

SPOT

Sequential Parameter Optimization Toolbox

Thomas Bartz-Beielstein

Faculty of Computer Science and Engineering Science
Cologne University of Applied Sciences
D-51643 Gummersbach
Germany

Christian Lasarczyk, Mike Preuß

Algorithm Engineering
Dept. of Computer Science
Dortmund University
D-44221 Dortmund
Germany
Version 0.3

January 11, 2007

Contents

1	Introduction	2
2	Requirements	3
3	Installation	4
3.1	A first example	4
3.2	Adapting SPOT Settings to Your Needs	5
3.2.1	Modifying the Region of Interest (Algorithm Design) . . .	5
3.2.2	Modifying the Problem Dimension (Problem Design) . . .	9
3.2.3	Modifying Factors that Remain Constant (Algorithm Design)	9
3.2.4	Summary	9
4	Design and Result Files	9
4.1	Design Files	10
4.2	Result Files	10

5	Examples	10
5.1	Downloading and installing the SPOT program	11
5.2	Optimizing a JAVA algorithm	11
5.2.1	Downloading and installing the JAVA program	11
5.2.2	The SPOT interface to the JAVA program	11
5.2.3	Files for the JAVA ES Optimization	12
5.3	Optimizing a particle swarm optimization (MATLAB)	12
5.3.1	Downloading and installing the particle swarm optimization	12
5.3.2	The SPOT interface to the PSOTOOLBOX	12
6	Common Errors	12
7	Questions	12
8	Functions	15
8.1	SPOT	15
8.2	Additional files for the SPO toolbox	16
8.3	esmatlab Toolbox	17
9	Parameters, Variables, Factors	18
9.1	SPOT	18
9.2	esmatlab	18

1 Introduction

This article describes the SPO toolbox. SPO is an acronym for *sequential parameter optimization* (Bartz-Beielstein, 2006). The *SPOtoolbox* described in this article is referred to as SPOT. It was developed over the last years by Thomas Bartz-Beielstein, Christian Lasarczyk, and Mike Preuß at the Chair of Algorithm Engineering, Dortmund University. The main purpose of SPO is to determine improved parameter settings for optimization algorithms to analyze and understand their performance.

SPO was successfully applied to numerous optimization algorithms, especially in the field of evolutionary computation, i.e., evolution strategies, particle swarm optimization, algorithmic chemistries etc. in the following domains:

- machine engineering: design of mold temperature control (Mehnen et al., 2005; Weinert et al., 2004; Mehnen et al., 2004)
- aerospace industry: airfoil design optimization (Bartz-Beielstein & Naujoks, 2004)
- simulation and optimization: elevator group control (Bartz-Beielstein et al., 2005c; Markon et al., 2006)
- technical thermodynamics: non sharp separation (Bartz-Beielstein et al., 2005b)

- economy: agri-environmental policy-switchings (de Vegt, 2005)

Other fields of application are in fundamental research:

- algorithm engineering: graph drawing (Tosic, 2006)
- statistics: selection under uncertainty (optimal computational budget allocation) for PSO (Bartz-Beielstein et al., 2005a)
- evolution strategies: threshold selection and step-size adaptation (Bartz-Beielstein, 2005)
- other evolutionary algorithms: genetic chromodynamics (Stoean et al., 2005)
- computational intelligence: algorithmic chemistry (Bartz-Beielstein et al., 2005b)
- particle swarm optimization: analysis and application (Bartz-Beielstein et al., 2004a)
- numerics: comparison and analysis of classical and modern optimization algorithms (Bartz-Beielstein et al., 2004b)

Further projects, e.g., vehicle routing and door-assignment problems and the application of methods from computational intelligence to problems from bioinformatics are subject of current research.

SPOT relies on functions provided by the MATLAB Kriging Toolbox DACE developed by Lophaven et al. (2002).

2 Requirements

SPOT requires MATLAB, the MATLAB statistics toolbox, and the MATLABDACE toolbox. It runs under different operating systems. Experiments described in Bartz-Beielstein (2006) were performed on various Linux Systems (Debian, Suse).

MATLAB's `tdfread` function, which belongs to the statistic toolbox, uses the `assignin` function to export data from a function to the MATLAB workspace. This has to be modified, because the data should be exported to the caller function. Please perform the following steps to fix this problem:

1. Enter in the command window: `type tdfread`
2. Copy the result and save it as a new MATLAB function, i.e., `spottdfread.m`
3. Replace every occurrence of the string 'base' with 'caller' in `spottdfread.m`
4. Save the modified `spottdfread.m` in your MATLAB path.

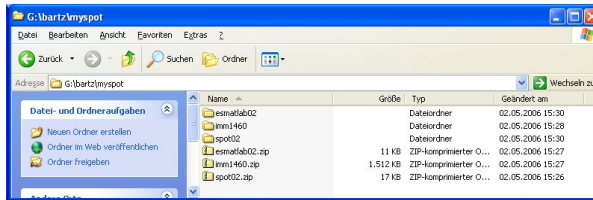


Figure 1: Directory structure after the files have been unzipped

Due to copyright restrictions, a modified `tdfread.m` is not delivered with the SPO toolbox. We have contacted MathWorks Inc. already, so that this problem may be solved in further versions.

3 Installation

3.1 A first example

1. Create a new directory, e.g., `g:\myspot`.
2. Download the `spot03.zip` file from <http://ls11-www.cs.uni-dortmund.de/people/tom/spot03.zip> and unzip it to `g:\myspot`.
3. Download and unzip the MATLAB DACE toolbox, which is freely available from: <http://www2.imm.dtu.dk/~hbn/dace/> to a directory of your choice, e.g., `g:\myspot`.
4. Start MATLAB. SPOT can be used to tune every algorithm that allows (or requires) the specification of exogenous parameters. Since there are several algorithms available, we use an *evolution strategy* (ES) to exemplify the procedure first. In a second step, we illustrate how this procedure can be generalized. To continue with the ES example, the reader should download and unzip the ES package to a directory of her choice, e.g., `g:\myspot`. The ES package is freely available from <http://ls11-www.cs.uni-dortmund.de/people/tom/esmatlab03.zip>
5. The resulting directory structure is shown in Fig. 1. MATLAB should be started in `g:\myspot`.
6. Add the ES toolbox directories to the MATLAB path. This can be done manually or you can also use `startspot.m` to add `myspot` and its subdirectories to the path from the command line. Please note, that the ES and the SPOT package can be used independently.
7. That is all. Now you can call `demoSpotMatlab('demo1')` to generate the initial designs and improve the performance of the ES algorithm sequentially. First `demoSpotMatlab` generates the initial design and starts the sequential parameter optimization next. You can delete results from previous run by selecting `new=1`. Otherwise, if you have chosen `new=0` existing

results are included in the current model. After the runs are finished, Fig. 2 and 3 are displayed.

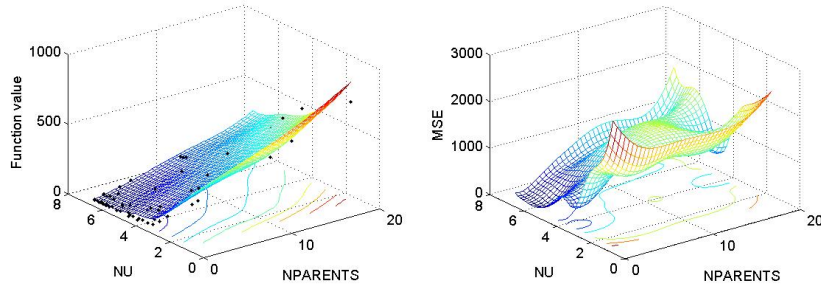


Figure 2: *Left*: Predicted values. *Right*: Mean squared error

3.2 Adapting SPOT Settings to Your Needs

Based on the example from Sect. 3.1, we demonstrate several features of the SPO toolbox. Note, there are configuration files related to SPOT:

1. ROI files, e.g., `demo1.roi`
2. CONF files, e.g., `demo1.m`

and files related to the algorithm

1. `esdemo1.m`

3.2.1 Modifying the Region of Interest (Algorithm Design)

The *region of interest* (ROI) contains interesting algorithm design points, e.g., interesting settings for the population size. Figure 4 reveals that small population sizes (NPARENTS) and low selective pressures (NU) worsen the performance of the ES. To adjust the region of interest, the ROI file can be modified

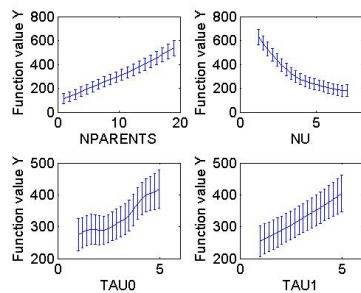


Figure 3: Effect plots

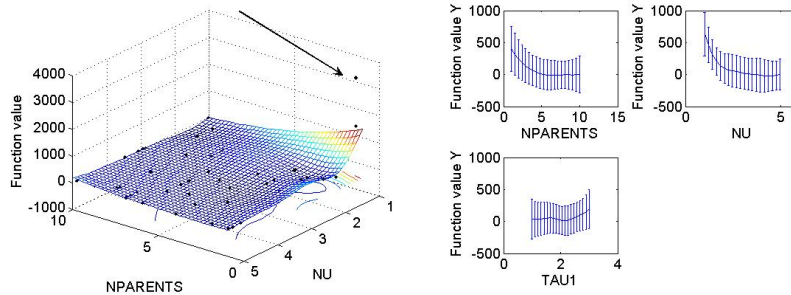


Figure 4: *Left:* Predicted values. *Right:* Effects

```
name low high isint pretty
NPARENTS 1 10 TRUE 'NPARENTS'
NU 1 5 FALSE 'NU'
TAU1 0.5 3 FALSE 'TAU1'
```

Figure 5: demo1.roi

using a simple text editor. Figure 5 shows the roi file `demo1.roi` from the example. The region of interest was modified as shown in Fig. 6 and saved as a new file `demo2.roi`. Alternatively, a modification of the file `demo1.roi` would have been possible, too. However, by saving the modifications in new files results can be reproduced very easily. SPOT uses a configuration files like `demo1.m` to specify SPOT related variables. `demo1` is the prefix of the names used for all files related to one experimental run, i.e., the resultfile, the designfile, and the roifile: `demo1.res`, `demo1.des`, `demo1.roi`, respectively.

The optimization algorithm, i.e., the ES, reads a design file written by SPOT and writes a result file which will be used by SPOT to fit the stochastic process model. The name of design file is `demo1.des`. The ES in our example adds the string 'es' to the filenames, so it reads automatically the configuration file 'esdemo1.m'.

If the ES is started, it opens a file `demo1.des` to read the parameter and writes results to the resultfile `demo1.res`.

```
name low high isint pretty
NPARENTS 2 10 TRUE 'NPARENTS'
NU 2 5 FALSE 'NU'
TAU1 1 3 FALSE 'TAU1'
```

Figure 6: `demo2.roi`. It defines an improved design based on results from the design specified in Fig. 5.

```

new=0
defaulttheta=1
loval=1E-3
upval=100
spotrmodel='regpoly2'
spotcmodel='corrgauss'
isotropic=0
repeats=3
maxrepeats=100
mergetype=1
ntestpointsperdim=10000
seed=0
na=40
newdesignsize=4
lhdsamples=30
lhdintervals=0
budget=250
nsteps=Inf
spotdirname='spot03/'
dacemodelname=''
npoints=10
tol=1.e-2
algorithmname= ''
yname='YALL'

```

Figure 7: demo1.m. Table 1 explains these variables.

<code>new</code>	To start a new run, especially with filenames that have been used for other runs and these files should be overwritten, set this value to "1". To continue an existing experiment, choose "0".
<code>spotdirname</code>	Directory where the optimization algorithm reads configuration and writes results files (use a closing \ or /). Default: Directory where the spot files reside.
<code>isotropic</code>	Define number of theta values, default is # of factors, which defines anisotropic models. Set nTheta to 1 to use isotropic models.
<code>nsteps</code>	Number of sequential optimization steps to be performed.
<code>algorithmname</code>	Call to the algorithm to be tuned. A script or batch file that calls the optimization algorithm has to be specified here.

```

esdirname=pwd
nparents = 1
nu = 1
sigmanull = 1
nsigma = 1
taunull = 1
tauone = 1
rho = 2
xreco = 'disc'
sreco = 'inter'
xrscale = 0.5
srscale = 0.5
kappa = Inf
fname = 'Sphere'
startpoint = 1000
lb = -22
ub = 42
stepinit = 'detmod'
delta = 0
dim = 10
init = 'nunirnd'
tmax = 1000
termcriteria = 'iterations'
noisetype = 'noNoise'
noisedistribution = 'noNoise'
noiselevel = 0
testset = 'standard'
gm = 0
xopt = 0
successlimit = 0.01
showrun = 0
showpoints = 0
showendpoints = 0
showsuccessfulstartpoints = 0
showunsuccessfulstartpoints = 0
interval = 0
intervalsize = 100

```

Figure 8: esdemo1.m

Table 2: SPOT: esdemo1

esdirname	Path to the *des and *res-files
-----------	---------------------------------

3.2.2 Modifying the Problem Dimension (Problem Design)

The experimenter might be interested in the question if similar algorithm designs are beneficial for different problem dimensions. Therefore, he has to modify the problem design. The problem design subsumes information about the objective function, e.g., the problem dimension, and specifies the available budget, e.g., time, number of function evaluations etc. To modify the problem dimension, the file `esdemo1.m` (Fig. 8) has to be edited. For example, using a text editor, you can change `dim = 2` to `dim =10` to analyze a 10 dimensional problem.

3.2.3 Modifying Factors that Remain Constant (Algorithm Design)

Factors that are varied during the sequential parameter optimization are specified in the ROI file. Many algorithms have factors that remain constant during the sequential parameter optimization. They are specified in the `esdemo1.m` file. By default, every factor of the optimization algorithm is specified in the `esdemo1.m` file, SPO overwrites these values with values generated in the ROI file. Since `nparents`, `nu`, and `tau1` are listed in the ROI file, we can modify the `kappa` value in the `esdemo1.m` file.

3.2.4 Summary

SPO requires the specification of the following parameters:

1. Algorithm parameters to be tuned and varied. They are specified in the ROI file, e.g., `demo1.roi`.
2. SPOT specific parameters, e.g., (`demo1.m`)

In addition, the algorithm which should be tuned requires the specification of the following parameters:

1. Algorithm parameters that remain constant
2. Problem specific parameters

In our example (`esdemo1.m`) was used to specify these parameters.

4 Design and Result Files

The SPOT concept is very simple and can be used for any optimization algorithm that can read and write ASCII files.

SPOT proposes algorithm designs for the algorithm. The algorithm is run with these designs, results from these runs are written to *result files* (res-files). SPOT reads the res-files and builds a stochastic process model. Based on this model, new algorithm design points are generated and written to the *design file* (des-file). See also Bartz-Beielstein et al. (2005b).

4.1 Design Files

Design files are ASCII files which store information columnwise. They consist of two parts: Header and body. The first line of a design file, the header, describes the variable names. Entries are separated by blanks. Each of the following lines contains information from one algorithm run. A minimum design file has two lines. It might contain the following information:

```
A REPEATS CONFIG SEED
2 3 1 123
```

“A” denotes the name of algorithm parameter to be tuned, e.g., NPARENTS. Each run is repeated three times with different random seeds, the first run uses seed 123. This is configuration 1.

Based on the values from the des-file the optimization algorithm is run. Its results are written to the res-file.

The header has to be the first line while the order of the other lines (rows) is arbitrary. The order of the columns is arbitrary, too. The example from above could be written as:

```
REPEATS CONFIG SEED A
3 1 123 2
```

4.2 Result Files

The structure of result files is the same as for design files: They are des-files with one additional column which contains the result from the optimization runs. A minimum result file contains the following information:

```
Y A REPEATS CONFIG SEED
0.2 2 3 1 123
```

This might be one result from an experiment which is based on the des-file from above. The algorithm run yields an function value of $Y = 0.2$.

Based on the values from the res-file SPOT will fit a stochastic process model to predict promising design points.

5 Examples

The following examples describe simple parameter optimization scenarios. Section 5.1 describes the installation of the SPOTMATLAB files. This installation is required for any of the following scenarios:

- Section ?? shows how SPOT can be used to tune an evolution strategy that was written in MATLAB.
- Section 5.3 demonstrates how an existing optimization toolbox, the particle swarm optimization toolbox written by Jagatpreet Singh <http://psotoolbox.sourceforge.net/> can be integrated into SPOT.

- Section 5.2 explains how to tune an evolution strategy which was implemented in JAVA with SPOT.

5.1 Downloading and installing the SPOT program

1. Download the SPOT and DACE packages. In our example, both files are downloaded to a newly generated directory, i.e., `c:\myspot`. After the SPOT and DACE packages are unzipped, the following directories exist:

- (a) `c:\myspot\spot03`
- (b) `c:\myspot\imm1460\dace25`

2. Add these directories to the MATLAB path.

Note, the previous steps were already described in Sect. 3. Now we consider how implementing interfaces to other optimizers.

5.2 Optimizing a JAVA algorithm

5.2.1 Downloading and installing the JAVA program

We use an evolution strategy (ES) which is implemented in JAVA. The zip file can be downloaded from:

<http://ls11-www.cs.uni-dortmund.de/people/tom/esjava.zip>. Again, download the zip file and unzip it to a newly generated directory, i.e., `c:\myspot\esjava`. The directory should now contain the evolution strategy and a parameter file `demojava1.par`¹, that is needed to start the ES run.

5.2.2 The SPOT interface to the JAVA program

The command

```
java -jar es.jar demojava1.par
```

should be written to a batch file (Windows, e.g., `runJava.bat`) or to a shell script (UNIX, e.g., `runJava.sh`), which can be called from within MATLAB.

The passage from the MATLAB script `demoSpotJava.m` reads:

```
%(8) Call the algorithm to be tuned.
if SpotOptions.System.Unix
    [u v] = unix('./esjava/runJava.sh')
else
    [u v] = dos('./\esjava\runJava.bat');
end
```

The batch file `runJava.bat` reads:

```
cd c:\myspot\esjava
```

```
java -jar es.jar demojava1.par
```

The batch file from this example can be downloaded from <http://ls11-www.cs.uni-dortmund.de/people/tom/runJava.bat>

¹The documentation of the parameters in the file `demoSpot.par` can be downloaded from <http://ls11-www.cs.uni-dortmund.de/people/tom/esParameter.htm>

5.2.3 Files for the JAVA ES Optimization

The following files are used to specify the experimental setup:

1. `demojava1.roi`: Region of interest.
2. `demojava1.m`: SPOT settings.
3. `demojava1.par`: ES parameter file.
4. `runJava.bat` (WINDOWS) or `runJava.sh` (UNIX): Batch file/script for calling the optimization algorithm from SPO.

5.3 Optimizing a particle swarm optimization (MATLAB)

5.3.1 Downloading and installing the particle swarm optimization

We use the `psotoolbox` which can be downloaded from <http://psotoolbox.sourceforge.net/>. This toolbox is a collection of MATLAB files. Downloading and unzipping the toolbox might create the following directory:

`c:\myspot\spot03\pso\psotb-beta-0.3` Add this directory to the MATLAB path and test the toolbox by starting the function `RunExp.m`.

5.3.2 The SPOT interface to the PSOTOOLBOX

Based on the `RunExp.m` function we implement an interface to SPOT.

TBD

6 Common Errors

Table 3 lists errors, possible causes, and solutions.

7 Questions

- *Where can I find more information about the DACE toolbox?*
Check: <http://www2.imm.dtu.dk/~hbn/dace/>
- *My optimization algorithm was written in JAVA, C, C++,... Can I use SPOT?*
You can use any language if your algorithm is able to handle `des-` and `res-`files.
- *Does SPOT run without MATLAB?*
No.
- *My optimization algorithm is written in MATLAB. Is there a generic method to read `des-`files?*
Yes. Reading `des-`files from MATLAB can be done as follows.

Table 3: Common errors.

Error message	Possible cause	Solution
Unable to open file. ??? Undefined function or variable 'Y'.	Wrong MATLAB working directory	Change the MATLAB working directory. The MATLAB <code>cd</code> command changes the working directory. Alternatively you can use the Current Directory field in the MATLAB desktop toolbar.
	Batch file/shell script is not executed correctly	Test the script (permissions etc.) in a DOS/UNIX shell separately. If it works fine, you can execute it in the MATLAB command window.
??? Error using ==> dacefit least squares problem is underdetermined	Not enough design points	Increase the number of design points by modifying the value of <code>SpotOptions.Design.LHDSamples</code> . Reduce the model complexity.
??? Error using ==> textread File not found. Error in ==> spotreadroi at 6 [factornames, lb, ub, whole, pp] = textread(spotoptions.File.roifilename,... Error in ==> demoSpotJava at 18 SpotOptions = spotreadroi(SpotOptions);	Wrong <code>spotdirname</code>	Select the correct value for <code>spotdirname</code> in <code>demoSpotJava.m</code> or related files.
??? Error using ==> eval Undefined function or variable 'Y'. Error in ==> demoSpotJava at 64 Y=eval(SpotOptions.Vars.yname)	Wrong dir for external batch file (shell script)	Select the correct setting in external files, e.g., <code>runJava.bat</code>
??? Error using ==> predictor DMODEL has not been found Error in ==> demoSpotMatlab at 147 [yLhd0, dy0, mse0] = predictor(xLhd(i,:), dmodel0);	Result file contains non numerical results (Y values), e.g., Inf 13	1. Modify algorithm or problem designs to avoid divergent behaviour or 2. Replace "Inf" with large values (not recommended)

1. Specify the algorithm design variables from your algorithm, e.g., mu and nu:

```
factornames = {'MU', 'NU'};
```

2. Read the des-file:

```
tdfread(designfilename, ' ');
```

3. Overwrite default values with values from the des-file:

```
for i=1:length(factornames)
    A(i)=factornames(i);
end
for i=1:length(eval(A{i})) %number of experiments
    for j=1:REPEATS(i) %each experimental setting is
        % repeated REPEAT(i) times.
            % Overwrite default values with algorithm
            % design values.
            if ismember('MU', A)
                mu = MU(i);
            end
            if ismember('NU', A)
                nu = NU(i);
            end
            ...
            RUN YOUR ALGORITHM
            ...
            WRITE RESULT FILE
        end
    end
end
```

- *I found a bug in the toolbox.*
Please report bugs and suggestions for improvement to thomas.bartz-beielstein@udo.edu.
- *Is there any support available?*
Currently we cannot provide any support.
- *Is SPOT free?*
Yes.
- *Where can I find more information about experimental research, design of experiments (DOE), and design and analysis of computer experiments (DACE)?*
Bartz-Beielstein (2006) and Santner et al. (2003) are good starting points.
- *How can I quote this article?*
Please mention Bartz-Beielstein et al. (2006) if you have used the SPO toolbox.

8 Functions

8.1 SPOT

The SPO toolbox contains the following MATLAB functions

`startspot.m`: Helper function to set the path and working directory.

`spotWriteNewDesign.m`: Update the design file based on predictions from the DACE model.

`spotWriteInitialDesign.m`: Write the initial design based on the LHD.

`spotWriteDaceModel.m`: Write DACE model parameters to a `*.dac` file.

`spottdfread.m`: This function is required, but due to copyright problems not included in the SPOT packages. Users have to copy the `tdfread.m` to a new file, i.e., `spottdfread.m` and perform some minor editing as explained in Sect. 2.

`spotreadroi.m`: Read the region of interest file.

`spotreadfactornames.m`: Returns factornames from res and des files.

`spotPlotEffectsSchonlau.m`: Plot the factor effects. Based on Schonlau (1997).

`spotPlotDACEModel.m`: Plot the 3 dim DACE model (the first two factors from the ROI file and the predicted response).

`spotMergeData.m`: Merge data from several repeats of one run configuration. The following merge functions are implemented:

1. mean (1)
2. median (2)
3. min (3)
4. max (4)
5. spotbestoutof (4)

This function can also return the standard deviation (0).

`spotLatinHypercube.m`: Generate Latin hypercube distributed random numbers:

- m : number of sample points to generate, if unspecified $m = 1$.
- n : number of dimensions, if unspecified $n = m$

Returns S , the generated n dimensional m sample points chosen from uniform distributions on m subdivisions of the interval $(0.0, 1.0)$ Example: Let $a = [1\ 2\ 3\ 4\ 5\ 6\ 7]$, be the vector of the lower bounds for 7 variables, e.g., PSO, and $b = [10\ 20\ 30\ 40\ 50\ 60\ 70]$ the vector of the upper bounds. Then `latinHypercube(10,7,a,b)` generates 10 design points.

`spotGetOptions.m`: Read the SPOT options file.

`spotGenerateFileNames.m`: Generate generic file names, i.e.,

1. des
2. bst
3. res
4. dac
5. bcl
6. xbst
7. fbst
8. roi

`spotCleanUp.m`: Delete the following files to enable a clean start:

1. des
2. bst
3. res
4. dac
5. bcl
6. xbst
7. fbst

`spotbestoutof.m`: Determines the average from `nSamples` that have been created as the minimum from a random sample of size `sampleSize`. Note: `spotbestoutof(x,inf,1) = mean(x)`, `spotbestoutof(x,1,inf) = min(x)`

`demoSpotMatlab.m`: Main program.

8.2 Additional files for the SPO toolbox

`demospotplotresfile.m`: Visualize the results from an existing resfile.

8.3 esmatlab Toolbox

The `esmatlab` toolbox contains the following MATLAB functions

`addNoise.m`: Add noise to the function value.

`eseval.m`: Enable evaluation of test functions from the More et al. (1981) test set.

`esInitPopulation`: Implement initialization schemes from Bartz-Beielstein (2006, pp. 88) To initialize the set of search points $X^{(0)} = \{x_1^{(0)}, \dots, x_p^{(0)}\}$, the following methods can be used:

(DETEQ) *Deterministic*. Each search point uses the same vector, which is selected deterministically, i.e., $x_{\text{init}} = \mathbf{1}^T \in \mathbb{R}^d$. As this method uses only one starting point x_{init} , it is not suitable to visualize the starting points for which the algorithm converged to the optimum.

Example 1 Schwefel (1995) proposed the following initialization scheme for high-dimensional nonquadratic problems:

$$x_i^{(0)} = x^* + \frac{(-1)^i}{\sqrt{d}}, \quad \text{for } i = 1, \dots, d. \quad (1)$$

■

(DETMOD) *Deterministically modified starting vectors*. The algorithm can be tested with starting vectors $x^{(0)}$, $10x^{(0)}$, and $100x^{(0)}$ More et al. (1981), or any other scheme that generates starting points deterministically.

(UNIRND) *Uniform random starts*. Every search point ($i = 1, \dots, p$) uses the same vector $x_{\text{init}} \in \mathbb{R}^d$, where the d components are realizations of independent $U[x_l, x_u]$ random variables. This method introduces an additional source of randomness. It is suitable to visualize the starting points for which the algorithm converged to the optimum. This visualization technique is useful to get some insight into the behavior of the algorithm.

(NUNIRND) *Nonuniform random starts*. Every search point uses a different vector $x_i^{(0)}$, ($i = 1, \dots, p$), that is, $X^{(0)} = \{x_1^{(0)}, \dots, x_p^{(0)}\}$, with $x_i^{(0)} \neq x_j^{(0)} \forall i \neq j$. Each of the p vectors $x_{\text{init}} \in \mathbb{R}^d$ consists of d components that are realizations of independent $U[x_l, x_u]$ random variables. This initialization method is used by many authors. It introduces an additional source of randomness, and it is not suitable to visualize the starting points for which the algorithm converged to the optimum.

`esInitStepSizes.m`: Initialization of the step sizes.

`esKappaSelection.m`: Environmental selection for ES.

`esMakeHandle.m`: Function handle.

`esMutation.m`: Mutation as described in Beyer & Schwefel (2002).

`esrecombination.m`: Implements discrete, intermediate, scalable, and no recombination.

`eswritexbest.m`: Write object variables (x values) of the best individual to the `xbest` file.

`terminate.m`: Implement three termination criteria, see Bartz-Beielstein (2006, p.89):

1. budget (number of function evaluations) exhausted
2. minimum distance in the objective space reached
3. minimum distance to the optimum reached

`esdemo1.m`: Example ES configuration file. Note, that in our implementation the name of the ES configuration file is generated by adding the prefix ‘es’ to the SPOT configuration file name, i.e., ‘demo1’ results in ‘esdemo1’. This is done in `esGetOptions.m`.

`esGetOptions.m`: Read options (problem and algorithm designs) from an options file, e.g., `esdemo1.m`.

`generateresfile.m`: Write data to the resultfile.

`esSpot.m`: The ES in MATLAB, i.e., the main class/function/method. It can also be used as a stand alone ES without the SPOT framework.

9 Parameters, Variables, Factors

This section describes factors and their meaning.

9.1 SPOT

TBD

9.2 esmatlab

lb lower initialization bounds for object variables

ub upper initialization bounds for object variables

References

- Bartz-Beielstein, T. (2005). Evolution strategies and threshold selection. In M. J. Blesa Aguilera, C. Blum, A. Roli, & M. Sampels (Eds.), *Proceedings Second International Workshop Hybrid Metaheuristics (HM'05)*, volume 3636 of *Lecture Notes in Computer Science* (pp. 104–115). Berlin, Heidelberg, New York: Springer.
- Bartz-Beielstein, T. (2006). *Experimental Research in Evolutionary Computation—The New Experimentalism*. Berlin, Heidelberg, New York: Springer.
- Bartz-Beielstein, T., Blum, D., & Branke, J. (2005a). Particle swarm optimization and sequential sampling in noisy environments. In R. Hartl & K. Doerner (Eds.), *Proceedings 6th Metaheuristics International Conference (MIC2005)* (pp. 89–94). Vienna, Austria.
- Bartz-Beielstein, T., de Vegt, M., Parsopoulos, K. E., & Vrahatis, M. N. (2004a). *Designing Particle Swarm Optimization with Regression Trees*. Interner Bericht des Sonderforschungsbereichs 531 Computational Intelligence CI–173/04, Universität Dortmund, Germany.
- Bartz-Beielstein, T., Lasarczyk, C., & Preuß, M. (2005b). Sequential parameter optimization. In B. McKay & others (Eds.), *Proceedings 2005 Congress on Evolutionary Computation (CEC'05), Edinburgh, Scotland*, volume 1 (pp. 773–780). Piscataway NJ: IEEE Press.
- Bartz-Beielstein, T., Lasarczyk, C., & Preuß, M. (2006). *Sequential Parameter Optimization Toolbox*. Technical Report CI–15x/06, Universität Dortmund, Germany.
- Bartz-Beielstein, T. & Naujoks, B. (2004). *Tuning Multicriteria Evolutionary Algorithms for Airfoil Design Optimization*. Interner Bericht des Sonderforschungsbereichs 531 Computational Intelligence CI–159/04, Universität Dortmund, Germany.
- Bartz-Beielstein, T., Parsopoulos, K. E., & Vrahatis, M. N. (2004b). Design and analysis of optimization algorithms using computational statistics. *Applied Numerical Analysis & Computational Mathematics (ANACM)*, 1(2), 413–433.
- Bartz-Beielstein, T., Preuß, M., & Markon, S. (2005c). Validation and optimization of an elevator simulation model with modern search heuristics. In T. Ibaraki, K. Nonobe, & M. Yagiura (Eds.), *Metaheuristics: Progress as Real Problem Solvers*, Operations Research/Computer Science Interfaces (pp. 109–128). Berlin, Heidelberg, New York: Springer.
- Beyer, H.-G. & Schwefel, H.-P. (2002). Evolution strategies—A comprehensive introduction. *Natural Computing*, 1, 3–52.

- de Vegt, M. (2005). *Einfluss verschiedener Parametrisierungen auf die Dynamik des Partikel-Schwarm-Verfahrens: Eine empirische Analyse*. Interner Bericht der Systems Analysis Research Group SYS-3/05, Universität Dortmund, Fachbereich Informatik, Germany.
- Lophaven, S., Nielsen, H., & Søndergaard, J. (2002). *DACE—A Matlab Kriging Toolbox*. Technical Report IMM-REP-2002-12, Informatics and Mathematical Modelling, Technical University of Denmark, Copenhagen, Denmark.
- Markon, S., Kita, H., Kise, H., & Bartz-Beielstein, T., Eds. (2006). *Modern Supervisory and Optimal Control with Applications in the Control of Passenger Traffic Systems in Buildings*. Berlin, Heidelberg, New York: Springer.
- Mehnen, J., Michelitsch, T., Bartz-Beielstein, T., & Henkenjohann, N. (2004). Systematic analyses of multi-objective evolutionary algorithms applied to real-world problems using statistical design of experiments. In R. Teti (Ed.), *Proceedings Fourth International Seminar Intelligent Computation in Manufacturing Engineering (CIRP ICME'04)*, volume 4 (pp. 171–178). Naples, Italy.
- Mehnen, J., Michelitsch, T., Bartz-Beielstein, T., & Lasarczyk, C. W. G. (2005). Multiobjective evolutionary design of mold temperature control using DACE for parameter optimization. In H. Pfützner & E. Leiss (Eds.), *Proceedings Twelfth International Symposium Interdisciplinary Electromagnetics, Mechanics, and Biomedical Problems (ISEM 2005)*, volume L11-1 (pp. 464–465). Vienna, Austria: Vienna Magnetics Group Reports.
- More, J. J., Garbow, B. S., & Hillstom, K. E. (1981). Testing unconstrained optimization software. *ACM Transactions on Mathematical Software*, 7(1), 17–41.
- Santner, T. J., Williams, B. J., & Notz, W. I. (2003). *The Design and Analysis of Computer Experiments*. Berlin, Heidelberg, New York: Springer.
- Schonlau, M. (1997). *Computer Experiments and Global Optimization*. PhD thesis, University of Waterloo, Ontario, Canada.
- Schwefel, H.-P. (1995). *Evolution and Optimum Seeking*. Sixth-Generation Computer Technology. New York NY: Wiley.
- Stoian, C., Preuss, M., Gorunescu, R., & Dumitrescu, D. (2005). Elitist Generational Genetic Chromodynamics - a New Ranks-Based Evolutionary Algorithm for Multimodal Optimization. In B. McKay & others (Eds.), *Proc. 2005 Congress on Evolutionary Computation (CEC'05)*, volume 2 (pp. 1839 – 1846). Piscataway NJ: IEEE Press.
- Tosic, M. (2006). Evolutionäre Kreuzungsminimierung. Diploma thesis, University of Dortmund, Germany.

Weinert, K., Mehnen, J., Michelitsch, T., Schmitt, K., & Bartz-Beielstein, T. (2004). A multiobjective approach to optimize temperature control systems of moulding tools. *Production Engineering Research and Development, Annals of the German Academic Society for Production Engineering*, XI(1), 77–80.

Index

CONFIG, 10

des-file, *see* design file

design file, 9, 10

evolution strategy (ES), 11

initialization method

 DETEQ, 17

 DETMOD, 17

 NUNIRND, 17

 UNIRND, 17

installation, 4

MATLAB DACE toolbox, 3

matlab path, 11

region of interest (ROI), 5

REPEATS, 10

res-file, *see* result file

result file, 9, 10

SEED, 10

sequential parameter optimization (SPO),
 2

sequential parameter optimization tool-
 box (SPOT), 2

startspot.m, 4