SSmGO Toolbox: SSm & eSS User's guide + tutorial

Process Engineering group
Instituto de Investigaciones Marinas (C.S.I.C.)
c/Eduardo Cabello, 6
36208, Vigo (Spain)
gingproc@iim.csic.es

October 28, 2010

Contents

1 Introduction

The SSmGO Toolbox contains the scatter search-based methods *SSm* (Rodríguez-Fernández et al., 2006; Egea et al., 2007) and *eSS* (Egea et al., 2009, 2010), which seek the global minimum of Mixed-integer nonlinear programming (MINLP) problems specified by

$$\min_x f(x, p_1, p_2, ..., p_n)$$

subject to

$$\begin{aligned} c_{eq} &= 0 \\ c_L \leq c(x) &\leq c_U \\ x_L \leq x &\leq x_U \end{aligned}$$

where x is the vector of decision variables, and x_L and x_U its respective bounds. $p_1, \ldots, p_n$ are optional extra input parameters to be passed to the objective function (see examples in Sections 5.3, 5.5 and 5.6). c_{eq} is a set of equality constraints. $c(x)$ is a set of inequality constraints with lower and upper bounds, c_L and c_U. Finally, $f(x, p_1, p_2, ..., p_n)$ is the objective function to be minimized.

Both *SSm* and *eSS* are written in Matlab. *SSm* is based on the design described by Laguna and Martí (2003) for continuous optimization problems. *eSS* presents a modified design which converges faster to the optimal solutions using less tuning parameters.

2 Quick start: How to carry out an optimization with the *SSm* Toolbox

- Start Matlab
- Go to the SSm Toolbox folder and run the script `ssm_startup`
- Define your problem and options (see the following sections)
- Type: `Results=ssm_kernel(problem,opts)` or `Results=ess_kernel(problem,opts)`. If your problem has additional constant parameters to be passed to the objective function, they are declared as input parameters after the "opts" (e.g., for two extra input parameters, p1 and p2: `Results=ess_kernel(problem,opts,p1,p2)`).

Regarding the objective function, the input parameter is the decision vector (with extra parameters p1, p2,..., pn if they were defined before calling the solver). The objective function must provide a scalar output parameter (the objective function value) and, for constrained problems, a second output parameter, which is a vector containing the values of the constraints. For problems containing equality constraints (= 0), they must be defined before the inequality constraints. Some examples are provided in section 5. For

a quick reference, consider the following example which will be later extended in section 5.3.

$$\min_x f(x) = -x_4$$

subject to

$$\begin{aligned} x_4 - x_3 + x_2 - x_1 + k_4 x_4 x_6 &= 0 \\ x_1 - 1 + k_1 x_1 x_5 &= 0 \\ x_2 - x_1 + k_2 x_2 x_6 &= 0 \\ x_3 + x_1 - 1 + k_3 x_3 x_5 &= 0 \\ x_5^{0.5} + x_6^{0.5} &\leq 4 \\ 0 \leq x_1, x_2, x_3, x_4 &\leq 1 \\ 0 \leq x_5, x_6 &\leq 16 \end{aligned}$$

with k_1, k_2, k_3, k_4 being extra parameters defined before calling the solver. The objective function for this problem would be:

example of objective function

```
function [f,g]=obj_func(x,k1,k2,k3,k4)
f=-x(4);

%Equality constraints. Declared BEFORE inequality constraints
g(1)=x(4)-x(3)+x(2)-x(1)+k4*x(4).*x(6);
g(2)=x(1)-1+k1*x(1).*x(5);
g(3)=x(2)-x(1)+k2*x(2).*x(6);
g(4)=x(3)+x(1)-1+k3*x(3).*x(5);

%Inequality constraint
g(5)=x(5).^0.5+x(6).^0.5;

return
```

2.1 *SSm* or *eSS*?

Having two different optimization solvers in SSmGOToolbox, which one should we use? In general, *eSS* is more efficient than *SSm*, mainly due to its global search design. It is also faster than *SSm*, specially for large-scale problems. Besides, it contains less tuning parameters (i.e., options). *SSm* may result more useful for instances where a higher diversity is needed due to its higher number of tuning parameters. Therefore, we highly recommend the use of *eSS* as the first option to solve global optimization problems. *SSm* should only be used if the results with *eSS* are not satisfactory. Figure 1 shows the performance profiles resulting from the application of both solvers to the set of problems solved in Egea (2008)

For users who have previously solved problems with *SSm*, swtiching to *eSS* is quite straightforward. The only real difference when calling these methods is the number of options that can be defined for each of them (see details below in this manual).

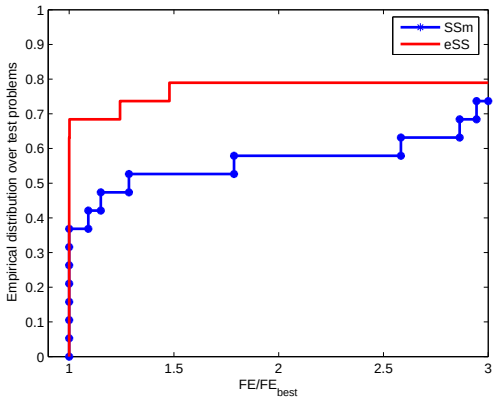

Figure 1: Performance profiles resulting from the application of *SSm* and *eSS* to the set of problems presented in Egea (2008)

3 *SSmGO* toolbox

3.1 Problem definition

In order to solve an optimization problem with either *SSm* or *eSS*, a structure (named **problem** here)containing the following fields must be defined:

- **f**: String containing the name of the objective function.
- **x_L**: Vector containing the lower bounds of the variables.
- **x_U**: Vector containing the upper bounds of the variables.
- **x_0**: Vector containing the given initial point (optional)*.
- **vtr**: Objective function value to be reached (optional).

If the problem contains additional constraints and/or integer or binary variables, the following files should also be defined:

- **neq**†: Number of equality (= 0) constraints.
- **c_L**: Vector defining the lower bounds of the inequality constraints.
- **c_U**: Vector defining the upper bounds of the inequality constraints.
- **int_var**‡: Number of integer variables.
- **bin_var**‡: Number of binary variables.

SSm accepts only one initial point whereas *eSS* accepts several of them (introduced as rows).

†In problems with equality constraints they must be declared first before inequality constraints (See example 5.3).

‡For mixed integer problems, the variables must be defined in the following order: [cont., int., bin.].

3.2 User options

The user may define a set of different options related to the optimization problem. They are defined in another structure (named **opts** here) which has the following fields.

- **maxeval**: Maximum number of function evaluations (default: 1000).
- **maxtime**: Maximum CPU time in seconds (default: 60).
- **iterprint**: Print information on the screen after each iteration. 0: Deactivated; 1: Activated (default: 1).
- **plot**: Plots convergence curves. 0: Deactivated; 1: Real Time; 2: Final results (default: 0).
- **weight**: Weight that multiplies the penalty term added to the objective function in constrained problems (default: 10^6).
- **log_var**: Indexes of the variables which will be analyzed using a logarithmic distribution instead of an uniform one[§] (default: [ ]). See an example in Section 5.5.
- **tolc**: Maximum violation constraints violation allowed. This is also used as a tolerance for the local search (default: 10^{-5}).
- **prob_bound**: Probability (0-1) of biasing the search toward the bounds. 0: Never bias to bounds; 1: Always bias to bounds (default: 0.5).
- **strategy***: If > 0 ignores user declared options and chooses these strategies: 1: Fast; 2: Average; 3: Robust (default: 0).
- **inter_save**: Saves results in a .mat file in intermediate iterations. Useful for very long runs. 0: deactivated; 1: activated (default: 0).

3.3 Global options

A set of options related to the global search phase of the algorithm may also be defined also within the structure **opts**:

- **dim_refset**: Number of elements d in *RefSet* (default: *'auto'*, $\frac{d^2-d}{10 \cdot nvar} \geq 0$).
- **ndiverse**: Number of solutions generated by the diversificator in the initial stage (default: *'auto'*, $10 \cdot nvar$).
- **initiate***: Type of *RefSet* initialization: 0: Take bounds, middle point and fill by Euclidean distance; 1: Evaluate all the diverse solutions, take the *dim_refset*/2 best solutions and fill by Euclidean distance (default: 1).
- **combination**: Type of combination of *RefSet* elements. 1: Hyper-rectangles combinations; 2: Linear combinations (default: 1).

[§]Useful when the bounds of a decision variables have different orders of magnitude and they are both positive.

*This option does not apply to *eSS*

- **regenerate**[*]: Type of *RefSet* regeneration. 1: Regeneration by distance diversity; 2: Regeneration by direction diversity; 3: Randomly alternates 1 and 2 (default: 3).
- **delete**[*]:Number of *RefSet* elements deleted in the regeneration step. *'standard'*: delete *dim_refset*/2 (the worst half *RefSet* members); *'aggressive'*: delete *dim_refset*−1 (all of them except the best *RefSet* member). Default: 'standard'.
- **intens**[*]: Iteration interval between intensifications (default: 10).
- **tolf**[*]: Function tolerance for joining the *RefSet* (default: 10^{-4}).
- **diverse_criteria**[*]: Diversification criterion in the *RefSet*. 1: Euclidean distance; 2: Tolerances (default: 1).
- **tolx**[*]: Variable tolerance for joining the *RefSet* when the Euclidean distance is deactivated (default: 10^{-3} for all variables).

3.4 Local options

SSm and *eSS* are global optimization methods which perform local searches from selected initial points to accelerate the convergence to optimal solutions. Some options regarding the local search can be defined in a sub-structure within the options structure (named **opts.local** here), which has the following fields:

- **solver**: Local solver to perform the local search. Solvers available (names must be introduced as strings):
  - *fmincon*:[‡] Sequential quadratic programming method (The MathWorks^{TM}, 2008).
  - *solnp*: the SQP method[¶] by Ye (1987).
  - *fminsearch*:[‡]. Modification of the Simplex-based method implemented in Matlab (The MathWorks^{TM}, 2008) to handle bound constraints, by John D'Errico.
  - *dhc*: Direct search method (de la Maza and Yuret, 1994).
  - *hooke*: Hooke & Jeeves direct search method (Hooke and Jeeves, 1961)[‖].
  - *n2fb*: Specific method for non-linear least squares problems (Dennis et al., 1981).
  - *lsqnonlin*:[‡] Another method for non-linear least squares problems (The MathWorks^{TM}, 2008).
  - *ipopt*: Interior point method (Wächter and Biegler, 2006).
  - *misqp*[§]: Sequential quadratic programming method which handles integer and binary variables (Exler and Schittkowski, 2007).
- **tol**: Level of tolerance in local search. 1: Relaxed; 2: Medium; 3: Tight (default:2 in intermediate searches and 3 in the final stage).

[*]This option does not apply to *eSS*

[‡]Requires the Matlab Optimization Toolbox for its use.

[¶]The original Matlab code can be found at http://www.stanford.edu/ yyye/matlab/

[‖]C.T. Kelley's Matlab code is used (http://www4.ncsu.edu/ ctk/darts/hooke.m)

[§]Use this solver for MINLP problems.

- **iterprint**: Print each iteration of local solver on screen (only for local solvers that allow it). 0: Deactivated; 1: Activated (default: 0).
- **n1**[†]: Number of function evaluations before applying the local search for the first time (default: $100 \cdot nvar$).
- **n2**[†]: Minimum number of function evaluations in the global phase between two consecutive local searches (default: $200 \cdot nvar$).
- **finish**: Applies local search to the best solution found once the optimization is finished (default: same as *opts.local.solver*).
- **bestx**: If activated (i.e., positive value), applies the local search only when the algorithm finds a solution better than the current best solution. 0: Deactivated; 1: Activated (default: 0).
- **merit_filter**[*]: Activation of merit filter for local search. 0: Deactivated; 1: Activated (default: 1).
- **distance_filter**[*]: Activation of distance filter for local search. 0: Deactivated; 1: Activated (default: 1).
- **thfactor**[*]: Merit filter relaxation parameter. (default: 0.2).
- **maxdistfactor**[*]: Distance filter relaxation parameter. (default: 0.2).
- **wait_th_limit**[*]: Apply merit filter relaxation after this number of function evaluations without success in passing the merit filter (default: 20).
- **wait_maxdist**[*]: Apply distance filter relaxation after this number of function evaluations without success in passing the distance filter (default: 20).
- **balance**[¶]: Balances between quality (=0) and diversity (=1) for choosing initial points for the local search (default 0.5).

Note that, for some problems, the local search may be inefficient, spending a high computation time to provide low quality solutions. This is the case of many noisy or ill-posed problems. In these instances, the local search may be deactivated by user by defining the value of the field **solver** as zero.

When using *n2fb* (or *dn2fb*) and *lsqnonlin* as local solvers, the objective function value must be formulated as the square of the sum of differences between the experimental and predicted data (i.e., $\sum_{i=1}^{ndata}(yexp_i - yteor_i)^2$). Besides, a third output argument must be defined in the objective function: a vector containing those residuals (i.e., $R = [(yexp_1 - yteor_1), (yexp_2 - yteor_2), \ldots, (yexp_{ndata} - yteor_{ndata})]$). In Section 5.5 an application example illustrates the use of these local methods.

[†]For *eSS* this option defines the number of iterations (not the number of evaluations)

[*]This option does not apply to *eSS*

[¶]This option ONLY applies to *eSS*

3.5 Output

SSm and *eSS*'s output is a structure (called **Results** by default) containing the following fields:

- **Results.fbest**: Best objective function value found.
- **Results.xbest**: Vector providing the best function value found.
- **Results.cpu_time**: CPU Time (in seconds) consumed in the optimization.
- **Results.f**: Vector containing the best objective function value after each iteration.
- **Results.x**: Matrix containing the best vector after each iteration.
- **Results.time**: Vector containing the CPU time consumed after each iteration.
- **Results.neval**: Vector containing the number of evaluations after each iteration.
- **Results.numeval**: Total number of function evaluations.
- **Results.local_solutions**: Matrix of local solutions found.
- **Results.local_solutions_values**: Function values of the local solutions.
- **Results.end_crit**: Criterion to finish the optimization:
  - 1: Maximal number of function evaluations achieved.
  - 2: Maximum allowed CPU time achieved.
  - 3: Value to reach achieved.

The structure **Results** as well as **problem** and **opts** are stored and saved in a file called `xxx_report.mat` (where `xxx` is `ssm` or `ess`).

3.6 Guidelines for using *SSm* and *eSS*

Although *SSm*/*eSS* default options have been chosen to be robust for a high number of problems, the tuning of some parameters may help increase the efficiency for a particular problem. Here is presented a list of suggestions for parameter choice depending on the type of problem the user has to face.

- If the problem is likely to be convex, an early local search can find the optimum in short time. For that it is recommended to set the parameter **opts.local.n1 = 0**. Besides, setting **opts.local.n2 = 0** too, the algorithm increases the local search frequency, becoming an "intelligent" multistart.
- When the bounds differ in several orders of magnitude, as is the case of many parameter estimation problems, the *SSm* distance filter based on Euclidean distances might be inefficient. Choosing the tolerances-based distance filter (i.e., **opts.diverse_criteria = 2**) may help explore different parts of the search space. In those cases, the user should rely on a powerful local search to obtain refined solutions. Also, decision variables indexes may be included in **log_var**.

- For problems with discontinuities and/or noise, the local search should either be deactivated or performed by a direct search method. In those cases, activating the option **opts.local.bestx = 1** may help reduce the computation time wasted in useless local searches, performing one every time the search goes into a new basin of attraction.

- When the function values are very high in absolute value, the weight (**opts.weight**) should be increased to be at least 3 orders of magnitude higher than the mean function value of the solutions found.

- When the search space is very big compared to the area in which the global solution may be located, a first investment in diversification may be useful. For that, a high value of **opts.ndiverse** can help finding good initial solutions to create the initial *RefSet*. A preliminary run with aggressive options can locate a set of good initial solutions for a subsequent optimization with more robust settings. This aggressive search can be performed by reducing the size of the *RefSet* (**opts.dim_refset**), setting a high function tolerance for joining the *RefSet* (**opts.tolf**, only for *SSm*), and setting **opts.delete = 'aggressive'** (only for *SSm*). A more robust search is produced increasing the *RefSet* size.

- When the solutions are very scattered and the best solution in *RefSet* (*fbest*) is close to the global optimum but does not improve as fast as we wish, the intensification frequency (**opts.intens**, only for *SSm*) can be increased to create solutions close to the best one.

- If local searches are very time-consuming, their tolerance can be relaxed by reducing the value of **opts.local.tol** not to spend a long time in local solution refinements.

- When there are many local solutions close to the global one, the distance filter for the local search may be deactivated in *SSm*(**opts.local.distance_filter = 0**) or the relaxation should be higher by decreasing **opts.local.maxdistfactor**. For *eSS*, this can be accomplished by setting **opts.local.balance** close to 0.

In order to reduce the complexity associated to the high number of parameters included in *SSm*, three different basic strategies can be chosen by just adjusting the option **opts.strategy**. Setting this option cancels all the option setting that could have previously been made and uses pre-defined sets of options to perform different types of search. **opts.strategy** may have the three following numerical values (Note that choosing this options cancels other possible options previously defined by the user):

1. Efficient and fast search. Recommended for unimodal problems or problems in which we need a fast solution (in terms of CPU time or number of function evaluations).

2. Default *SSm* options. It offers a compromise between intensification and diversification and is recommended for most of the problems.

3. Robust search. Recommended for searching in different areas of the search space. This strategy sacrifices the speed of the convergence to the global optimum by searching in different areas.

3.7 SSmGO Toolbox interfaces: Modelling in Systems Biology

The SSmGO Toolbox can be used as an additional tool in two popular toolboxes for modelling dynamic biological systems: SBToolbox2 and PottersWheel. The SSmGO Toolbox must be installed in your computer to be usable within these toolboxes. More information can be found at `http://www.sbtoolbox2.org/` and `http://www.potterswheel.de/`

4 Extra tools

4.1 *ssm_multistart*

This tool allows the user to perform a multistart optimization procedure with any of the local solvers implemented in the *SSm* toolbox using the same problem declaration as with *SSm*. The script `ssm_multistart` has the same input arguments as `ssm_kernel`.

```
>> Results_multistart=multistart(problem,opts)
```

The structure **problem** has the same fields as in *SSm* (except **problem.vtr** which does not apply here). The structure **opts** has only a few fields compared with *SSm* (i.e., **opts.ndiverse**, **opts.local.solver**, **opts.local.tol** and **opts.local.iterprint**). They all work like in *SSm* except **opts.ndiverse**, which indicates the number of initial points chosen for the multistart procedure. A histogram with the final solutions obtained and their frequency is presented at the end of the procedure.

The output structure **Results_multistart** contains the following fields:

- **.fbest**: Best objective function value found after the multistart optimization.
- **.xbest**: Vector providing the best function value.
- **.x0**: Matrix containing the vectors used for the multistart optimization.
- **.f0**: Vector containing the objective function values of the vectors in **Results_multistart.x0**.
- **.func**: Vector containing the objective function values obtained after every local search.
- **.xxx**: Matrix containing the vectors provided by the local optimizations.
- **.no_conv**: Matrix containing the initial points that did not converge to any solution.
- **.nfuneval**: Matrix containing the number of function evaluations performed in every optimization.

4.2 *ssmgo_test*

This tool allows the user to perform a number of optimizations using *SSm/eSS* for different problems. It is useful to test the performance of these solvers with a problem using different sets of options or to check the same set of options with different problems.

```
>> ssmgo_test(solver,nproblem,noptim,pnames,lb,ub,problem,opts,test,param);
```

The following input parameters must be defined:

- **solver**: String defining the name of the solver we want to test (e.g., 'SSm' or 'eSS')
- **nproblem**: Number of problems to be tested (if we are testing the same problem n times, set `nproblem = n`, not 1).
- **noptim**: Number of optimizations to perform per problem.
- **pames**: Cell array containing the names of the problems as strings.
- **lb**: Cell array containing the lower bounds for all problems.
- **ub**: Cell array containing the upper bounds for all problems.
- **problem**: Matrix and structure containing problem settings for each problem. These settings are declared like in *SSm* but using indexes. For example, if we want to set an initial point for problem 3 we would type `problem(3).x_0=[x_0_1 x_0_2, ..., x_0_n]`.
- **opts**: General options for all problems. They are declared exactly the same way as in *SSm/eSS*.
- **test**: Matrix and structure declaring specific options for individual problems.
- **param**: Structure declaring extra input parameters to be passed to every problem.

ssm_test generates two output .mat files, called **Results_testssmgo_XXX.mat** and **testsummary_XXX.mat** (where XXX is a number related to the date and time when the test was performed), containing the following variables:

- **Results_testssmgo_XXX.mat**: Problem settings, options and results (with the same outputs as in *SSm*) for each run under the format $\mathbf{prob\_p_x}$, $\mathbf{opts\_p_x}$ and $\mathbf{res\_p_x\_r_y}$ respectively, where $\mathbf{x}$ is the problem number and $\mathbf{y}$ is the run number.
- **testsummary_XXX.mat**: Summary of some results:
  - $\mathbf{fbest\_p_x}$: Vector containing the best value found in each run for problem $\mathbf{x}$.
  - $\mathbf{neval\_p_x}$: Vector containing the number of evaluations in each run for problem $\mathbf{x}$.
  - $\mathbf{time\_p_x}$: Vector containing the CPU time consumed in each run for problem $\mathbf{x}$.
  - **best_values**: Vector containing the best function value found for each problem after all the runs.
  - **worst_values**: Vector containing the worst function value found for each problem after all the runs.
  - **mean_values**: Vector containing the mean function value found for each problem after all the runs.

5 Application examples

In this section we will illustrate the usage of *eSS* for solving different instances. If the user want to try *SSm* with this instances, he/she should change the line calling to the solver (i.e., write `ssm_kernel(problem,opts)` instead of `ess_kernel(problem,opts)`).

5.1 Unconstrained problem

$$\min_x f(x) = x_1^2 - 2.1x_1^4 + 1/3x_1^6 + x_1x_2 - 4x_2^2 + 4x_2^4$$

subject to

$$-1 \leq x_1, x_2 \leq 1$$

The objective function is defined in `ex1.m`. Note that being an unconstrained problem, there is only one output argument, f.

ex1.m script

```
function f=ex1(x)
f=4*x(1).*x(1)-2.1*x(1).^4+1/3*x(1).^6+x(1).*x(2)-4*x(2).*x(2)+4*x(2).^4;
return
```

The solver is called in `main_ex1.m`. This problem has two known global optima in $x^* = (0.0898, -0.7127)$ and $x^* = (-0.0898, 0.7127)$ with $f(x^*) = -1.03163$.

Options set:

- Maximum number of function evaluations set to 500.
- Maximum number of initial diverse solutions set to 40.
- Local solver chosen: *dhc*.
- Local solver for final refinement: *fmincon*.
- Show the information provided by local solvers on screen.

main_ex1.m script

```
%======================== PROBLEM SPECIFICATIONS ===========================
problem.f='ex1';                        %mfile containing the objective function
problem.x_L=-1*ones(1,2);               %lower bounds
problem.x_U=ones(1,2);                  %upper bounds

opts.maxeval=500;
opts.ndiverse=40;
opts.local.solver='dhc';
opts.local.finish='fmincon';
opts.local.iterprint=1;
%======================== END OF PROBLEM SPECIFICATIONS ====================

Results=ess_kernel(problem,opts);
```

5.2 Constrained problem

$$\min_x f(x) = -x_1 - x_2$$

subject to

$$\begin{aligned} x_2 &\leq 2x_1^4 - 8x_1^3 + 8x_1^2 + 2 \\ x2 &\leq 4x_1^4 - 32x_1^3 + 88x_1^2 - 96x_1 + 36 \\ 0 &\leq x_1 \leq 3 \\ 0 &\leq x_1 \leq 4 \end{aligned}$$

The objective function is defined in `ex2.m`. Note that being a constrained problem, there are two output argument, f and g.

ex2.m script

```
function [f,g]=ex2(x)
f=-x(1)-x(2);
g(1)=x(2)-2*x(1).^4+8*x(1).^3-8*x(1).^2;
g(2)=x(2)-4*x(1).^4+32*x(1).^3-88*x(1).^2+96*x(1);
return
```

The solver is called in `main_ex2.m`. The global optimum for this problem is located in $x^* = [2.32952, 3.17849]$ with $f(x^*) = -5.50801$.

Options set:

- Maximum number of function evaluations set to 750.
- Increase frequency of local solver calls. The first time the solver is called after 2 iterations. From that moment, the local solver will be called every 3 iterations.

main_ex2.m script

```
%======================== PROBLEM SPECIFICATIONS ===========================
problem.f='ex2';                          %mfile containing the objective function
problem.x_L=[0 0];                        %lower bounds
problem.x_U=[3 4];                        %upper bounds
problem.c_L=[-inf -inf];
problem.c_U=[2 36];

opts.maxeval=750;
opts.local.n1=2;
opts.local.n2=3;
%======================== END OF PROBLEM SPECIFICATIONS ====================

Results=ess_kernel(problem,opts);
```

5.3 Constrained problem with equality constraints

$$\min_x f(x) = -x_4$$

subject to

$$\begin{aligned} x_4 - x_3 + x_2 - x_1 + k_4 x_4 x_6 &= 0 \\ x_1 - 1 + k_1 x_1 x_5 &= 0 \\ x_2 - x_1 + k_2 x_2 x_6 &= 0 \\ x_3 + x_1 - 1 + k_3 x_3 x_5 &= 0 \\ x_5^{0.5} + x_6^{0.5} &\leq 4 \\ 0 \leq x_1, x_2, x_3, x_4 &\leq 1 \\ 0 \leq x_5, x_6 &\leq 16 \end{aligned}$$

with $k_1 = 0.09755988$, $k_3 = 0.0391908$ $k_2 = 0.99k_1$ and $k_4 = 0.9k_3$. The objective function is defined in `ex3.m`. Note that equality constraints must be declared before inequality constraints. Parameters $k_1, \ldots, k_4$ are passed to the objective function through the main script, therefore they do not have to be calculated in every function evaluation. See the input arguments below.

ex3.m script

```
function [f,g]=ex3(x,k1,k2,k3,k4)
f=-x(4);

%Equality constraints
g(1)=x(4)-x(3)+x(2)-x(1)+k4*x(4).*x(6);
g(2)=x(1)-1+k1*x(1).*x(5);
g(3)=x(2)-x(1)+k2*x(2).*x(6);
g(4)=x(3)+x(1)-1+k3*x(3).*x(5);

%Inequality constraint
g(5)=x(5).^0.5+x(6).^0.5;

return
```

The solver is called in `main_ex3.m`. The global optimum for this problem is located in $x^* = [0.77152, 0.516994, 0.204189, 0.388811, 3.0355, 5.0973]$ with $f(x^*) = -0.388811$.

Options set:

- Number of equality constraints set to 4 in `problem.neq`.
- Fields `problem.c_L` and `problem.c_U` only contain bounds for inequality constraints.
- Maximum computation time set to 7 seconds.
- Local solver chosen: *ipopt*.
- Parameters $k_1, \ldots, k_4$ are passed to the main routine as input arguments.

main_ex3.m script

```
%======================== PROBLEM SPECIFICATIONS ===========================
problem.f='ex3';
problem.x_L=[0 0 0 0 0 0];
problem.x_U=[1 1 1 1 16 16];
problem.neq=4;
problem.c_L=-inf;
problem.c_U=4;

opts.maxtime=7;
opts.local.solver='ipopt';

%======================== END OF PROBLEM SPECIFICATIONS =====================
k1=0.09755988;
k3=0.0391908;
k2=0.99*k1;
k4=0.9*k3;

[Results]=ess_kernel(problem,opts,k1,k2,k3,k4);;
```

5.4 Mixed integer problem

$$\min_x f(x) = x_2^2 + x_3^2 + 2x_1^2 + x_4^2 - 5x_2 - 5x_3 - 21x_1 + 7x_4$$

subject to

$$\begin{aligned} x_2^2 + x_3^2 + x_1^2 + x_4^2 + x_2 - x_3 + x_1 - x_4 &\leq 8 \\ x_2^2 + 2x_3^2 + x_1^2 + 2x_4^2 - x_2 - x_4 &\leq 10 \\ 2x_2^2 + x_3^2 + x_1^2 + 2x_2 - x_3 - x_4 &\leq 5 \\ 0 \leq x_i \leq 10 \quad \forall i \in [1, \ldots, 4] & \end{aligned}$$

Integer variables: x_2, x_3 and x_4. In the function declaration (`ex4.m`) they must have the last indexes.

ex4.m script

```
function [f,g]=ex4(x)
f = x(2)^2 + x(3)^2 + 2*x(1)^2 + x(4)^2 - 5*x(2) - 5*x(3) - 21*x(1) + 7*x(4);
g(1) = x(2)^2 + x(3)^2 + x(1)^2 + x(4)^2 + x(2) - x(3) + x(1) - x(4);
g(2) = x(2)^2 + 2*x(3)^2 + x(1)^2 + 2*x(4)^2 - x(2) - x(4);
g(3) = 2*x(2)^2 + x(3)^2 + x(1)^2 + 2*x(2) - x(3) - x(4);
return
```

The solver is called in `main_ex4.m`. The global optimum for this problem is located in $x^* = [2.23607, 0, 1, 0]$ with $f(x^*) = -40.9575$.

Options set:

- An initial point is specified.

- The number of integer variables is specified (mandatory).
- For mixed integer problems the only solver available is *misqp*.
- Stop criterion determined by the CPU time (2 seconds).

main_ex4.m script

```
%======================== PROBLEM SPECIFICATIONS ===========================
problem.f='ex4';
problem.x_L=[0 0 0 0];
problem.x_U=[10 10 10 10];
problem.x_0=[3 4 5 1];
problem.int_var=3;
problem.c_L=[-inf -inf -inf ];
problem.c_U=[8 10 5];

opts.maxtime=2;
opts.local.solver='misqp';

%======================== END OF PROBLEM SPECIFICATIONS ====================
Results=ess_kernel(problem,opts);
```

5.5 Dynamic parameter estimation problem using *n2fb*

Here we will illustrate the use of *SSm* using *n2fb* as local solver. In particular, the problem considered is the isomerization of α-pinene (Rodríguez-Fernández et al., 2006).

$$\min_p J = \sum_{j=1}^{5}\sum_{i=1}^{8}(y_j(p,t_i) - \tilde{y}_{ji})^2$$

subject to the system dynamics

$$\begin{aligned}
\frac{dy_1}{dt} &= -(p_1+p_2)y_1 \\
\frac{dy_2}{dt} &= p_1 y_1 \\
\frac{dy_3}{dt} &= p_2 y_1 - (p_3+p_4)y_3 + p_5 y_5 \\
\frac{dy_4}{dt} &= p_3 y_3 \\
\frac{dy_5}{dt} &= p_4 y_3 - p_5 y_5
\end{aligned}$$

and subject to parameter bounds

$$0 \le x_i \le 1 \quad \forall i \in [1,\ldots,5]$$

In order to use *n2fb* as local solver, in the script `ex5.m` there must be three output arguments: apart from the objective function and the constraints (empty in this case), a vector R containing the squares of the residuals must be defined.

ex5.m script

```
function [J,g,R]=ex5(x,texp,yexp)

[tout,yout] = ode15s(@ex5_dynamics,texp,[100 0 0 0 0],[],x);

R=(yout-yexp);
R=reshape(R,numel(R),1);


J = sum(sum((yout-yexp).^2));
g=0;
return

%************************************************
%Function of the dynamic system
function dy=ex5_dynamics(t,y,p)

dy=zeros(5,1);  %Initialize the state variables


dy(1)=-(p(1)+p(2))*y(1);
dy(2)=p(1)*y(1);
dy(3)=p(2)*y(1)-(p(3)+p(4))*y(3)+p(5)*y(5);
dy(4)=p(3)*y(3);
dy(5)=p(4)*y(3)-p(5)*y(5);

return
%************************************************
```

The solver is called in `main_ex5.m`. The global optimum for this problem is located in $\mathbf{p}^* = [5.93 \cdot 10^{-5}, 2.96 \cdot 10^{-5}, 2.0 \cdot 10^{-5}, 2.75 \cdot 10^{-4}, 4.00 \cdot 10^{-5}]$, with $f(\mathbf{p}^*) = 19.88$.

Options set:

- An initial point is specified.
- All the variables are declared as *log_var*.

5.6 Optimal control problem

Here we will illustrate the use of *SSm* for an optimal control problem. In particular, the problem considered aims to find the optimal temperature profile to maximize the concentration of a product at final time in a batch reactor (example 1 in Luus and Okongwu 1999). The solving methodology is the control vector parameterization (Vassiliadis et al., 1994). In this example, fixed final time and time interval length as well as piecewise linear control approximations are considered.

$$\min_T J = -x_2(t_f)$$

main_ex5.m script

```
%======================== PROBLEM SPECIFICATIONS ==========================
problem.f='ex5';

problem.x_L=zeros(1,5);
problem.x_U=ones(1,5);

problem.x_0=0.5*ones(1,5);

opts.maxeval=1e4;
opts.log_var=[1:5];
opts.local.solver='n2fb';
%======================== END OF PROBLEM SPECIFICATIONS ====================
%time intervals

texp=[0 1230 3060 4920 7800 10680 15030 22620 36420];

% Distribution of species concentration
 %      y(1)    y(2)    y(3)    y(4)    y(5)

 yexp=[ 100.0   0.0   0.0     0.0     0.0
  88.35    7.3     2.3     0.4     1.75
        76.4    15.6     4.5     0.7     2.8
        65.1    23.1     5.3     1.1     5.8
        50.4    32.9     6.0     1.5     9.3
        37.5    42.7     6.0     1.9    12.0
        25.9    49.1     5.9     2.2    17.0
        14.0    57.4     5.1     2.6    21.0
         4.5    63.1     3.8     2.9    25.7 ];

Results=ess_kernel(problem,opts,texp,yexp);
```

subject to the system dynamics

$$\begin{aligned} \frac{dx_1}{dt} &= -k_1 x_1 \\ \frac{dx_2}{dt} &= k_2 x_2 - k_1 x_1 \end{aligned}$$

$$\begin{aligned} k_1 &= 5.35 \cdot 10^{10} \cdot e^{(-9000/T)} \\ k_2 &= 4.61 \cdot 10^{17} \cdot e^{(-15000/T)} \end{aligned}$$

and subject to bounds for the control variable

$$320 \leq T \leq 350$$

The script `ex6.m` shows the objective function to be minimized. Like in the previous example, a numerical integration is needed to solve the systems of differential algebraic equations.

The solver is called in `main_ex6.m`. Options set:

- 10 nodes for the control profile (specific option for optimal control problems).

ex6.m script

```
function J=ex6(u,tfinal,xx,rho)

%Set integration options
tol=1e-7;
options = odeset('RelTol',tol,'AbsTol',tol*ones(1,2));
%Integration
[tout,yout] = ode15s(@dynamic_model,[0 tfinal],[0.95 0.05],[],xx,u',tfinal,rho);

J=-yout(end,2);       %Change sign since we are maximizing

%************************************************************
%Function of the dynamic system
function dy=dynamic_model(t,y,xx,u,tfinal,rho)

dy=zeros(2,1);  %Initialize the state variables

%Linear interpolation of the control variable
T = interp1q(xx,u,t);

k1=5.35e10*exp(-9000/T);
k2=4.61e17*exp(-15000/T);

dy(1)=-k1*y(1);
dy(2)=k1*y(1)-k2*y(2);
%************************************************************
```

- Maximum number of function evaluations set to 2000.
- Local search deactivated.
- Final local refinement with *fminsearch*.

5.7 *ssm_multistart* application

An application of *ssm_multistart* with the problem *ex3* using *solnp* as local solver is presented in the script `main_multistart_ex3.m`. The number of initial points chosen is 25.

5.8 *ssmgo_test* application

This example will perform a test for problems *ex1-ex6* described above. The code for doing this test is presented in the script `main_test.m`

main_ex6.m script

```
%======================== PROBLEM SPECIFICATIONS ========================
rho=10;        %Number of nodes for the control profile
tfinal=30;     %Total operation time (put right units)

xl=320;         %Lower bound for the control
xu=350;         %Upper bound for the control

problem.f='ex6';

%Set the bounds for the optimization problem
problem.x_L=xl*ones(1,rho);
problem.x_U=xu*ones(1,rho);

opts.maxeval=2000;      %Maximum number of evaluations

opts.local.solver=0;
opts.local.finish='fminsearch';

%Set the nodes (equally spaced)
xx=transpose(linspace(0,tfinal,rho));
%======================== END OF PROBLEM SPECIFICATIONS ==================

Results=ess_kernel(problem,opts,tfinal,xx,rho);

%Plot optimal control profile
plot(linspace(0,tfinal,rho),Results.xbest)
xlabel('Process Time (min)')
ylabel('Control profile (K) ')
axis([0 tfinal xl xu])
```

main_multistart_ex3.m script

```
%======================== PROBLEM SPECIFICATIONS ==========================
problem.f='ex3';
problem.x_L=[0 0 0 0 0 0];
problem.x_U=[1 1 1 1 16 16];
problem.neq=4;
problem.c_L=-inf;
problem.c_U=4;

opts.ndiverse=25;

opts.local.solver='solnp';
opts.local.iterprint=1;
opts.local.tol=3;
%==========================================================================
k1=0.09755988;k3=0.0391908;k2=0.99*k1;k4=0.9*k3;

Results_multistart=ssm_multistart(problem,opts,k1,k2,k3,k4);
```

main_test.m script

```
nproblem=6;
noptim=10;                                          %Number of optimization runs per problem
pnames={'ex1', 'ex2', 'ex3', 'ex4','ex5','ex6'};    %Name of all problems
%Parameter for problem 6
rho=10;
lb={-1*ones(1,2), [0 0], [0 0 0 0 0 0], [0 0 0 0], zeros(1,5),320*ones(1,rho)};     %Lower bounds for all problems
ub={ones(1,2), [3 4], [1 1 1 1 16 16], [10 10 10 10], ones(1,5),350*ones(1,rho)};   %Upper bounds for all problems

%Specific problem settings
%Problem 1
problem(1).vtr=-1.031;                              %Value to reach for problem 1
%Problem 2
problem(2).vtr=-5.5;
problem(2).c_L=[-inf -inf];                         %Lower bounds for problem 2 nonlinear inequality constraints
problem(2).c_U=[2 36];                              %Upper bounds for problem 2 nonlinear inequality constraints
%Problem 3
problem(3).neq=4;                                   %Number of nonlinear equality constraints in problem 3
problem(3).c_L=-inf;                                %Lower bounds for problem 3 nonlinear inequality constraints
problem(3).c_U=4;                                   %Upper bounds for problem 3 nonlinear inequality constraints
problem(3).vtr=-0.388;                              %Value to reach for problem 3
%Problem 4
problem(4).x_0=[3 4 5 1];                           %Initial point for problem 4
problem(4).int_var=3;                               %Number of integer variables in problem 4
problem(4).c_L=[-inf -inf -inf ];                   %Lower bounds for problem 4 nonlinear inequality constraints
problem(4).c_U=[8 10 5];                            %Upper bounds for problem 4 nonlinear inequality constraints
problem(4).vtr=-40.95;
%Problem 5
problem(5).vtr=19.9;                                %Value to reach for problem 5
%Problem 6
problem(6).vtr=-0.768;                              %Value to reach for problem 5

%Options for all problems
opts.maxeval=1e3;                                   %Maximum number of function evaluations for all problems
opts.maxtime=5;                                     %Maximum computation time for all problems

%Specific options for some problems
test(3).local.solver='ipopt';                       %Specific local solver for problem 3
test(4).local.solver='misqp';                       %Specific local solver for problem 4
test(5).maxtime=100;                                %Change the optimization time for problem 5
test(5).log_var=[1:5];                              %Declare all variables as log_var for problem 5
test(5).local.solver='n2fb';                        %Specific local solver for problem 5
test(6).maxeval=2000;                               %Change the number of maximum evaluations for problem 6
test(6).maxtime=1e12;                               %Change the optimization time for problem 6

%Extra input parameters for some problems
%Problem 3
k1=0.09755988;
k3=0.0391908;
k2=0.99*k1;
k4=0.9*k3;
param{3}={k1,k2,k3,k4};

%Problem5
%time intervals
t=[0.0 1230.0 3060.0 4920.0 7800.0 10680.0 15030.0 22620.0 36420.0];

% Distribution of species concentration
 %      y(1)    y(2)    y(3)    y(4)    y(5)
 yexp=[ 100.0  0.0  0.0  0.0     0.0
     88.35    7.3     2.3     0.4     1.75
        76.4    15.6     4.5     0.7     2.8
        65.1    23.1     5.3     1.1     5.8
        50.4    32.9     6.0     1.5     9.3
        37.5    42.7     6.0     1.9    12.0
        25.9    49.1     5.9     2.2    17.0
        14.0    57.4     5.1     2.6    21.0
         4.5    63.1     3.8     2.9    25.7 ];
param{5}={t,yexp};

%Problem 6 needed parameters
tfinal=30;
xx=transpose(linspace(0,tfinal,rho));
param{6}={tfinal,xx,rho};

%Call ssmgo_test
ssmgo_test('ess',nproblem,noptim,pnames,lb,ub,problem,opts,test,param);
```

Appendix A: List of options for SSm

Option	Description	Type	Options	Default value
		User options		
opts.maxeval	Maximum number of function evaluations	Integer	–	1000
opts.maxtime	Maximum CPU time in seconds	Integer	–	60
opts.iterprint	Print information on the screen after each iteration	Binary	[0|1]	1
opts.plot	Plots convergence curves	Integer	[0|1|2]	0
opts.weight	Weight for penalizing infeasible solutions	Real [Positive]	–	10^6
opts.log_var	Indexes of "logarithmic" variables	Real [vector positive]	–	[ ]
opts.tolc	Tolerance for local search and constraints violation	Real [Positive]	–	10^{-5}
opts.prob_bound	Probability (0-1) of biasing the search toward the bounds	Real [Positive]	–	0.5
opts.strategy	Search strategy	Integer	[0|1|2|3]	0
opts.inter_save	Saves results in a .mat file in intermediate iterations	Binary	[0|1]	0
		Global options		
opts.dim_refset	Number of elements in *RefSet*	Integer ≥ 6	–	'auto'
opts.ndiverse	Number initial diverse solutions	Integer	–	$10 \cdot nvar$
opts.initiate	Type of *RefSet* initialization	Binary	[0|1]	1
opts.combination	Type of combination of *RefSet* elements	Integer	[1|2]	1
opts.regenerate	Type of *RefSet* regeneration	Integer	[1|2|3]	3
opts.delete	Number of *RefSet* elements deleted in the regeneration step	String	['standard'|'aggressive']	'standard'
opts.intens	Iteration interval between intensifications	Integer	–	10
opts.tolf	Function tolerance for joining the *RefSet*	Real [Positive]	–	10^{-4}
opts.diverse_criteria	Diversification criterion in the *RefSet*	Integer	[1—2]	1
opts.tolx	Variable tolerance for joining the *RefSet*	Real [Positive]	–	10^{-3}
		Local options		
opts.local.solver	Local solver	String	see section 3.4	'fmincon'
opts.local.tol	Level of tolerance in local search	Integer [Positive]	[1|2|3]	2
opts.local.iterprint	Print each iteration of local solver on screen	Binary	[0|1]	0

Option	Description	Type	Options	Default value
Local options (continued)				
opts.local.n1	Number of evaluations before the 1st local search	Integer	–	$100 \cdot nvar$
opts.local.n2	Number of evaluations between 2 local searches	Integer	–	$200 \cdot nvar$
opts.local.finish	Local solver for final search	String	see section 3.4	opts.local.solver
opts.local.bestx	Applies the local search only when *fbest* is improved	Binary	[0|1]	1
opts.local.merit_filter	Activation of merit filter for local search	Binary	[0|1]	1
opts.local.distance_filter	Activation of distance filter for local search	Binary	[0|1]	1
opts.local.thfactor	Merit filter relaxation parameter	Real [Positive]	–	0.2
opts.local.maxdistfactor	Distance filter relaxation parameter	Real [Positive]	–	0.2
opts.local.wait_th_limit	Failed evaluations before applying merit filter relaxation	Integer [Positive]	–	20
opts.local.wait_maxdist	Failed evaluations before applying distance filter relaxation	Integer [Positive]	–	20

Appendix B: List of options for eSS

Option	Description	Type	Options	Default value
		User options		
opts.maxeval	Maximum number of function evaluations	Integer	–	1000
opts.maxtime	Maximum CPU time in seconds	Integer	–	60
opts.iterprint	Print information on the screen after each iteration	Binary	[0|1]	1
opts.plot	Plots convergence curves	Integer	[0|1|2]	0
opts.weight	Weight for penalizing infeasible solutions	Real [Positive]	–	10^6
opts.log_var	Indexes of "logarithmic" variables	Real [vector positive]	–	[ ]
opts.tolc	Tolerance for local search and constraints violation	Real [Positive]	–	10^{-5}
opts.prob_bound	Probability (0-1) of biasing the search toward the bounds	Real [Positive]	–	0.5
opts.inter_save	Saves results in a .mat file in intermediate iterations	Binary	[0|1]	0
		Global options		
opts.dim_refset	Number of elements in *RefSet*	Integer ≥ 6	–	'auto'
opts.ndiverse	Number initial diverse solutions	Integer	–	$10 \cdot nvar$
opts.combination	Type of combination of *RefSet* elements	Integer	[1|2]	1
		Local options		
opts.local.solver	Local solver	String	see section 3.4	'fmincon'
opts.local.tol	Level of tolerance in local search	Integer [Positive]	[1|2|3]	2
opts.local.iterprint	Print each iteration of local solver on screen	Binary	[0|1]	0
opts.local.n1	Number of iterations before the 1st local search	Integer	–	1
opts.local.n2	Number of iterations between 2 local searches	Integer	–	10
opts.local.finish	Local solver for final search	String	see section 3.4	opts.local.solver
opts.local.bestx	Applies the local search only when *fbest* is improved	Binary	[0|1]	1
opts.local.balance	Balances between quality (= 0) and diversity (= 1)	Real [Positive]	–	0.5

References

de la Maza, M. and Yuret, D. (1994). Dynamic hill climbing. *AI Expert*, 9(3):26–31.

Dennis, J. E., Gay, D. M., and Welsch, R. E. (1981). An adaptive non-linear least-squares algorithm. *ACM Transactions on Mathematical Software*, 7(3):348–368.

Egea, J., Balsa-Canto, E., García, M.-S., and Banga, J. (2009). Dynamic optimization of nonlinear processes with an enhanced scatter search method. *Industrial & Engineering Chemistry Research*, 48(9):4388–4401.

Egea, J., Martí, R., and Banga, J. (2010). An evolutionary algorithm for complex process optimization. *Computers & Operations Research*, 37(2):315–324.

Egea, J. A. (2008). *New Heuristics for Global Optimization of Complex Bioprocesses*. PhD thesis, University of Vigo, Vigo (Spain).

Egea, J. A., Rodríguez-Fernández, M., Banga, J. R., and Martí, R. (2007). Scatter search for chemical and bio-process optimization. *Journal of Global Optimization*, 37(3):481–503.

Exler, O. and Schittkowski, K. (2007). A trust region sqp algorithm for mixed-integer nonlinear programming. *Optimization Letters*, 1(3):269–280.

Hooke, R. and Jeeves, T. (1961). Direct search solution of numerical and statistical problems. *Journal of the ACM*, 8(2):212–229.

Laguna, M. and Martí, R. (2003). *Scatter Search: Methodology and Implementations in C*. Kluwer Academic Publishers, Boston.

Luus, R. and Okongwu, O. N. (1999). Towards practical optimal control of batch reactors. *Chemical Engineering Journal*, 75(1):1–9.

Rodríguez-Fernández, M., Egea, J. A., and Banga, J. R. (2006). Novel metaheuristic for parameter estimation in nonlinear dynamic biological systems. *BMC Bioinformatics*, 7:483+.

The MathWorks^{TM} (2008). Optimization toolbox 4 user's guide.

Vassiliadis, V. S., Sargent, R. W. H., and Pantelides, C. C. (1994). Solution of a class of multistage dynamic optimization problems. 1. problems without path constraints. *Industrial and Engineering Chemistry Research*, 33(9):2111–2122.

Wächter, A. and Biegler, L. (2006). On the implementation of an interior point filter line search algorithm for large-scale nonlinear programming. *Mathematical Programming*, 106(1):25–57.

Ye, Y. (1987). *Interior algorithms for linear, quadratic and linearly constrained non-linear programming*. PhD thesis, Stanford University.