadr');
            yPred_all(jj,2)=POLY_eval(Data.S(jj,:),beta,'quadr');
        end
    else %k-fold cross-validation
        Ss=Data.S(mix,:); %resorted sample sites
        Ys=Data.Ymed(mix);
        for jj=1:ceil(m/kfold)
            %validation set
            if jj*kfold <=m
                validation_S=Ss((jj-1)*kfold+1:jj*kfold,:);
            else %left-overs
                validation_S=Ss((jj-1)*kfold+1:end,:);
            end
            %validation set
            if jj ==1 %first group
                trainset_S=Ss(jj*kfold+1:end,:);
                trainset_Y=Ys(jj*kfold+1:end,:);
            elseif 2<=jj && jj<= floor(m/kfold) && mod(m,kfold) >0 ||...
              2<=jj && mod(m,kfold)==0 && jj*kfold<m
                %group 2 till last full group
                trainset_S=[Ss(1:(jj-1)*kfold,:);Ss(jj*kfold+1:end,:)];
                trainset_Y=[Ys(1:(jj-1)*kfold,:);Ys(jj*kfold+1:end,:)];
            else %left-overs
                trainset_S=Ss(1:(jj-1)*kfold,:);
                trainset_Y=Ys(1:(jj-1)*kfold,:);
            end
            if 1<=jj && jj <= floor(m/kfold) $& mod(m,kfold) >0 ||...
              2<=jj && mod(m,kfold)==0 && jj*kfold<m
                % RBF model for prediction
                [lambda, gamma]=RBF(trainset_S,trainset_Y,'TPS');
                yPred_all((jj-1)*kfold+1:jj*kfold,1)=RBF_eval(validation_S,...
                  trainset_S,lambda,gamma,'TPS');
```

```
            % rduced quadratic polynomial model prediction
            beta=POLY(trainset_S,trainset_Y,'quadr'); %Polynomial
            yPred_all((jj-1)*kfold+1:jj*kfold,2)=POLY_eval(validation_S,beta,'quadr');
        else %left-overs
            id_a=(jj-1)*kfold+1;
            if mod(m,kfold)>0
                id_e=(jj-1)*kfold+mod(m,kfold);
            else
                id_e=jj*kfold;
            end
            % RBF model for prediction
            [lambda, gamma]=RBF(trainset_S,trainset_Y,'TPS');
            yPred_all(id_a:id_e,1)=RBF_eval(validation_S,trainset_S,...
              lambda,gamma,'TPS');
            % reduced quadratic polynomial model prediction
            beta=POLY(trainset_S,trainset_Y,'quadr');
            yPred_all(id_a:id_e,2)=POLY_eval(validation_S,beta,'quadr');
            yPred_all=sortrows([yPred_all,mix(:)],Data.numberOfModels+1);
            yPred_all=yPred_all(:,1:Data.numberOfModels);
        end
    end
end
```

If a new three-model mixture is introduced, the file DempsterFor3models.m must accordingly be altered.

The file PredictFunctionValues.m has to be extended by adding a branch to the if-elseif tree under the comment

%add more surrogate models here if desired:

```
elseif strcmp(Surrogate,'MIX_RBFtpsPqr') %Mixture model: RBF with tps & reduced quadratic polynomial
    CandValue_RBF = RBF_eval(CandPoint,Data.S,lambda,gamma,'TPS'); % objective function value prediction with cubic RBF model for all candidate points
    CandValue_Pqr=POLY_eval(CandPoint,beta,'quadr'); %objective function value prediction with reduced quadratic polynomial for all candidate points
    CandValue= w_m(1)*CandValue_RBF+w_m(2)*CandValue_Pqr; %weighted objective function value prediction
```

The file SurfaceMinSampling.m need to be altered only if the minimum of the response surface is chosen as sampling strategy (setting SURFmin). The if-elseif tree has to be extended under the comment

%add other 2-model mixtures here:

```
elseif strcmp(Surrogate,'MIX_RBFtpsPqr') %mixture model of Kriging with gaussian correlation and 1st order regression polynomial, and reduced quadratic polynomial
    myfun1=@(x)RBF_eval(x,Data.S,lambda,gamma,'TPS'); % cubic RBF model
    myfun2=@(x)POLY_eval(x,beta,'quadr'); %reduced quadratic model
    myfun=@(x)w_m(1)*myfun1(x)+w_m(2)*myfun2(x); %weighted mixture
```

The file ScoreMinSampling.m needs to be altered only if the minimum of the scoring function is used as sampling criterion. In that case the if-elseif tree needs to be extended under the comment

%add more 2-model mixtures here as follows:

```
elseif strcmp(Surrogate,'MIX_RBFtpsPqr') %mixture model of Kriging with gaussian correlation and 1st order regression polynomial, and reduced cubic polynomial
    myfun1=@(x)RBF_eval(x,Data.S,lambda,gamma,'TPS'); % cubic RBF model
    myfun2=@(x)POLY_eval(x,beta,'quadr'); %reduced quadratic model
    myfun=@(x)w_m(1)*myfun1(x)+w_m(2)*myfun2(x); %weighted mixture
```

If a completely new surrogate model is to be defined, two functions are needed, namely a first function that computes the model parameters (the output of FitSurrogateModel.m must be adjusted accordingly), and a second function that predicts the objective function values, where the predictions should be scalars.

10 Results

The algorithm saves the results of the optimization in every iteration to the file Results.mat. If the algorithm is not interrupted (e.g. by pressing CTRL+C), the following elements are contained in the saved Data structure (see Table 10).

Table 10: Saved data structure elements

Elements	Description
Data.xlow	vector with lower variable bound
Data.xup	vector with upper variable bounds
Data.dim	integer, problem dimension
Data.integer	vector with indices of integer variables
Data.continuous	vector with indices of continuous variables
Data.objfunction	function handle to objective function
Data.constraint	cell array with function handles to constraints; only for constrained problems
Data.S	matrix with sample sites
Data.Y	vector with objective function values corresponding to Data.S
Data.fevaltime	vector with time for each function evaluation
Data.gevaltime	vector with time for constraint evaluations; only for constrained problems
Data.pen	vector with squared constraint violations; only for constrained problems
Data.Feasible	binary, 1 if feasible solution found, 0 otherwise; only for constrained problems
Data.Fbest_inf	scalar, best infeasible function value found; only for constrained problems
Data.xbest_inf	vector, best infeasible point found; only for constrained problems
Data.xbest	vector, best (feasible) point found
Data.Ymed	vector with function values where large values have been replaced by median
Data.fbest	scalar, best feasible objective function value found
Data.Ypenalized	vector with penalty-augmented objective function values; only for constrained problems
Data.Problem	string, contains name of data file that specifies the optimization problem
Data.SurrogateModel	string, contains name of (mixture) surrogate model used in optimization
Data.SamplingTechnique	string, contains name of iterative sampling technique used in optimization
Data.InitialDesign	string, contains name of experimental design strategy
Data.NumberStartPoints	integer, number or points in initial experimental design
Data.StartingPoint	matrix, contains user-defined points that are added to the initial experimental design; only if defined, and for mixed-integer problems
Data.TotalTime	scalar, total computation time needed by the algorithm

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] A. Forrester, A. Sóbester, and A. Keane. *Engineering Design via Surrogate Modelling - A Practical Guide*. Wiley, 2008.

[3] H.-M. Gutmann. A radial basis function method for global optimization. *Journal of Global Optimization*, 19:201–227, 2001.

[4] K. Holmström, N.-H. Quttineh, and M.M. Edvall. An adaptive radial basis algorithm (ARBF) for expensive black-box mixed-integer constrained global optimization. *Journal of Global Optimization*, 41:447–464, 2008.

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

[6] D.R. Jones, M. Schonlau, and W.J. Welch. Efficient global optimization of expensive black-box functions. *Journal of Global Optimization*, 13:455–492, 1998.

[7] S.N. Lophaven, H.B. Nielsen, and J. Søndergaard. DACE a Matlab kriging toolbox. Technical report, Technical Report IMM-TR-2002-12, 2002.

[8] J. Müller and R. Piché. Mixture surrogate models based on Dempster-Shafer theory for global optimization problems. *Journal of Global Optimization*, 51:79–104, 2011.

[9] J. Müller, C.A. Shoemaker, and R. Piché. SO-MI: A surrogate model algorithm for computationally expensive nonlinear mixed-integer black-box global optimization problems. *Computers and Operations Research*, http://dx.doi.org/10.1016/j.cor.2012.08.022, 2012.

[10] 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.

[11] R.G. Regis and C.A. Shoemaker. A quasi-multistart framework for global optimization of expensive functions using response surface models. *Journal of Global Optimization*, DOI 10.1007/s10898-012-9940-1, 2012.

[12] B.A. Tolson and C.A. Shoemaker. Dynamically dimensioned search algorithm for computationally efficient watershed model calibration. *Water Resources Research*, 43:W01413, 16 pages, 2007.

[13] 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.