MEMPSODE-1.0

User Manual

C. Voglis, K.E. Parsopoulos
Department of Computer Science, University of Ioannina

D.G. Papageorgiou
Department of Materials Science and Engineering, University of Ioannina

I.E. Lagaris
Department of Computer Science, University of Ioannina

M.N. Vrahatis
Department of Mathematics, University of Patras

OPTIMA Research Group
http://optima.cs.uoi.gr

GREECE

Contents

1 Conventions

a) Lower case boldface letters ($\boldsymbol{x}$, $\boldsymbol{g}$, etc) stand for vectors in the $N-$dimensional space.

b) MEMPSODE options, C code and I/O data are printed using monospaced font

c) n: the number of parameters (dimensionality of the problem)

d) N: the size of the swarm (number of individuals)

e) $f(x)$: standard form of the objective function

f) l_i, u_i: lower and upper bounds for parameter x_i

2 Software distribution

The software is distributed as a `tar.gz` file. It can be uncompressed and extracted by issuing the following commands in a shell window:

```
gunzip mempsode-1.0.tar.gz
tar -xvf mempsode-1.0.tar
```

A directory named `mempsode-1.0` will be created that contains two folders `src` and `testrun` and the present manual. The following files are contained in `src/` folder:

1. `Makefile`: The input file for the `make` utility.
2. `SOURCE_main.c`, `SOURCE_main.h`: Source code file containing the `main` program, which parses arguments and prepares input/output files.
3. `SOURCE_UPSO_DE.c`, `SOURCE_UPSO_DE.h`: Source code file that implements hybrid UPSO and DE.
4. `SOURCE_local_opt.c`, `SOURCE_local_opt.h`: Implements the interface to the Merlin Optimization Environment.
5. `SOURCE_StopCriterion.c`: Contains the stopping rules that control the termination of the algorithms.
6. `SOURCE_print.c`, `SOURCE_LinAlg.c`: Various printing and Linear Algebra utility functions.
7. `SOURCE_problems.c`, `SOURCE_problems.h`, `data.h`: Implements 32 selected objective functions.
8. `SOURCE_params.h`: Defines appropriate variable structures for the problem, algorithm and experiment statistics.
9. `objective.f`: `FORTRAN` wrapper that enables Merlin to operate on the user's objective function.
10. `rastrigin.c`: Rastrigin objective function.
11. `lj.c`: Lennard–Jones potential objective function.

12. `tinker.c`: Objective function wrapper to the Tinker package.
13. `vp8.int` : Describes a sample protein (ACE-ALA_8-NME) in internal coordinates.
14. `input.tinker`: Input file for use with Tinker package.
15. `in.dat`: Sample file containing Merlin instructions for local minimization.
16. `ifunmind.f`: A FORTRAN wrapper to the Tinker package. Called from `tinker.c`.
17. `chemin.f`: Contains a utility subroutine that calculates the root mean square gradient.

The following files are contained in `testrun1/` folder:

1. `run.sh` : A shell script that executes the test run.
2. `TestRunOutput.txt` : A sample output for comparison.

The following files are contained in `testrun2/` folder:

1. `run.sh` : A shell script that executes the test run.
2. `fit.c` : A C source file that contains the fitting objective function.
3. `sample_main.c` : A sample main C source file.
4. `data.txt` : Data for the fitting.
5. `in.dat`: Sample file containing Merlin instructions for local minimization.
6. `PDESC`: Merlin's panel description file.
7. `TestRunOutput.txt` : A sample output for comparison.

3 Configuring and building

Prior to building the MEMPSODE software, the user must provide some system–related information. This information includes the path of the Merlin software as well as the `C` and `FORTRAN` compilers that will be used for the compilation of the source code. This information has to be provided in the `Makefile`. The first lines of a `Makefile` are listed bellow:

```
#-------------------------------------------------------------
# System dependent compilers, programs and directories
#-------------------------------------------------------------
#
# Merlin distribution
#-------------------------------------------------------------
MERLIN=/usr/local/merlin
#
# Compilers, flags
#-------------------------------------------------------------
CC=gcc
F77=gfortran
```

```
LDFLAGS=-lgfortran
#F77=g77
#LDFLAGS=-lg2c
CFLAGS=-O3
FFLAGS=-O3
NM=nm -u
```

The user can edit `Makefile` to properly parametrize the software. A brief explanation of the configuration parameters is given below:

(1) `MERLIN`: The user provides Merlin's complete installation path.

(2) `CC`: It specifies the name of the `C` compiler. In systems supplied with the GNU C compiler, `CC` must be set to `gcc`.

(3) `F77`: It specifies the `FORTRAN` compiler. In systems supplied with the GNU Compiler Collection v4.0 or newer, this must be set to `gfortran`. In older distributions, the compiler is `g77`.

(4) `LDFLAGS`: This parameter is used to load the appropriate libraries to combine `C` and `FORTRAN` source code. If GNU Compiler Collection v4.0 or newer is used, this parameter must be set to `-lgfortran`. In older distributions, it must be set to `-lg2c`.

(5) `CFLAGS`, `FFLAGS`: The default values for these parameters set the optimization level for both `C` and `FORTRAN` compilers to 3.

(6) `NM`: It specifies the name/path for the `nm` utility.

3.1 Building the executable

After specifying the system–related parameters, the `make` utility is used to configure and build the MEMPSODE software. The basic syntax is:

```
make OBJECTIVE=<filename>
```

The `make` utility builds the software using the objective function contained in the `<filename>.c` source code file and creates the executable program `mempsode`. For example the commands:

```
make OBJECTIVE=SOURCE_problems
make OBJECTIVE=rastrigin
```

will create executable files for the objective functions defined in `SOURCE_problems.c` and `rastrigin.c`, respectively.

For the solution of protein conformation problems, the Tinker's potential energy function shall be used. For this purpose, the objective function `tinker.c` must be used and the `TINKER` variable needs to be set with Tinker's source code directory. For example:

```
make OBJECTIVE=tinker TINKER=/home/user/tinker/source
```

The software can also be configured to ignore the Merlin optimization environment if no local optimization is desirable. In this case, the software reduces to a classical UPSO or DE implementation. To set this, the `NOMERLIN` variable must be set to 1:

```
make OBJECTIVE=SOURCE_problems NOMERLIN=1
```

Simply issuing the command:

```
make
```

builds the software using the objective functions from `SOURCE_problems.c`.

3.2 Building the library `libmemspode.a`

To integrate MEMPSODE to an existing application we provide a user callable interface which is delivered to the user via the `libmemspode.a` library. The user can build the basic MEMPSODE library including Merlin support by issuing the following command:

```
make lib
```

To disable Merlin the user can build the library by issuing:

```
make lib NOMERLIN=1
```

To include Tinker routines in the library one must issue:

```
make lib TINKER=/path/to/tinker/source
```

Upon successful compilation the file `libmemspode.a` will be created.

4 Routines supplied by the user

We provide a template of objective function written in the `C` programming language in Listing 5. The code from line 6 to line 26 defines the Merlin wrapper for the user–defined objective function and must be kept unchanged. The user must provide the following subprograms:

a) `void Objective_F (double x[], int n, double *f)`
Returns the value of the objective function evaluated at x.

`x`	(input)	Array containing the point at which the calculation is desired.
`n`	(input)	The dimensionality of the function.
`f`	(output)	Objective function value.

b) `void Bounds_F(double l[], double r[], int n)`
Returns the double–precision array `l` with the lower bounds and the double–precision array `r` with the upper bounds of the variables.

`n`	(input)	The dimensionality of the function.
`l`	(output)	Array containing the lower bounds.
`r`	(output)	Array containing the upper bounds.

If the first order derivatives are analytically known, they can be provided by the following optional subprogram.

c) `void Objective_G (double x[], int n, double g[])`
Returns the gradient vector evaluated at x.

`x`	(input)	Array containing the point at which the calculation is desired.
`n`	(input)	The dimensionality of the function.
`g`	(output)	Gradient vector.

Listing 1: Objective function template.

```
/*
 * ============================================================
                    Begin: Do not change
 * ============================================================
 */
 extern struct{
     int ipro;
}funcid_;

void Objective_F (double X[], int N, double *F);
void objwrap_ (double *X, int *N, double *F, int *problem)
{
        int NN;
        NN = *N;
        funcid_.ipro = *problem;
        Objective_F (X, NN, F);
}
void Objective_G (double X[], int N, double G[]);
void gradwrap_ (double *X, int *N, double *G, int *problem)
{
        int NN;
        NN = *N;
        funcid_.ipro = *problem;
        Objective_G (X, NN, G);
}
/*
 * ============================================================
                    End: Do not change
 * ============================================================
 */

void Objective_F (double X[], int N, double *F) { ... }

void Bounds_F(double *xl, double *xr, int N) { ... }

void Objective_G (double X[], int N, double G[]) { ... }
```

The user is allowed to include more than one objective function in a single file. In this case, the parameter `funcid_.ipro` of the structure `funcid_` in Listing 5, is used to distinguish between the different problems. The software provides specific command line option that allows users to specify which objective function shall be used (`-o`). An example code that uses `funcid_` inside the `Objective_F` function to distinguish between three objective functions is shown in Listing 2. Similar format must be used also in functions `Bounds_F` and `Objective_G`.

Listing 2: Multiple objective functions in a single file.

```
...
// SPHERE
```

```
if (funcid_.ipro == 0) {
    *F = 0.0;
    for (i=0; i<N; i++)
        *F = *F + pow(X[i],2);
}

// ROSENBROCK
else if (funcid_.ipro == 1) {
    *F = 0.0;
    for (i=0; i<N-1; i++)
        *F = *F + 100.0*pow((X[i+1]-X[i]*X[i]),2)
                + pow((X[i]-1.0),2);
}

// RASTRIGIN
else if (funcid_.ipro == 2) {
    *F = 0.0;
    for (i=0; i<N; i++)
        *F = *F + pow(X[i],2) + 10.0
                - 10.0*cos(2*pi*X[i]);
} ...
```

If the user wishes to run MEMPSODE with LS then Merlin must be instructed to execute a local optimization method. For this purpose, a file that contains Merlin instructions or the object code of an MCL program. The user can edit this file to change the local optimization algorithm at will. The software distribution provides the following sample `in.dat` file:

Listing 3: Sample Merlin instructions file

```
bfgs noc 500 gtol 0.01
```

This file instructs Merlin to use the BFGS local optimization algorithm for a maximum of 500 function evaluations (per local search) and set the gradient termination criterion to 0.01. We present a brief list of Merlin optimization commands and a subset of their parameters in Table 1. Also a list of Merlin control commands that affect the optimization procedures is displayed in Table 2.

A complete list of the available local optimization algorithms and their options is provided in Merlin's user manual [9].

5 User callable interface routines

We provide a complete set of user callable interface routines that enable the programmer to fully control MEMPSODE's behavior. These routines are listed below and are divided into the *core routines* and *set-get routines*

5.1 Core routines

a) `init_mempsode()`
   Initializes the default parameters.

b) `mempsode()`
   Calls the optimizer.

5.2 Set routines

a) `void set_mempsode_iparam(char *name, int val)`
Sets integer MEMPSODE parameters. The names and the allowed values correspond to the long version of command line arguments presented in Table 3.

`name` (input) The name of the parameter to be set.

- `"algorithm"`: Switch between uPSO and DE.
- `"objective"`: Set the objective function id.
- `"dimension"`: Set the dimensionality of the problem.
- `"swarm-size"`: Set the swarm size.
- `"max-fun-evals"`: Set the maximum number of function evaluations.
- `"max-grad-evals"`: Set the maximum number of gradient evaluations.
- `"max-iter"`: Set the maximum number of iterations.
- `"radius"`: Set the radius of the local uPSO neighborhood.
- `"seed"`: Set the seed for random number generators.
- `"memetic"`: Set the memetic strategy.
- `"dump-iterations"`: Set the number
- `"pso-params"`: Switch between two sets of PSO paramters.
- `"use-mutation"`: Enable/disable the usage of mutation.
- `"de-strategy"`: Set the DE strategy.
- `"file-output"`: Enable/disable output to file.

`val` (input) The corresponding value to be set.

b) `void set_mempsode_dparam(char *name, double val)`
Sets double precision MEMPSODE parameters. The names and the allowed values correspond to the long version of command line parameters presented in Table 3.

`name` (input) The name of the parameter to be set.

- `"target-value"`: Set the target value to be reached.
- `"unification-factor"`: Set the unification factor for uPSO.
- `"mutation-mean"`: Set the mean value for the mutation.
- `"mutation-std"`: Set the standard deviation for the mutation.
- `"F-param"`: Set the F parameter for the DE algorithm.
- `"CR-param"`: Set the CR parameter for the DE algorithm.
- `"velocity-scaling"`: Set the scaling quantity for initializing velocity.
- `"prob-local-search"`: Set the probability to start a local search.
- `"EPS"`: Specifies a small positive number.
- `"XEPS"`: Set the x-tolerance theshold.
- `"FEPS"`: Set the function value tolerance theshold.

- `"GEPS"`: Set the gradient tolerance theshold.

`val` (input) The corresponding value to be set.

c) `void set_mempsode_cparam(char *name, char *val)`
Sets character values MEMPSODE parameters.

`name` (input) The name of the parameter to be set.

- `"algorithm"`: Switch between uPSO and DE.
- `"infile"`: Set the Merlin input file.
- `"outfile"`: Set the Merlin output file.

`val` (input) The corresponding value to be set.

d) `void set_mempsode_bounds(char *name, double *array)`
Sets the upper/lower bounds.

`name` (input) A string describing which bounds are to be affected.

- `"xmin"`: Set the lower bounds
- `"xmax"`: Set the upper bounds

`array` (input) Array containing the bounds.

5.3 Get routines

a) `void get_mempsode_iparam(char *name, int *val)`
Returns integer MEMPSODE parameters.

`name` (input) The name of the parameter to be returned.

`val` (output) Pointer to store the requested value.

b) `void get_mempsode_dparam(char *name, double *val)`
Returns double precision MEMPSODE parameters.

`name` (input) The name of the parameter to be returned.

`val` (output) Pointer to store the requested value.

c) `void get_mempsode_cparam(char *name, char *val)`
Returns character values MEMPSODE parameters.

`name` (input) The name of the parameter to be returned.

`val` (output) Pointer to store the requested value.

d) `get_mempsode_min(char *name, double *val)`
Returns the best objective function value and the best point.

`name` (input) A string describing which information is to be returned.

- `"minval"`: Return the best objective function value
- `"minx"`: Return the best point (minimizer)

`val` (input) Pointer to hold the results.

e) `get_mempsode_stats(char *name, int *val)`

Returns various statistics counters.

`name`	(input)	A string describing which information is to be returned. • `"iter"`: Return the number of iterations. • `"fevals"`: Return the number of function evaluations. • `"gevals"`: Return the number of gradient evaluations. • `"local"`: Return the number of local searches. • `"bpup"`: Return the number of best position updates. • `"lasthit"`: Return the last iteration in which the target value was reached. • `"lastf"`: Return the number of function evaluations spent when the target value was reached. • `"lastg"`: Return the number of gradient evaluations spent when the target value was reached. • `"lastl"`: Return the number of local searches applied when the target value was reached.
`val`	(input)	Pointer to hold the result.

f) `void get_mempsode_bounds(char *name, double *array)`

Returns the upper/lower bounds.

`name`	(input)	A string describing which bounds are to be returned.
`array`	(output)	Array to store the corresponding bounds.

6 Running the executable

After the successful compilation, the software can be executed using the command `mempsode` in the command prompt. A large number of options are available to configure the execution and their use is similar to passing arguments in unix/linux commands. A complete list of these options along with their default values is provided in Table 3.

A simple running example with minimum options looks as follows:

```
make OBJECTIVE=sample_objective
mempsode -a pso -d 20 -s 100 -f 10000
```

The first command compiles MEMPSODE for use with the objective function provided in the file `sample_objective.c`. The second command executes MEMPSODE using the UPSO algorithm (option `-a pso`) on the 20–dimensional instance of the provided objective function (option `-d 20`), using swarm size 100 and maximum number of function evaluations 10000 (options `-s 100` and `-f 10000`, respectively). All other necessary options are automatically set to their default values reported in Table 3. More comprehensive examples are presented in the following sections.

6.1 Stopping criteria

Execution of the software terminates when any one of the following criteria is met.

a) The number of function evaluations performed so far (calls to the user supplied routine `Objective_F`) exceeds a user defined upper bound which is specified using the `-f` command line option.

b) The number of gradient evaluations performed so far (calls to the user supplied routine `Objective_G`) exceeds a user defined upper bound which is specified using the `-g` command line option.

c) The number of UPSO or DE cycles exceeds a user defined upper bound, which is specified using the `-i` command line option.

d) The value of the objective function reaches or falls below a user defined target value, which is specified using the `-t` command line option.

If any one of the `-g`, `-i` or `-t` options is not supplied then the corresponding termination criterion is not used. In case the `-f` option is not specified then a upper bound of $100\,000\,n$ function evaluations is assumed.

6.2 Execution output

MEMPSODE produces 4 output files containing execution results, unless the user sets a no–print command line option. The Merlin environment redirects its output received for the execution of the commands provided in `in.dat` to a file named `out.dat`. This output file is overwritten each time an LS procedure is evoked.

MEMPSODE output files adhere to the following naming conventions:

UPSO output files

(a) `Default_Alg<X>_Fun<Y>_Dim<N>_Swarm<S>_Mem<M>_Fmax<F>_UF<U>_R3use<R>_rep`

(b) `Default_Alg<X>_Fun<Y>_Dim<N>_Swarm<S>_Mem<M>_Fmax<F>_UF<U>_R3use<R>_sol`

(c) `Default_Alg<X>_Fun<Y>_Dim<N>_Swarm<S>_Mem<M>_Fmax<F>_UF<U>_R3use<R>_tmp`

(d) `Default_Alg<X>_Fun<Y>_Dim<N>_Swarm<S>_Mem<M>_Fmax<F>_UF<U>_R3use<R>_ver`

DE output files

(a) `Default_Alg<X>_Fun<Y>_Dim<N>_Swarm<S>_Mem<M>_Fmax<F>_OP<P>_rep`

(b) `Default_Alg<X>_Fun<Y>_Dim<N>_Swarm<S>_Mem<M>_Fmax<F>_OP<P>_sol`

(c) `Default_Alg<X>_Fun<Y>_Dim<N>_Swarm<S>_Mem<M>_Fmax<F>_OP<P>_tmp`

(d) `Default_Alg<X>_Fun<Y>_Dim<N>_Swarm<S>_Mem<M>_Fmax<F>_OP<P>_ver`

where,

`<X>` is 1 for UPSO or 2 for DE.

`<Y>` is the identification number of the objective function (if there are several objective functions in a single file).

`<N>` is the problem's dimension.

`<S>` is the swarm/population size.

`<M>` is 0 for no local search or 1, 2 or 3 for the corresponding memetic schemes as they are reported in the paper.

`<F>` is the maximum number of function evaluations.

`<U>` is the unification factor (only in UPSO).

`<R>` is 0 if mutation is used or 1 otherwise (only in UPSO).

`<P>` ranges from 1 to 5, denoting the corresponding DE operator (only in DE).

For both UPSO and DE, file (a) contains the final output (function value, number of function evaluations, number of local searches) for one run (experiment) with the specific set of parameters (algorithm options). The user can use the -e option in order to define the number of independent runs for the specific set of parameters. File (b) contains the final solution reached from each experiment. File (c) contains temporary results (iteration, function value reached and function evaluations). File (d) contains the values of all parameters that control the software. The prefix `Default` of the filenames can change to any desirable string by using the `-O` option (see Table 3).

6.3 Execution examples

MEMPSODE command line options can be entered in two formats. The *short format* uses only one character to specify the option, followed by the desirable value. The examples presented so far have adopted this notation. On the other hand, the *long format* uses complete words to specify the options. Table 3 reports both formats for each option. For example, the short format command:

`mempsode`	`-a pso`	`-d 20`	`-s 100`	`-f 10000`	`-l 2`
	choice between UPSO and DE	problem's dimension	swarm size	maximum function evaluations	memetic strategy

which executes MEMPSODE using UPSO (option `-a pso`) on the 20–dimensional instance of the provided objective function (option `-d 20`), using swarm size 100 (option `-s100`), maximum number of function evaluations 10000 (option `-f 10000`) and memetic strategie 2 (option `-l 2`), can be equivalently written in long format as:

```
mempsode --algorithm=pso --dimension=20 --swarm-size=100  ...
         --max-fun-evals=10000 --memetic=2
```

Below, we present six execution examples using objective functions provided in the file `SOURCE_problems.c`. Initially, the user shall build MEMPSODE as follows:

```
make OBJECTIVE=SOURCE_problems
```

In our examples, we will assume that the problem under consideration is the Rastrigin function, which has the identification number 2 in `SOURCE_problems.c`.

6.3.1 Example 1

If the plain UPSO (no local search) with unification factor 0.5 shall be applied on the 20–dimensional Rastrigin test function, using a swarm size of 50, maximum number of function

evaluations 100000 and initial velocity scaling 0.1, then the following command shall be used:

```
mempsode -a pso -o 2 -d 20 -s 50 -f 100000 -u 0.5 -c 0.1 -l 0
```

6.3.2 Example 2

If the OP1 operator (see paper) of the DE algorithm shall be applied on the 20–dimensional Rastrigin test function, using the second memetic strategy with a population of size 50, maximum number of function evaluations 100000, and 10 independent experiments, then the following command shall be used:

```
mempsode -a de -v 1 -o 2 -d 20 -s 50 -f 100000 -l 2 -e 10
```

6.3.3 Example 3

Considering the scenario where the standard lbest PSO without LS shall be applied on the 30–dimensional Rastrigin test function, using a swarm of 70 particles for a maximum of 100000 iterations. Also, assume that, upon successful termination, the particles, best positions and velocities shall be saved into the file `psode_state.out`. Then the following command can be used:

```
mempsode -a pso -o 2 -d 30 -s 70 -f 100000 -W psode_state.out
```

Notice that in this scenario, the default unification factor 1 and no local search will be used, since the corresponding parameters are not defined by the user.

6.3.4 Example 4

Same as Example 3, with the only difference that initial particles, best positions and velocities are loaded from the file `psode_state.out`:

```
mempsode -a pso -o 2 -d 30 -s 70 -f 100000 -R psode_state.out ...
          -W psode_state.out
```

6.3.5 Example 5

In this example we apply a hybrid UPSO with Scheme 3, swarm size 70 and unification factor 0.1. The execution is terminated until maximum number of iteration reaches 1000 or maximum function evaluation reaches 4000000 or when the target value of 0.0 is reached by the default accuracy.

```
mempsode -a pso -o 3 -u 0.1 -d 30 -s 70 -i 1000 -f 4000000 -t 0.0
```

6.3.6 Example 6

In this example we apply a hybrid DE with Scheme 2, swarm size 100. We will use the `rastrigin.c` source code because it implements `Objective_G`.

```
make OBJECTIVE=rastrigin
```

File `myin.dat` is used to define the local search applied. The contents of this file are:

Listing 4: File `myin.dat`

```
ana
bfgs noc 1000
```

This instruct Merlin to use the gradient subroutine `Objective_G`. The execution until the target value of 0.0 is reached by 0.01. Also a local minimum is identified (after a local search) if the gradient norm falls bellow 0.0001.

```
mempsode -a de -o 2 -d 10 -s 100 -t 0.0 -Y 0.01 -G 0.001 -y myin.dat
```

The user is encouraged to compare the options in the examples above with those in Table 3.

7 Linking with the library

Suppose that we have successfully compiled MEMPSODE library in the directory `/path/mempsode`.
The user must provide a main program (`sample_main.c`) and the objective function (`sample_objective.c`) following the guidelines of Section 4.

Listing 5: A sample main file.

```
#include "mempsode.h"

int main(int argc, char *argv)
{
    double val;
    init_mempsode();
    set_mempsode_iparam("algorithm", 1);            /* Set algorithm uPSO*/
    set_mempsode_iparam("dimension", 10);           /* Set dimension to 10*/
    set_mempsode_iparam("swarm-size", 50);          /* Set swarm size to 50*/
    set_mempsode_iparam("max-fun-evals", 10000);    /* Set maximum number of
                                                       function evals to 10000*/

    mempsode();                                     /* Run mempsode */

    get_mempsode_min("minval", &val);               /* Return the minimum value*/
    printf("Minimum value -->%f \n", val);
}
```

The default linking command line that enables Merlin and disables Tinker support is:

```
gcc sample_main.c sample_objective.c -lmempsode -L/path/mempsode
    -I/path/mempsode -lgfortran
```

If the gradient code is provided in `sample_objective.c` then the command line to enable its usage should be:

```
gcc sample_main.c sample_objective.c -lmempsode -L/path/mempsode
    -I/path/mempsode -lgfortran -DGRAD=1
```

To disable Merlin the user can issue:

```
gcc sample_main.c sample_objective.c -lmempsode -L/path/mempsode
    -I/path/mempsode -lgfortran -DNOMERLIN=1
```

To enable Tinker objective function the user issues:

```
gcc sample_main.c -lmempsode -L/path/mempsode -I/path/mempsode -lgfortran
                  -DTINKER=1 -DGRAD=1 -L/path/to/tinker/soure -ltinker
```

Notice that file `tinker.c` is now used.

8 Sample applications

We provide four sample applications to illustrate the use of MEMPSODE and demonstrate its applicability in various types of problems. In the first three MEMPSODE solving capabilities are explored. In the fourth demo we provide a shell script for multiple invocations of MEMPSODE suitable for comparative experiments.

8.1 The Rastrigin objective function

The Rastrigin function is well established in the assessment of global optimization methods. It has the form:

$$f(x) = 10\,n + \sum_{i=1}^{n} \left(x_i^2 - 10\cos(2\pi x_i)\right), \tag{1}$$

where $x_i \in [-5.12, 5.12]$, $i = 1, 2, \ldots, n$. This function has a global minimizer, $x^* = (0, 0, \ldots, 0)^\top$, with value $f(x^*) = 0$. In the software distribution the Rastrigin function is coded in `rastrigin.c`.

Listing 6: File: rastrigin.c.

```
#include <math.h>
double  pi=3.14159265358979;
/*
                     Begin: Do not change   */
 extern struct{
     int ipro;
}funcid_;

void Objective_F (double X[], int N, double *F);
void objwrap_ (double *X, int *N, double *F, int *problem){
        int NN;
        NN = *N;
        funcid_.ipro = *problem;
        Objective_F (X, NN, F);
}
void Objective_G (double X[], int N, double G[]);
void gradwrap_ (double *X, int *N, double *G, int *problem){
        int NN;
        NN = *N;
        funcid_.ipro = *problem;
        Objective_G (X, NN, G);
}
/*                      End: Do not change   */
/*
   ------------------------------------------
   N-dimensional Rastrigin objective function:
   ------------------------------------------
        Global minimum  : [0, 0, ..., 0]^N
```

```
29          Value            : 0.0
30   */
31  void Objective_F (double X[], int N, double *F){
32          int i;
33          double sum;
34          *F = 0;
35          for (i=0; i<N; i++)
36                  *F = *F + pow(X[i],2) + 10.0 - 10.0*cos(2*pi*X[i]);
37  }
38  void Bounds_F(double *xl, double *xr, int N){
39          int i;
40          for (i=0; i<N; i++)
41          {
42                  xl[i] = -5.12;
43                  xr[i] = 5.12;
44          }
45  }
46  void Objective_G (double X[], int N, double G[]){
47          int i;
48          double sum;
49          for (i=0; i<N; i++)
50                  G[i] = 0.0;

52          for (i=0; i<N; i++)
53                  G[i] = G[i] + 2.0*X[i] + 20.0*pi*sin(2*pi*X[i]);
54  }
```

The user compiles MEMPSODE using the command:

```
make OBJECTIVE=rastrigin
```

We focus on the 30–dimensional instance of this function, using UPSO with unification factor 1.0 (standard gbest PSO), swarm size 20, memetic Scheme 2 with local search probability 0.1, and a maximum of 500000 function evaluations. In addition partial results are displayed every 500 iterations and the initial velocity scaling factor is 0.01. Since local search is used, the user must also define the LS method and its parameters. In this example we use the BFGS method with a maximum of 1000 function evaluations per LS. The corresponding file is defined with the `-y` command line option and contains the following line:

```
bfgs noc 1000
```

The program is executed through the command:

```
mempsode -d 30 -a pso -l 2 -s 20 -f 500000 -c 0.01 -u 1.0 -n 1 -D 500 -y in.dat
```

and produces the following screen output:

```
---------------------------------
Experiment 0
---------------------------------
Iter:   500, FunEvals:      63121, Val: 3.283363E+01, Std: 4.250217, Vel: 0.102400
Iter:  1000, FunEvals:     106916, Val: 1.492438E+01, Std: 3.148346, Vel: 0.102400
Iter:  1500, FunEvals:     127425, Val: 1.492438E+01, Std: 3.146285, Vel: 0.102400
Iter:  2000, FunEvals:     166089, Val: 1.094454E+01, Std: 3.068551, Vel: 0.102400
Iter:  2500, FunEvals:     196388, Val: 8.954626E+00, Std: 3.407066, Vel: 0.102400
Iter:  3000, FunEvals:     229887, Val: 5.969754E+00, Std: 3.105790, Vel: 0.102400
Iter:  3500, FunEvals:     266735, Val: 4.974795E+00, Std: 2.830046, Vel: 0.102400
Iter:  4000, FunEvals:     312303, Val: 4.974795E+00, Std: 3.100652, Vel: 0.102400
```

```
Iter:  4500, FunEvals:     356134, Val: 2.984877E+00, Std: 3.757278, Vel: 0.102400
Iter:  5000, FunEvals:     395173, Val: 1.989918E+00, Std: 2.084046, Vel: 0.102400
Iter:  5500, FunEvals:     426884, Val: 9.949591E-01, Std: 2.120236, Vel: 0.102400
Iter:  6000, FunEvals:     464246, Val: 9.949591E-01, Std: 3.980862, Vel: 0.102400
---------------------------------
PROBLEM PARAMETERS
Problem   : 0
Dimension : 30
NumOfExp  : 1
Seed      : 1
MaxIter   : inf
MaxFev    : 500000
MaxGev    : inf
Target    : -inf
Xmin      : [ -5.120 -5.120 -5.120 -5.120 -5.120 -5.120 -5.120 -5.120
Xmax      : [ 5.120 5.120 5.120 5.120 5.120 5.120 5.120 5.120 5.120

UPSO PARAMETERS
Memetic   : 2
UF        : 1.000
Vscale    : 0.010
Prob      : 0.010
x, c1, c2 : 0.729 2.050 2.050
Nradius   : 1
R3use     : 0
R3mean    : 0.00
R3std     : 1.00
SS        : 20
=================================================================================================================
EXP - S -    F(SOL)    -   ITER  -   FEVALS - LOCAL - GEVALS -               LAST HIT            -  BPUPD - CPU TIME
                                                              ITER ,  FEVALS, GEVALS,  LOCAL

  1 - 0 - 0.000000E+00 -    6388 -   500667 -   502 -      0 - [   6053,  473070,      0,    470] -   7602 -    4.38
=================================================================================================================
Execution Complete
```

In the upper part of the output, MEMPSODE prints intermediate results every 500 iterations (`Iter`). Note that an iteration of the algorithm corresponds to a complete cycle where all particles are updated. The output includes the number of function evaluations (`FunEvals`) performed up to the specific iteration (including those of the local search) and the best approximation to the global minimum found so far (`Val`). In addition the output contains the standard deviation of the average swarm position, defined as:

$$\texttt{Std} = \sum_{i=1}^{N} (x_i - \tilde{x})^\top (x_i - \tilde{x}),$$

where $\tilde{x} = \dfrac{1}{N} \sum_{i=1}^{N} x_i$. In the case of PSO the output also contains the maximum velocity component defined as:

$$\texttt{Vel} = \max_{\substack{i=1,N \\ j=1,n}} |v_{ij}|. \tag{2}$$

When execution completes, MEMPSODE prints a complete list of the parameters used in the experiment(s) as well as the final output in a single line per experiment. `EXP` is the serial number of the experiment and `S` is a flag that is equal to 1 if the algorithm successfully reached the target value (as specified by the `-t` command line option). Two sets of counters are reported: `ITER`, `FEVALS`, `LOCAL` and `GEVALS` are the total number of iterations, function evaluations, local searches and gradient evaluations respectively, that were performed by the algorithm. The value of these counters at the time the lowest function value was discovered is displayed under the heading

`LAST HIT`. `BPUPD` is the total number of best position updates and `CPU TIME` is the execution time for the specific experiment. MEMPSODE writes the global minimizer in an output file that follows a specific naming convention and ends in `_sol` (Details are given in the user's manual).

8.2 Lennard–Jones clusters

In this example application, we find the global minimum of atomic clusters with $m = 5, 13, 19$ and 25 atoms interacting through the Lennard–Jones potential. The total potential energy of the cluster has the form:

$$E = \sum_{i<j}^{m} 4\varepsilon \left[\left(\frac{\sigma}{r_{ij}} \right)^{12} - \left(\frac{\sigma}{r_{ij}} \right)^{6} \right], \tag{3}$$

where r_{ij} is the distance between atoms i and j.

The above function, using reduced units ($\sigma = \varepsilon = 1$), is provided in file `lj.c` (Listing 7).

Listing 7: File: lj.c.

```
#include <math.h>
double      pi=3.14159265358979;
/*                    Begin: Do not change */
 extern struct{
     int ipro;
}funcid_;
void Objective_F (double X[], int N, double *F);
void objwrap_ (double *X, int *N, double *F, int *problem){
      int NN;
      NN = *N; funcid_.ipro = *problem;
      Objective_F (X, NN, F);
}
void Objective_G (double X[], int N, double G[]);
void gradwrap_ (double *X, int *N, double *G, int *problem){
        int NN;
        NN = *N; funcid_.ipro = *problem;
        Objective_G (X, NN, G);
}
/*                    End: Do not change */
double finmin ( double [], int );
double ljpot ( double );
double lj ( double p[], int n ){
        int np, i, j, i3, j3, np1;
        double v, dx, dy, dz, rij;
        double xi, yi, zi, xj, yj, zj;
        np = n/3; v = 0.0; np1 = np-1;
        for (i=0; i<np1; ++i) {
                i3 = 3*i; xi = p[i3]; yi = p[i3+1]; zi = p[i3+2];
                for (j=i+1; j<np; ++j) {
                        j3 = 3*j;
                        xj = p[j3]; yj = p[j3+1]; zj = p[j3+2];
                        dx = xi-xj; dy = yi-yj; dz = zi-zj;
                        rij = sqrt(dx*dx+dy*dy+dz*dz);
                        v += ljpot(rij);
                }
        }
        return v;
}
```

```
#define SIGMA    1.0
#define EPSILON4 4.0
double ljpot ( double r )\{
        double sr, sr3, sr6, sr12;
        sr = SIGMA/r;sr3 = sr*sr*sr; sr6 = sr3*sr3; sr12 = sr6*sr6;
        return EPSILON4*(sr12-sr6);
}
void Objective_F (double X[], int N, double *F){
      int i;
      double sum;
      *F = lj(X, N);
}
void Bounds_F(double *xl, double *xr, int N){
      int i;
      for (i=0; i<N; i++) {
            xl[i] = -2.0; xr[i] = 2.0;
      }
}
```

The user compiles MEMPSODE using the command:

```
make OBJECTIVE=lj
```

Suppose that we want to calculate the conformation of 13 atoms. The objective function for this instance has $13 \times 3 = 39$ free parameters. We are using UPSO with unification factor 1.0 (standard gbest PSO), swarm size 30, memetic Scheme 2 with and a maximum of 500000 function evaluations. In addition partial results are displayed every 500 iterations. Since local search is used, the user must also define the LS method and its parameters. In this example we use the BFGS method with a maximum of 1000 function evaluations per LS. The corresponding file is defined with the `-y` command line option and contains the following line:

```
bfgs noc 1000
```

The program is executed through the command:

```
mempsode -d 39 -a pso -l 2 -s 30 -f 500000 -u 1.0 -n 1 -D 500 -y in.dat
```

and produces the following screen output:

```
----------------------------------
Experiment 0
----------------------------------
Iter:   500, FunEvals:     185576, Val: -4.066981E+01, Std: 2.637274, Vel: 0.400000
Iter:  1000, FunEvals:     358728, Val: -4.432680E+01, Std: 2.561342, Vel: 0.400000
Iter:  1500, FunEvals:     489708, Val: -4.432680E+01, Std: 2.489601, Vel: 0.400000
----------------------------------
PROBLEM PARAMETERS
Problem   : 0
Dimension : 39
NumOfExp  : 1
Seed      : 1
MaxIter   : inf
MaxFev    : 500000
MaxGev    : inf
Target    : -inf
Xmin      : [ -2.000 -2.000 -2.000 -2.000 -2.000 -2.000 -2.000 -2.000 -2.000 -2.000 ...
Xmax      : [ 2.000 2.000 2.000 2.000 2.000 2.000 2.000 2.000 2.000 2.000 2.000 ...

UPSO PARAMETERS
```

```
Memetic   : 2
UF        : 1.000
Vscale    : 0.100
Prob      : 0.010
x, c1, c2 : 0.729 2.050 2.050
Nradius   : 1
R3use     : 0
R3mean    : 0.00
R3std     : 1.00
SS        : 30
=============================================================================================================
EXP - S -    F(SOL)    -   ITER  -   FEVALS - LOCAL - GEVALS -                LAST HIT            -  BPUPD - CPU TIME
                                                                ITER ,  FEVALS, GEVALS,  LOCAL

  1 - 0 - -4.432680E+01 -    1563 -   501236 -   386 -      0 - [    757,  280247,      0,    214] -    236 -    3.63
=============================================================================================================
Execution Complete
```

8.3 Protein conformation

In order to understand the diverse functionality of proteins at the molecular level, it is necessary to determine their three–dimensional structure. This task is often formulated as a global optimization problem of a suitable chosen potential energy function. The *protein folding problem* is defined as the problem of determining the three–dimensional structure of a protein, called its *tertiary structure*, solely by the sequence of amino acids that it is composed of (also called its *primary structure*) [1]. Under the assumption that in the native state the potential energy of a protein is globally minimized, the protein folding problem can be regarded as equivalent to the global minimization problem:

$$\min_{x \in \mathbf{R}^{3n}} E(x), \tag{4}$$

where $E(x)$ is the value of a potential energy function of the n atom protein described by the $3n$–dimensional coordinate vector x.

In general, such optimization problems are multi–dimensional (depending on the size of the amino acid sequence) and they have a multitude of local minimizers, whose number is usually exponential to the number of atoms. It is a general belief that in the native state of a protein, its potential energy function takes its global energy minimum value.

There are two different but equivalent coordinate systems used to describe the conformation of a protein: *internal* and *external* (or Cartesian) coordinates. In external coordinates each atom is represented by its Cartesian coordinates (x, y, z). In the internal coordinates system, which is more closely related to the structure of a protein, we use the *bond length*, *bond angle* and *dihedral angle*, to specify the coordinates of the atom.

The bond length is defined as the Euclidean distance between two consecutive atoms. The bond angle is the angle between three consecutive atoms. Finally, for every sequence of four consecutive atoms, the dihedral angle is the angle defined by the plain of the first three atoms and the last three atoms of the sequence. Both coordinate systems require $3n$ variables to describe the conformation of a protein. It is known that, in the native state of the protein, the variation in the values of the bond angles and the bond lengths is minimal. In addition, the bond between two amino acids is very rigid and may be kept fixed.

It is easier to keep bond lengths and bond angles fixed in internal rather than in external coordinates. Hence, the use of internal coordinates is computationally more efficient than external coordinates. For amino acids, fixing the bond lengths and bond angles to the common values,

results in three free variables (dihedral angles). Hence, a large dimensionality reduction can be achieved by using internal coordinates, without limiting the possible folded states attained by a protein.

In this example we find the minimum energy conformation of a gas phase Alanine octamer. For the potential energy we employ the Amber [2] force field which has the form :

$$\begin{aligned} E &= \sum_{\text{bonds}} k_r(r-r_0)^2 + \sum_{\text{angles}} k_\theta(\theta-\theta_0)^2 + \sum_{\text{dihedrals}} \frac{V_n}{2}[1+\cos(n\phi-\gamma)] \\ &+ \sum_{i<j}\left[\frac{A_{ij}}{r_{ij}^{12}} - \frac{B_{ij}}{r_{ij}^{6}}\right] + \sum_{i<j}\left[\frac{q_i q_j}{\epsilon r_{ij}}\right]. \end{aligned}$$

The parameters are taken from the 1996 parametrization [8]. In order to implement the above energy function we have used the Tinker molecular modelling package [13]. In the software distribution we provide the file `tinker.c` which offers an interface to the Tinker energy function and its gradient.

Listing 8: File: tinker.c.

```
void Objective_F (double X[], int N, double *F);
void Objective_G (double X[], int N, double G[]);

 extern struct{
     int ipro;
}funcid_;

extern double tinker_ (double *X, int *N);
extern double granalize_ (int *N, double *X, double *G);

void objwrap_ (double *X, int *N, double *F, int *problem)
{
        int NN;
        NN = *N;
        funcid_.ipro = *problem;
        Objective_F (X, NN, F);
}
void gradwrap_ (double *X, int *N, double *G, int *problem)
{
        int NN;
        NN = *N;
        funcid_.ipro = *problem;
        Objective_G (X, NN, G);
}
void Objective_F (double X[], int N, double *F)
{
        int *NN = &N;
        int i;

        *F = tinker_(X, NN);
}
void Bounds_F(double *xl, double *xr, int N)
{
        int i;
        for (i=0; i<N; i++)
        {
                xl[i] = -180.0;
```

```
                xr[i] =  180.0;
        }
}
void Objective_G(double X[], int N, double G[])
{
        int *NN=&N;
        granalize_(NN, X, G);
}
```

To enable Tinker support one must supply the full path of the Tinker installation. Consequently MEMPSODE is built by using:

```
make OBJECTIVE=tinker TINKER=/path/to/tinker/source
```

Before running the executable the user has to prepare a Tinker input file containing the description of the protein in internal coordinates. A sample file (`vp8.int`) that is included in the distribution is displayed in Listing 9.

Listing 9: File: `vp8.int`.

```
    92  Polyalanine-8
     1  CT    340
     2  C     342     1   1.51050
     3  O     343     2   1.22613     1  119.9528
     4  HC    341     1   1.08868     2  109.4155     3  110.9880     0
     5  HC    341     1   1.08787     2  109.2063     4  109.7323     1
     6  HC    341     1   1.08691     2  109.6322     4  109.7783    -1
     7  N       7     2   1.32934     1  116.5149     3  123.9424    -1
     8  CT      8     7   1.46485     2  125.9743     1 -177.0262     0
     9  C       9     8   1.53973     7  110.7148     2 -155.5942     0
    10  O      11     9   1.23082     8  119.7921     7  -37.3120     0
    11  H      10     7   1.01327     2  119.0392     8  120.0883     1
    12  H1     12     8   1.09153     7  109.0587     9  108.7400    -1
    ...
    78  CT      8    77   1.46334    69  124.8476    68 -177.2743     0
    79  C       9    78   1.53869    77  110.8150    69 -153.2055     0
    80  O      11    79   1.23121    78  119.9793    77  -31.8200     0
    81  H      10    77   1.00955    69  121.4158    78  117.8066    -1
    82  H1     12    78   1.09028    77  109.4646    79  108.7996    -1
    83  CT     13    78   1.53519    77  108.5939    79  110.6660     1
    84  HC     14    83   1.09087    78  109.8634    77  -59.7235     0
    85  HC     14    83   1.09045    78  109.7242    84  108.7582     1
    86  HC     14    83   1.09110    78  110.0195    84  109.3539    -1
    87  N     346    79   1.33761    78  117.6247    80  123.0024     1
    88  CT    348    87   1.46117    79  124.6431    78  175.2712     0
    89  H     347    87   1.01139    79  118.4478    88  120.9440     1
    90  H1    349    88   1.09137    87  109.7642    79  -70.7298     0
    91  H1    349    88   1.09096    87  110.0602    90  109.3936     1
    92  H1    349    88   1.09197    87  109.7969    90  108.6671    -1
```

The dimensionality of the problem is determined by the number of free dihedral angles in the input. In this example we use an Alanine octamer blocked by the Acetyl and N–Methyl groups. Each Alanine residue has four dihedral angles (ϕ, ψ, ω, χ) resulting in 35 optimization variables. A corresponding input file (`vp8.int`) is provided in the distribution, while further details on running the software with Tinker support, is provided in the user manual.

The corresponding Merlin input file is defined with the `-y` command line option and contains the following lines that enable first order derivative information:

```
ana
bfgs noc 1000
```

The program is executed through the command:

```
mempsode -a pso -s 100 -d 35 -u 1.0 -l 2 -P 0.1 -f 15000000 -D 100  < input.tinker
```

and produces the following screen output:

```
---------------------------------
Experiment 0
---------------------------------
Iter:   100, FunEvals:     103036, Val: -2.162610E+01, Std: 364.535534, Vel: 36.000000
Iter:   200, FunEvals:     182889, Val: -2.310521E+01, Std: 387.863649, Vel: 36.000000
Iter:   300, FunEvals:     256804, Val: -2.703174E+01, Std: 324.197461, Vel: 36.000000
Iter:   400, FunEvals:     346899, Val: -2.951969E+01, Std: 376.090091, Vel: 36.000000
Iter:   500, FunEvals:     427020, Val: -2.952570E+01, Std: 354.763858, Vel: 36.000000
Iter:   600, FunEvals:     515875, Val: -3.128046E+01, Std: 586.591717, Vel: 36.000000
Iter:   700, FunEvals:     587275, Val: -3.347804E+01, Std: 358.814968, Vel: 36.000000
Iter:   800, FunEvals:     663487, Val: -3.347804E+01, Std: 619.234927, Vel: 179.992583
Iter:   900, FunEvals:     769654, Val: -3.347804E+01, Std: 372.319558, Vel: 36.000000
Iter:  1000, FunEvals:     848719, Val: -3.347804E+01, Std: 358.663255, Vel: 36.000000
Iter:  1100, FunEvals:     920119, Val: -3.347804E+01, Std: 361.569285, Vel: 36.000000
Iter:  1200, FunEvals:    1001444, Val: -3.347804E+01, Std: 354.937482, Vel: 36.000000
Iter:  1300, FunEvals:    1087552, Val: -3.347804E+01, Std: 512.447495, Vel: 36.000000
Iter:  1400, FunEvals:    1149618, Val: -3.347804E+01, Std: 352.272477, Vel: 36.000000
Iter:  1500, FunEvals:    1226634, Val: -3.347804E+01, Std: 359.445850, Vel: 36.000000
Iter:  1600, FunEvals:    1302420, Val: -3.347804E+01, Std: 354.161961, Vel: 36.000000
...
...
...
Iter: 18100, FunEvals:   13911549, Val: -4.209679E+01, Std: 356.216668, Vel: 36.000000
Iter: 18200, FunEvals:   13957403, Val: -4.209679E+01, Std: 601.879656, Vel: 179.996288
Iter: 18300, FunEvals:   14058492, Val: -4.209679E+01, Std: 363.650181, Vel: 36.000000
Iter: 18400, FunEvals:   14131706, Val: -4.209679E+01, Std: 357.395970, Vel: 36.000000
Iter: 18500, FunEvals:   14206447, Val: -4.209679E+01, Std: 366.269736, Vel: 36.000000
Iter: 18600, FunEvals:   14283906, Val: -4.209679E+01, Std: 599.044905, Vel: 36.000000
Iter: 18700, FunEvals:   14381499, Val: -4.209679E+01, Std: 362.047349, Vel: 36.000000
Iter: 18800, FunEvals:   14458628, Val: -4.209679E+01, Std: 360.760335, Vel: 36.000000
Iter: 18900, FunEvals:   14524413, Val: -4.209679E+01, Std: 362.272162, Vel: 36.000000
Iter: 19000, FunEvals:   14593118, Val: -4.209679E+01, Std: 361.301720, Vel: 36.000000
Iter: 19100, FunEvals:   14677642, Val: -4.214155E+01, Std: 355.223248, Vel: 36.000000
Iter: 19200, FunEvals:   14748737, Val: -4.214155E+01, Std: 377.060474, Vel: 36.000000
Iter: 19300, FunEvals:   14820687, Val: -4.214155E+01, Std: 354.997007, Vel: 36.000000
Iter: 19400, FunEvals:   14895630, Val: -4.214155E+01, Std: 365.626321, Vel: 36.000000
Iter: 19500, FunEvals:   14970416, Val: -4.214155E+01, Std: 361.689891, Vel: 36.000000

 ###############################################################################
 ###############################################################################
 ##                                                                           ##
 ##             TINKER  ---  Software Tools for Molecular Design              ##
 ##                                                                           ##
 ##                         Version 5.1  February 2010                        ##
 ##                                                                           ##
 ##               Copyright (c)  Jay William Ponder  1990-2010                ##
 ##                              All Rights Reserved                          ##
 ##                                                                           ##
 ###############################################################################
 ###############################################################################


 Enter Internal Coordinate File Name :
 Enter Potential Parameter File Name :
 ---------------------------------
PROBLEM PARAMETERS
Problem   : 0
Dimension : 35
NumOfExp  : 1
Seed      : 1299954573
```

```
MaxIter   : inf
MaxFev    : 15000000
MaxGev    : inf
Target    : -inf
Xmin      : [ -180.000 -180.000 -180.000 -180.000 -180.000 -180.000 ...
Xmax      : [ 180.000 180.000 180.000 180.000 180.000 180.000 180.000 ...

UPSO PARAMETERS
Memetic   : 2
UF        : 1.000
Vscale    : 0.100
Prob      : 0.100
x, c1, c2 : 0.729 2.050 2.050
Nradius   : 1
R3use     : 0
R3mean    : 0.00
R3std     : 1.00
SS        : 100
================================================================================================================================
EXP - S -    F(SOL)    -   ITER  -   FEVALS - LOCAL - GEVALS -                LAST HIT            -  BPUPD - CPU TIME
                                                                 ITER ,  FEVALS, GEVALS,  LOCAL

  1 - 0 - -4.214155E+01 -   19525 - 15000170 - 39981 - 12133785 - [  15140,11720482,9482169,  30996] -  67890 - 16014.34
Output tinker file
================================================================================================================================
Execution Complete
```

8.4 Comparative experiments

In this case the target is to explore the effect of the hybrid schemes (1, 2 and 3) in the UPSO framework on an objective function of variable dimensionality. Also we would like to study the behavior of the hybrid schemes for different values of unification factor. This experiment coded as a shell script is shown in Listing 10. In this script MEMPSODE is called inside a three–loop structure $4 \cdot 3 \cdot 6 = 72$ times.

Listing 10: Sample bash script file `run.sh`

```
#! /bin/bash
# Loop problem dimension
for n in `seq 15 15 60`;
do
# Set swarm size
        let ss=50
# Set maximum number of function evals
        let ff=10000*$n
# Loop on memmetic strategy
        for mem in `seq 1 3`;
        do
# Loop on unification factor
          for uf in `uf 0.0 0.2 1.0`;
          do
          ./mempsode -a pso -d $n -s $ss -f $ff -l $mem -u $uf -e 50
          done
        done
done
```

```
make OBJECTIVE=rastrigin
sh run.sh
```

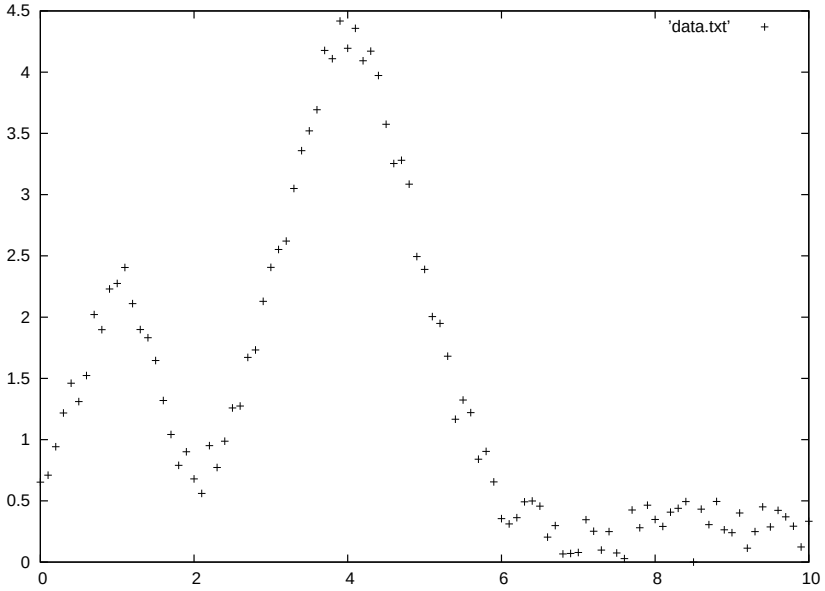

Figure 1: Data fitting example data

Upon successful termination 72 sets of output files will be created, each one containing the results of 50 independent experiments. The results may be analyzed using the following command which prints the average number of function evaluations, local searches and gradient evaluations:

```
awk '{s1+=$5; s2+=$6; s3+=$7} END{printf("%f %f %f \n", s1/NR, s2/NR, s3/NR)}' filename_rep
```

8.5 User callable interface: A data fitting example

In order to illustrate the usage of user callable interface we provide a simple data fitting example. In Figure 1 we give a plot of 101 pairs $\{t_i, y_i\}$.

We will attempt to fit the data using a sum of two gaussians

$$m(t;x) = x_1 e^{-\frac{|x_2-t|^2}{x_3}} + x_4 e^{-\frac{|x_5-t|^2}{x_6}} \tag{5}$$

The fitting problem reduces to minimize the following sum:

$$f(x) = \frac{1}{101}\sum_{i=1}^{101}(m(t_i;x) - y_i)^2 \tag{6}$$

In file `fit.c` we provide an implementation of the objective function:

Listing 11: File `fit.c`

```
#include <math.h>
#include <stdio.h>
double  pi=3.14159265358979;
/*
 * ==========================================================
                     Begin: Do not change
 * ==========================================================
 */
 extern struct{
     int ipro;
}funcid_;
```

```
void Objective_F (double X[], int N, double *F); void objwrap_
(double *X, int *N, double *F, int *problem) {
    int NN;
    NN = *N;
    funcid_.ipro = *problem;
    Objective_F (X, NN, F);
}
void Objective_G (double X[], int N, double G[]); void gradwrap_
(double *X, int *N, double *G, int *problem) {
    int NN;
    NN = *N;
    funcid_.ipro = *problem;
    Objective_G (X, NN, G);
} /*
 * ========================================================
                    End: Do not change
 * ========================================================
 */

void Objective_F (double X[], int N, double *F) {
    int i;
    double x[101], y[101], fit[101];
    double sum;
    *F = 0;
    FILE *fid;

    fid = fopen("data.txt", "r");
    for (i=0; i<101; i++)
    {
        fscanf(fid, "%lf␣%lf\n", &x[i], &y[i]);
    }
    fclose(fid);

    for (i=0;i<101;i++)
    {
        fit[i] = X[0]*exp(-(x[i]-X[1])*(x[i]-X[1])/X[2]) +
                 X[3]*exp(-(x[i]-X[4])*(x[i]-X[4])/X[5]);
        *F = *F  + (fit[i] - y[i])*(fit[i] - y[i]);
    }
    *F = *F / 101.0;
}

void Bounds_F(double *xl, double *xr, int N) {
    int i;
    for (i=0; i<N; i++)
    {
        xl[i] = 0.0;
        xr[i] = 10.0;
    }
}
```

A sample main program is shown in file `sample_main.c`:

Listing 12: File `sample_main.c`

```
#include <stdio.h>
#include <stdlib.h>
```

```
#include "mempsode.h"

int main(int argc, char **argv)
{
    int i, id;
    double val, point[100];

    init_mempsode();
    set_mempsode_iparam("algorithm", 1);
    set_mempsode_iparam("dimension", 6);
    set_mempsode_iparam("swarm-size", 20);
    set_mempsode_iparam("max-fun-evals", 1000);
    set_mempsode_iparam("dump-iterations", 0);
    set_mempsode_iparam("memetic", 0);
    set_mempsode_iparam("seed", -1);

    mempsode(); /* Call the minimizer */

    get_mempsode_min("minval", &val);
    get_mempsode_min("minx", point);
        printf("Val-->%f␣\n", val);
    for (i=0; i<prob.dim; i++)
        printf("X(%i)=%f\n", i, point[i]);
}
```

The user links and runs the library by issuing the following:

```
$ gcc sample_main.c fit.c -lmempsode -L/home/voglis/mempsode
                        -I/home/voglis/mempsode -lgfortran
$./a.out
Val-->0.046554
X(0)=2.217607
X(1)=0.970539
X(2)=0.617581
X(3)=4.216262
X(4)=4.027260
X(5)=1.746662
```

The minimizer returned corresponds to the 6 parameters of the gaussian model function. In Figure 2 we plot the calculated model against the data.

9 Provided Objective Functions

The software distribution contains a file `SOURCE_problems.c` where 31 well established test functions are implemented in `C`. These test function are described below and can be used by the `-o` option. For example, function TP3 can be chosen by issuing the `-o 3` option.

TP0: **Sphere Problem ([11])** This is a separable n–dimensional problem, defined as:

$$f(x) = \sum_{i=1}^{n} x_i^2,$$

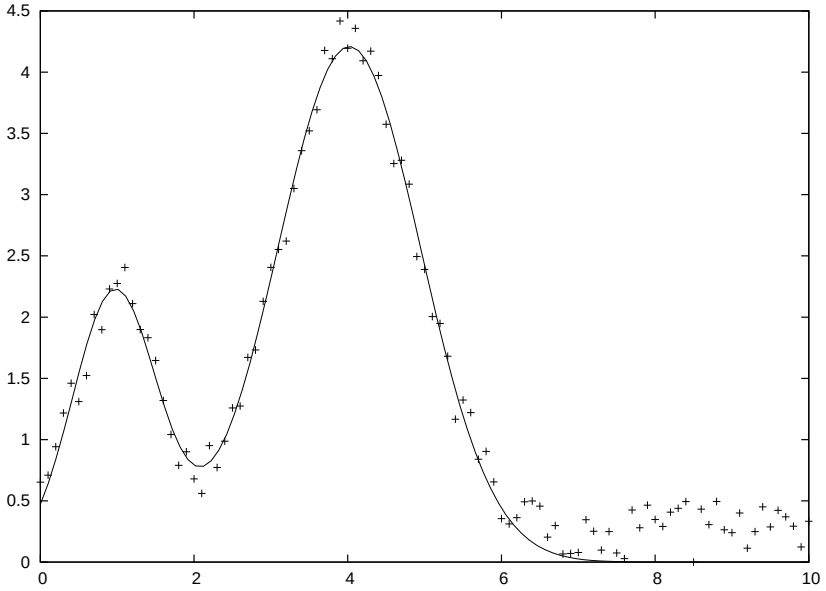

Figure 2: Fitted data

and it has a single global minimizer, $x^* = (0, 0, \ldots, 0)^\top$, with $f(x^*) = 0$.

TP1: **Generalized Rosenbrock ([11])** This is a non–separable n–dimensional problem, defined as:

$$f(x) = \sum_{i=1}^{n-1} \left(100 \left(x_{i+1} - x_i^2 \right)^2 + (x_i - 1)^2 \right),$$

and it has a global minimizer, $x^* = (1, 1, \ldots, 1)^\top$, with $f(x^*) = 0$.

TP2: **Rastrigin ([11])** This is a separable n–dimensional problem, defined as:

$$f(x) = 10n + \sum_{i=1}^{n} \left(x_i^2 - 10 \cos(2\pi x_i) \right),$$

and it has a global minimizer, $x^* = (0, 0, \ldots, 0)^\top$, with $f(x^*) = 0$.

TP3: **Griewank ([11])** This is a non–separable n–dimensional problem, defined as:

$$f(x) = \sum_{i=1}^{n} \frac{x_i^2}{4000} - \prod_{i=1}^{n} \cos\left(\frac{x_i}{\sqrt{i}} \right) + 1,$$

and it has a global minimizer, $x^* = (0, 0, \ldots, 0)^\top$, with $f(x^*) = 0$.

TP4: **Ackley ([11])** This is a non–separable n–dimensional problem, defined as:

$$f(x) = 20 + \exp(1) - 20 \exp\left(-0.2 \sqrt{\frac{1}{n} \sum_{i=1}^{n} x_i^2} \right) - \exp\left(\frac{1}{n} \sum_{i=1}^{n} \cos(2\pi x_i) \right),$$

and it has a global minimizer, $x^* = (0, 0, \ldots, 0)^\top$, with $f(x^*) = 0$.

TP5: **Freundenstein-Roth ([11])** This is a non–separable 2–dimensional problem, defined as:

$$f(x) = (-13 + x_1 + ((5 - x_2)x_2 - 2)x_2)^2 + (-29 + x_1 + ((x_2 + 1)x_2 - 14)x_1)^2,$$

and it has a global minimizer, $x^* = (4, 5)^\top$, with $f(x^*) = 0$.

TP6: **Levy No3 ([11])** This is a non–separable 2–dimensional problem, defined as:

$$f(x) = \sum_{i=1}^{5} i\cos((i+1)x_1 + i) \sum_{j=1}^{5} cos((j+1)x_2 + j),$$

and it has a global minimizer with $f(x^*) = -176.542$.

TP7: **Levy No8 ([11])** This is a non–separable 2–dimensional problem, defined as:

$$f(x) = \sin^2(\pi y_1) + \sum_{i=1}^{n-1} \left((y_1 - 1)^2 + (1 + 10\sin^2(\pi y_{i+1}))\right) + (y_n - 1)^2,$$

where $y_i = 1 + \frac{x_i - 1}{4}$ and it has a global minimizer, $x^* = (1, 1, \ldots, 1)^\top$, with $f(x^*) = 0$.

TP8: **Powell badly scaled ([11])** This is a non–separable 2–dimensional problem, defined as:

$$f(x) = (10000x_1x_2 - 1)^2 + (e^{-x_1} + e^{-x_2} - 1.001)^2,$$

and it has a global minimizer, $x^* = (1.09815e - 5; 9.106146)^\top$, with $f(x^*) = 0$.

TP9: **Wood ([11])** This is a non–separable 4–dimensional problem, defined as:

$$f(x) = (10(x_1 - x_2^2))^2 + (1 - x_1)^2 + (x_4 - x_1x_1)^2 + (1 - x_3)^2 + 10(x_2 - x_4 - 2)^2 + 0.1(x_2 - x_4)^2,$$

and it has a global minimizer, $x^* = (1, 1, 1, 1)^\top$, with $f(x^*) = 0$.

TP10: **Trigonometric ([11])** This is a non–separable n–dimensional problem, defined as:

$$f(x) = \sum_{i=1}^{n} \left(n - \sum_{j=1}^{n} cosx_j + i(1 + cosx_i) - sinx_i \right)^2,$$

and it has a global minimizer, $x^* = (1/n, 1/n, \ldots, 1/n)^\top$, with $f(x^*) = 0$.

TP11: **Schaffer F6 ([11])** This is a non–separable n–dimensional problem, defined as:

$$f(x) = 0.5 + \frac{\sin^2\left(\sqrt{x_1^2 + x_2^2}\right) - 0.5}{1 + 0.001\left(x_1^2 + x_2^2\right)^2},$$

and it has a global minimizer, $x^* = (0, 0, \ldots, 0)^\top$, with $f(x^*) = 0$.

TP12: **Schwefel's double sum ([11])** This is a non–separable n–dimensional problem, defined as:

$$f(x) = \sum_{i=1}^{n} \left(\sum_{j=1}^{i} x_j \right)^2$$

and it has a global minimizer, $x^* = (0, 0, \ldots, 0)^\top$, with $f(x^*) = 0$.

TP13: **Schwefel ([11])** This is a non–separable n–dimensional problem, defined as:

$$f(x) = 418.9829n + \sum_{i=1}^{n} x_i sin\left(\sqrt{|x_i|}\right),$$

and it has a global minimizer, $x^* = (-420.9687, -420.9687, \ldots, -420.9687)^\top$, with $f(x^*) = 0$.

TP14 **Interval Arithmetic Benchmark ([6])** This problem consists of the following system:

$$\begin{cases} x_1 - 0.25428722 - 0.18324757\ x_4x_3x_9 &= 0, \\ x_2 - 0.37842197 - 0.16275449\ x_1x_{10}x_6 &= 0, \\ x_3 - 0.27162577 - 0.1695507\ x_1x_2x_{10} &= 0, \\ x_4 - 0.19807914 - 0.15585316\ x_7x_1x_6 &= 0, \\ x_5 - 0.44166728 - 0.19950920\ x_7x_6x_3 &= 0, \\ x_6 - 0.14654113 - 0.18922793\ x_8x_5x_{10} &= 0, \\ x_7 - 0.42937161 - 0.21180486\ x_2x_5x_8 &= 0, \\ x_8 - 0.07056438 - 0.17081208\ x_1x_7x_6 &= 0, \\ x_9 - 0.34504906 - 0.19612740\ x_{10}x_6x_8 &= 0, \\ x_{10} - 0.42651102 - 0.21466544\ x_4x_8x_1 &= 0. \end{cases}$$

The resulting objective function defined by:

$$f(x) = \sum_{i=1}^{m} |f_i(x)|,$$

with the global minimum $f(x^*) = 0$.

TP15 **Neurophysiology Application ([6])** This problem consists of the following system:

$$\begin{cases} x_1^2 + x_3^2 &= 1, \\ x_2^2 + x_4^2 &= 1, \\ x_5x_3^3 + x_6x_4^3 &= c_1, \\ x_5x_1^3 + x_6x_2^3 &= c_2, \\ x_5x_1x_3^2 + x_6x_4^2x_2 &= c_3, \\ x_5x_1^2x_3 + x_6x_2^2x_4 &= c_4, \end{cases}$$

where the constants, $c_i = 0$, $i = 1, 2, 3, 4$. The resulting objective function defined by:

$$f(x) = \sum_{i=1}^{m} |f_i(x)|,$$

, with the global minimum $f(x^*) = 0$.

TP16 **Chemical Equilibrium Application ([6])** This problem consists of the following system:

$$\begin{cases} x_1x_2 + x_1 - 3x_5 &= 0, \\ 2x_1x_2 + x_1 + x_2x_3^2 + R_8x_2 - Rx_5 + 2R_{10}x_2^2 + R_7x_2x_3 + R9x_2x_4 &= 0, \\ 2x_2x_3^2 + 2R_5x_3^2 - 8x_5 + R_6x_3 + R_7x_2x_3 &= 0, \\ R_9x_2x_4 + 2x_4^2 - 4Rx_5 &= 0, \\ x_1(x_2 + 1) + R_{10}x_2^2 + x_2x_3^2 + R_8x_2 + R_5x_3^2 + x_4^2 - 1 + R_6x_3 + R_7x_2x_3 + R_9x_2x_4 &= 0, \end{cases}$$

where,

$$R = 10, \quad R_5 = 0.193, \quad R_6 = \frac{0.002597}{\sqrt{40}}, \quad R_7 = \frac{0.003448}{\sqrt{40}},$$
$$R_8 = \frac{0.00001799}{40}, \quad R_9 = \frac{0.0002155}{\sqrt{40}}, \quad R_{10} = \frac{0.00003846}{40}.$$

The corresponding objective function

$$f(x) = \sum_{i=1}^{m} |f_i(x)|,$$

, with the global minimum $f(x^*) = 0$.

TP17 **Kinematic Application) ([6])** This problem consists of the following system:

$$\begin{cases} x_i^2 + x_{i+1}^2 - 1 & = & 0, \\ a_{1i}x_1x_3 + a_{2i}x_1x_4 + a_{3i}x_2x_3 + a_{4i}x_2x_4 + a_{5i}x_2x_7 + \\ a_{6i}x_5x_8 + a_{7i}x_6x_7 + a_{8i}x_6x_8 + a_{9i}x_1 + a_{10i}x_2 + a_{11i}x_3 + \\ a_{12i}x_4 + a_{13i}x_5 + a_{14i}x_6 + a_{15i}x_7 + a_{16i}x_8 + a_{17i} & = & 0, \end{cases}$$

with a_{ki}, $1 \leqslant k \leqslant 17$, $1 \leqslant i \leqslant 4$, is the corresponding element of the k–th row and i–th column of the matrix:

$$A = \begin{bmatrix} \text{-0.249150680} & 0.125016350 & \text{-0.635550077} & 1.48947730 \\ 1.609135400 & \text{-0.686607360} & \text{-0.115719920} & 0.23062341 \\ 0.279423430 & \text{-0.119228120} & \text{-0.666404480} & 1.32810730 \\ 1.434801600 & \text{-0.719940470} & 0.110362110 & \text{-0.25864503} \\ 0.000000000 & \text{-0.432419270} & 0.290702030 & 1.16517200 \\ 0.400263840 & 0.000000000 & 1.258776700 & \text{-0.26908494} \\ \text{-0.800527680} & 0.000000000 & \text{-0.629388360} & 0.53816987 \\ 0.000000000 & \text{-0.864838550} & 0.581404060 & 0.58258598 \\ 0.074052388 & \text{-0.037157270} & 0.195946620 & \text{-0.20816985} \\ \text{-0.083050031} & 0.035436896 & \text{-1.228034200} & 2.68683200 \\ \text{-0.386159610} & 0.085383482 & 0.000000000 & \text{-0.69910317} \\ \text{-0.755266030} & 0.000000000 & \text{-0.079034221} & 0.35744413 \\ 0.504201680 & \text{-0.039251967} & 0.026387877 & 1.24991170 \\ \text{-1.091628700} & 0.000000000 & \text{-0.057131430} & 1.46773600 \\ 0.000000000 & \text{-0.432419270} & \text{-1.162808100} & 1.16517200 \\ 0.049207290 & 0.000000000 & 1.258776700 & 1.07633970 \\ 0.049207290 & 0.013873010 & 2.162575000 & \text{-0.69686809} \end{bmatrix}.$$

The corresponding objective function

$$f(x) = \sum_{i=1}^{m} |f_i(x)|,$$

, with the global minimum $f(x^*) = 0$.

TP18 **Combustion Application ([6])** This problem consists of the following system:

$$\begin{cases} x_2 + 2x_6 + x_9 + 2x_{10} & = & 10^{-5}, \\ x_3 + x_8 & = & 3 \times 10^{-5}, \\ x_1 + x_3 + 2x_5 + 2x_8 + x_9 + x_{10} & = & 5 \times 10^{-5}, \\ x_4 + 2x_7 & = & 10^{-5}, \\ 0.5140437 \times 10^{-7}x_5 & = & x_1^2, \\ 0.1006932 \times 10^{-6}x_6 & = & 2x_2^2, \\ 0.7816278 \times 10^{-15}x_7 & = & x_4^2, \\ 0.1496236 \times 10^{-6}x_8 & = & x_1x_3, \\ 0.6194411 \times 10^{-7}x_9 & = & x_1x_2, \\ 0.2089296 \times 10^{-14}x_{10} & = & x_1x_2^2. \end{cases}$$

The corresponding objective function

$$f(x) = \sum_{i=1}^{m} |f_i(x)|,$$

, with the global minimum $f(x^*) = 0$.

TP19 **Economics Modeling Application ([6])** This problem consists of the following system:

$$\begin{cases} \left(x_k + \sum_{i=1}^{n-k-1} x_i x_{i+k}\right) x_n - c_k &= 0, \\ \sum_{l=1}^{n-1} x_l + 1 &= 0, \end{cases}$$

where $1 \leqslant k \leqslant n-1$, and $c_i = 0$, $i = 1, 2, \ldots, n$. The problem was considered in its 20–dimensional instance. Thus, the corresponding objective function was also 20–dimensional, with global minimum $f(x^*) = 0$.

TP20: **Shifted Sphere Problem [16]** This is a separable n–dimensional problem, defined as:

$$f(x) = \sum_{i=1}^{n} (x_i - o_i)^2 + f_{\text{bias}},$$

and it has one global minimizer, $x^* = (o_1, o_2, \ldots, o_n)^\top$, with $f(x^*) = f_{\text{bias}} = -450$.

TP21: **Shifted Generalized Rosenbrock ([16])** This is a non–separable n–dimensional problem, defined as:

$$f(x) = \sum_{i=1}^{n-1} \left(100 \left(z_{i+1} - z_i^2\right)^2 + (z_i - 1)^2\right) + f_{\text{bias}},$$

where $z = x - o + (1, 1, \ldots, 1)^\top$, and it has a global minimizer, $x^* = (o_1, o_2, \ldots, o_n)^\top$, with $f(x^*) = f_{\text{bias}} = 390$.

TP22: **Shifted Rastrigin ([16])** This is a separable n–dimensional problem, defined as:

$$f(x) = 10n + \sum_{i=1}^{n} \left(z_i^2 - 10\cos(2\pi z_i)\right) + f_{\text{bias}},$$

where $z = x - o$, and it has a global minimizer, $x^* = (o_1, o_2, \ldots, o_n)^\top$, with $f(x^*) = f_{\text{bias}} = -330$.

TP23: **Shifted Griewank ([16])** This is a non–separable n–dimensional problem, defined as:

$$f(x) = \sum_{i=1}^{n} \frac{z_i^2}{4000} - \prod_{i=1}^{n} \cos\left(\frac{z_i}{\sqrt{i}}\right) + 1 + f_{\text{bias}},$$

where $z = x - o$, and it has a global minimizer, $x^* = (o_1, o_2, \ldots, o_n)^\top$, with $f(x^*) = f_{\text{bias}} = -180$.

TP24: **Shifted Ackley ([16])** This is a non–separable n–dimensional problem, defined as:

$$f(x) = 20 + \exp(1) - 20\exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} z_i^2}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n} \cos(2\pi z_i)\right) + f_{\text{bias}},$$

where $z = x - o$, and it has a global minimizer, $x^* = (o_1, o_2, \ldots, o_n)^\top$, with $f(x^*) = f_{\text{bias}} = -140$.

TP25: **Shifted Schwefel ([16])** This is a non–separable n–dimensional problem, defined as:

$$f(x) = \max_i\{|z_i|\} + f_{\text{bias}}, \quad 1 \leq i \leq n$$

where $z = x - o$, and it has a global minimizer, $x^* = (o_1, o_2, \ldots, o_n)^\top$, with $f(x^*) = f_{\text{bias}} = -450$.

TP26: **Bird ([17])** This is a non–separable 2–dimensional problem, defined as:

$$f(x_1, x_2) = sin(x_1)\ e^{(1-cos(x_2))^2} + cos(x_2)\ e^{(1-sin(x_1))^2} + (x(1) - x(2))^2$$

and has a global minimizer with $f(x^*) = -106.764537$.

TP27: **Bohachevsky ([17])** This is a separable 2–dimensional problem, defined as:

$$f(x_1, x_2) = x_1^2 + 2x_2^2 - 0.3\ cos(3\pi x_1) - 0.4\ cos(4\pi x_2) + 0.7$$

and has a global minimizer, $x^* = (0, 0)^\top$, with $f(x^*) = 0$.

TP28: **Carrom Table ([17])** This is a non–separable 2–dimensional problem, defined as:

$$f(x_1, x_2) = -\frac{1}{30}\left(\cos x_1\ \cos x_2\ e^{\left(1 - \frac{\left(x_1^2 + x_2^2\right)^{0.5}}{\pi}\right)}\right)^2$$

and has 4 global minimizers with value $f(x^*) = 24.156815$.

TP29: **Guilin Hills ([17])** This is a non–separable 2–dimensional test problem, defined as:

$$f(x) = 3 + \sum_{i=1}^{n} \frac{c_i(x_i + 9)}{x_i + 10} \sin\left(\frac{\pi}{1 - x_i + \frac{1}{2k}}\right)$$

where $c_i = 2,\ i = 1, \ldots, n$ and $k = 5$, that has a global minimizer $x^* = (1 - \frac{1}{8k^2 - 4k}, 1 - \frac{1}{8k^2 - 4k})^\top$ with value $f(x^*) = 0.72750$

TP30: **Holder function ([17])** This is a non–separable 2–dimensional test problem, defined as:

$$f(x_1, x_2) = -\cos x_1 \cos x_2 e^{1 - \frac{\sqrt{x_1^2 + x_2^2}}{\pi}}$$

and has a global minimizer with value $f(x^*) = -10.8723$.

TP31: **Langermann ([17])** This test function has 270 minima inside $[0, 7]^2$.

$$f(x) = \sum_{k=0}^{5} c_k e^{\frac{||x - A(i)||^2}{\pi}}\ \cos\left(\pi||x - A(i)||^2\right)$$

In current implementation $a = (3, 5, 2, 1, 7)^T$, $c = (1, 2, 5, 2, 3)^T$.

References

[1] C.B. Anfinsen. Principles that govern the folding of protein chains. *Science*, 181(96):223–230, 1973.

[2] W.D. Cornell, P. Cieplak, C.I. Bayly, I.R. Gould, K.M. Merz, D.M. Ferguson, D.C. Spellmeyer, T. Fox, J.W. Caldwell, and P.A. Kollman. A second generation force field for the simulation of proteins, nucleic acids, and organic molecules. *Journal of the American Chemical Society*, 117(19):5179–5197, 1995.

[3] W.C. Davidon. Variable metric method for minimization, Argonne Natl. *Labs.(19 59) ANL-599 0 Rev.*

[4] R. Fletcher. A new approach to variable metric algorithms. *The Computer Journal*, 13(3):317, 1970.

[5] R. Fletcher and C.M. Reeves. Function minimization by conjugate gradients. *The Computer Journal*, 7(2):149, 1964.

[6] C. Grosan and A. Abraham. A new approach for solving nonlinear equations systems. *IEEE Transactions on Systems, Man, and Cybernetics–Part A: Systems and Humans*, 38(3):698–714, 2008.

[7] K.M. Khoda, Y. Liu, and C. Storey. Generalized Polak-Ribiere algorithm. *Journal of Optimization Theory and Applications*, 75(2):345–354, 1992.

[8] P. Kollman, R. Dixon, W. Cornell, T. Fox, C. Chipot, and A. Pohorille. The development/application of a minimalistorganic/biochemical molecular mechanic force field using a combination of ab initio calculations and experimental data. *Computer Simulations of Biomolecular Systems: Theoretical and Experimental Applications*, pages 83–96, 1997.

[9] I.E. Lagaris, D.G. Papageorgiou, and I.N. Demetropoulos. MERLIN-3.0 A multidimensional optimization environment. *Comput. Phys. Commun*, 109:227–249, (CPC Program Library, program ADHQ)1998.

[10] J.A. Nelder and R. Mead. A simplex method for function minimization. *The computer journal*, 7(4):308, 1965.

[11] K. E. Parsopoulos and M. N. Vrahatis. *Particle Swarm Optimization and Intelligence: Advances and Applications*. Information Science Publishing (IGI Global), 2010.

[12] E. Polak and G. Ribiere. Note sur la convergence de méthodes de directions conjuguées. *Revue Française dInformatique et de Recherche Opérationnelle*, 16:35–43, 1969.

[13] J.W. Ponder et al. TINKER: software tools for molecular design. *Department of Biochemistry and Molecular Biophysics, Washington University School of Medicine, St. Louis, MO*, 1998.

[14] M.J.D. Powell. Rank one methods for unconstrained optimization. Technical report, Atomic Energy Research Establishment, Harwell (England), 1969.

[15] M.J.D. Powell. A tolerant algorithm for linearly constrained optimization calculations. *Mathematical Programming*, 45(1):547–566, 1989.

[16] K. Tang, X. Yao, P. N. Suganthan, C. MacNish, Y. P. Chen, C. M. Chen, and Z. Yang. Benchmark functions for the CEC'2008 special session and competition on large scale global optimization. Technical report, Nature Inspired Computation and Applications Laboratory, University of Science and Technology of China, China, 2007.

[17] C. Voglis and I.E. Lagaris. Towards "Ideal Multistart". A stochastic approach for locating the minima of a continuous function inside a bounded domain. *Applied Mathematics and Computation*, 213(1):216–229, 2009.

Command	Method	Parameters	Meaning
`bfgs`	The BFGS method [4] ($L^\top DL$ decomposition) with line search	`noc 1000`	Maximum number of function evaluations
		`iter 100`	Maximum number of iterations
		`gtol 1.e-7`	Gradient convergence criterion
		`xtol 1.e-7`	X-convergence criterion
		`ftol 1.e-7`	F-convergence criterion
`dfp`	The DFP method [3] ($L^\top DL$ decomposition) with line search	`noc 1000`	Maximum number of function evaluations
		`iter 100`	Maximum number of iterations
		`gtol 1.e-7`	Gradient convergence criterion
		`xtol 1.e-7`	X-convergence criterion
		`ftol 1.e-7`	F-convergence criterion
`tolmin`	The BFGS method [15] (Goldfarb-Idnani) with line search	`noc 1000`	Maximum number of function evaluations
		`acc 1.e-7`	Termination accuracy
`trust`	The BFGS method [14] within trust region framework	`noc 1000`	Maximum number of function evaluations
		`iter 100`	Maximum number of iterations
		`gtol 1.e-7`	Gradient convergence criterion
		`xtol 1.e-7`	X-convergence criterion
		`ftol 1.e-7`	F-convergence criterion
`congra`	The conjugate gradient method [5, 7, 12] with line search	`noc 1000`	Maximum number of function evaluations
		`method 'PR'`	Determines the method to be used. 'PR': Polak-Ribiere, 'FR': Fletcher-Reeves, 'GPR': Generalized PR
		`iter 100`	Maximum number of iterations
		`gtol 1.e-7`	Gradient convergence criterion
		`xtol 1.e-7`	X-convergence criterion
		`ftol 1.e-7`	F-convergence criterion
`simplex`	The Nelder and Mead [10] method for nonlinear optimization	`noc 300`	Maximum number of function evaluations
		`iter 100`	Maximum number of iterations
		`init 1`	1:Initialization using `disp`, 2: Initialization with line search
		`disp 0.01`	Displacement vector for initialization
		`xtol 1.e-7`	X-convergence criterion
		`ftol 1.e-7`	F-convergence criterion
`roll`	Alternating directions method (no derivatives)	`noc 300`	Maximum number of function evaluations
		`sfactor 0.01`	Step enhancement factor
		`fail 5`	Number of successive cycle failures allowed

Table 1: A list of Merlin optimization commands and most common parameters

Command	Description
`ana`	Switches to analytic derivatives. The user must provide `Objective_G`.
`numer`	Finite difference derivatives of accuracy $O(n^6)$
`quad`	Finite difference derivatives of accuracy $O(n^2)$
`fast`	Finite difference derivatives of accuracy $O(n)$
`fix 1 2 7-10`	Fix parameters 1, 2 and $7, 8, 9, 10$ to their current values. The optimization does not affect the specified parameters
`gnorm`	Calculates various gradient norms and outputs the result to the specified output file (Default: `out.dat`)

Table 2: Merlin control and configuration commands

Cat.	Option	Values/Range	Default	Description
General	`-a ALG --algorithm=ALG`	{pso, de}	pso	Switches between UPSO and DE.
	`-d NUM --dimension=NUM`	$\{1, 2, \ldots\}$	2	Sets the problem's dimension.
	`-s NUM --swarm-size=NUM`	$\{1, 2, \ldots\}$	20	Sets the swarm size.
	`-e NUM --num-experiments=NUM`	$\{1, 2, \ldots\}$	1	Number of experiments. The algorithm is restarted for every experiment.
	`-l NUM --memetic=NUM`	$\{0, 1, 2, 3\}$	0	Memetic strategy (0: no local search).
	`-P NUM --prob-local-search=NUM`	$[0, 1]$	0.01	Probability of local search (parameter ρ in hybrid scheme(see paper).
	`-n SEED --seed=SEED`	$\{1, 2, \ldots\}$	-	Sets the seed for the random number.
	`-o NUM --objective=NUM`	$\{0, 1, \ldots\}$	0	The objective function's identification number.
	`-E FLOAT --EPS=FLOAT`	$(0, \infty)$	$1.0\text{E}-08$	Specifies a small positive number.
	`-X FLOAT --XEPS=FLOAT`	$(0, \infty)$	$1.0\text{E}-08$	Sets the x–tolerance threshold.
	`-Y FLOAT --FEPS=FLOAT`	$(0, \infty)$	$1.0\text{E}-06$	Sets the function value threshold for termination.
	`-G FLOAT --GEPS=FLOAT`	$(0, \infty)$	$1.0\text{E}-05$	Sets the gradient norm threshold.
	`-h --help`	-	-	Displays help.
Termination rules	`-f NUM --max-fun-evals=NUM`	$\{1, 2, \ldots\}$	$100000n$	Sets the maximum number of function evaluations.
	`-i NUM --max-iter=NUM`	$\{1, 2, \ldots\}$	∞	Sets the maximum number of iterations (cycles).
	`-g NUM --max-grad-evals=NUM`	$\{1, 2, \ldots\}$	∞	Sets the maximum number of gradient evaluations.
	`-t NUM --pso-params=NUM`	$[-infty\,\infty]$	$-\infty$	Sets the desired target value. The algorithm stops if it approximates (less than FEPS) this value.
Input/Output	`-O PREFIX --out-file-prefix=PREFIX`	string	"Default"	Sets the prefix for output files.
	`-p NUM --file-output=NUM`	$\{0, 1\}$	0	Disables/enables output files.
	`-x`	–	–	Enables on–screen printing of the solution vector.
	`-D NUM --dump-iteration=NUM`	$\{0, 1, \ldots\}$	50	Specifies the number of iterations to write temporary results.
	`-R FILE --restore-from-file=FILE`	string	".psode"	Reads initial swarm parameters (positions and velocities) from the provided file.
	`-W FILE --restore-from-file=FILE`	string	".psode"	Creates a file that writes the swarm parameters (positions and velocities).
	`-y FILE --infile=FILE`	string	"in.dat"	File containing Merlin commands.
	`-z FILE --outfile=FILE`	string	"out.dat"	File for Merlin's temporary minimization output.
	`-V`	-	-	When a swarm is read from a file the velocity is randomly re–initialized.
UPSO–related	`-r NUM --radius=NUM`	$\{0, 1, \ldots, M\}$	1	Neighborhood radius in lbest UPSO.
	`-u FLOAT --unification-factor=FLOAT`	$[0, 1]$	1	Unification factor.
	`-U BIN --use-mutation=BIN`	$[0, 1]$	0	Disables/enables velocity mutation.
	`-M FLOAT --mutation-mean=FLOAT`	$(-\infty, \infty)$	0.0	Mutation mean.
	`-S FLOAT --mutation-std=FLOAT`	$(-\infty, \infty)$	1.0	Mutation standard deviation.
	`-c FLOAT --velocity-scaling=FLOAT`	$(0, \infty)$	0.5	Scaling factor for velocity initialization.
	`-b NUM --pso-params=NUM`	$\{1, 2\}$	1	Selects the set of PSO parameters (1: $\chi = 0.729, c_1 = c_2 = 2.05$, 2: $\chi = 0.6, c_1 = c_2 = 2.83$).
DE–related	`-F FLOAT --F-param=FLOAT`	$(-\infty, \infty)$	0.5	F parameter of DE
	`-C FLOAT --C-param=FLOAT`	$(-\infty, \infty)$	0.7	C parameter of DE.
	`-v NUM --de-strategy=NUM`	$\{1, 2, 3, 4, 5\}$	1	Switch DE operator.

Table 3: Complete list of MEMPSODE command line options.