MOSAIC group
Institute of Theoretical Computer Science
Department of Computer Science

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

pCMALib - Manual for version 1.0

Christian Lorenz Müller[1*],
Georg Ofenbeck[1],
Benedikt Baumgartner[2],
Ivo F. Sbalzarini[1]

May 21, 2010

Addresses:
[1]: Institute for Theoretical Computer Science and Swiss Institute of Bioinformatics, ETH Zurich, CH 8092 Zurich, Switzerland
[2]: Robotics and Embedded Systems Group, Department of Informatics, Technische Universität München, Munich, Germany
[*]: Corresponding author: christian.mueller@inf.ethz.ch

Abstract

We present pCMALib, a parallel software library that implements the Evolution Strategy with Covariance Matrix Adaptation (CMA-ES). The library is written in Fortran 90/95 and uses the Message Passing Interface (MPI) for efficient parallelization on shared and distributed memory machines. It allows single CMA-ES optimization runs, embarrassingly parallel CMA-ES runs, and coupled parallel CMA-ES runs using a cooperative island model. So far, one instance of an island model CMA-ES, the recently presented Particle Swarm CMA-ES (PS-CMA-ES) is included using collaborative concepts from Swarm Intelligence for the migration model. Special attention has been given to an efficient design of the MPI communication protocol, a modular software architecture, and a user-friendly programming interface. The library includes a Matlab interface and is supplemented with an efficient Fortran implementation of the official CEC 2005 set of 25 real-valued benchmark functions. This is the first freely available Fortran implementation of this standard benchmark test suite. In this document we focus on all technical aspects of the library, such as compilation, user options and set-up of test problems. We also review in short the implemented algorithms. Further details can be found in the referred literature.

In order to ensure financial support for our project at ETH and allow further development of this software, please cite the following publication in all your documents and manuscripts that made use of this software. Thanks a lot!

C.L. Müller, B. Baumgartner, G. Ofenbeck, B. Schrader, and I. F. Sbalzarini.
pCMALib: a parallel FORTRAN 90 library for the evolution strategy with covariance matrix adaptation.
In Proc. ACM Genetic and Evolutionary Computation Conference (GECCO09),
Montreal, Canada, July 2009.

Contents

1 Introduction

This document comprises a manual to pCMALib, a parallel Fortran90 library for the Evolution Strategy with Covariance Matrix Adaptation (CMA-ES). pCMALib is intended to be a valuable tool for general black-box optimization tasks arising in many areas of science and engineering. A schematic overview of pCMALib is given in Fig. 1. The manual is structured as follows. For the impatient user we first provide

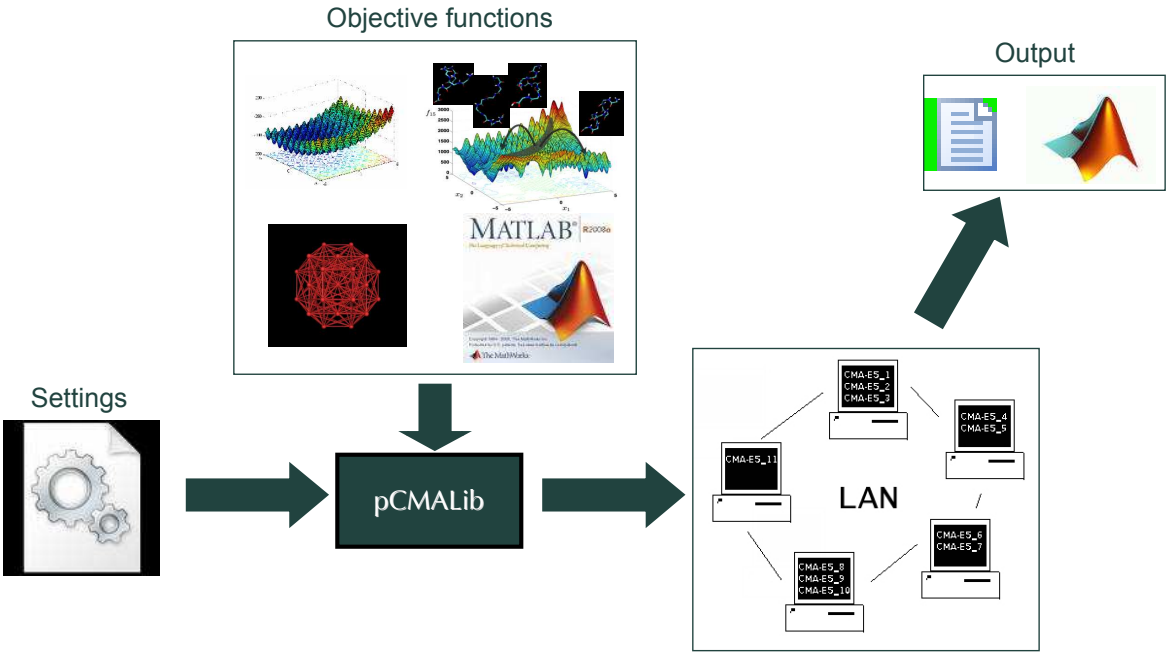


Figure 1: Schematic overview of pCMALib. pCMALib is controlled by a text file that specifies the settings of the algorithmic parameters etc. Within pCMALib, a set of objective functions are available, such as e.g. the CEC 2005 benchmarks test suite or a generic MATLAB test function. pCMALib can run in a single/multi-core or distributed processor environment. Results are stored in either text or MATLAB binary output files.

a Quick Start section. In Section 3 we revisit the standard CMA-ES algorithm and several variants that are available in pCMALib. We also describe the Particle Swarm CMA-ES (PS-CMA-ES), an instance of a parallel island CMA-ES that is included in pCMALib, and an experimental memetic CMA-ES. Readers already familiar with these algorithms can skip this section. Section 4 summarizes the general library features and the structure of library. The Getting Started section 5 contains a detailed description of the system requirements, the specification of the make.inc/makefile and the input/output files. We then provide examples how to test the installation and how to add new objective functions to the library. The document is concluded with a list of known issues with the current version and an outlook on planned developments in pCMALib. We also provide further reading and licensing information.

2 Quick start

pCMALib is a parallel FORTRAN library that implements the Evolution Strategy with Covariance Matrix Adaptation (CMA-ES).

The software can be retrieved as zip-file *soon* from `http://www.mosaic.ethz.ch/Downloads/pCMALib/` or via the svn repository. If you wish to get started by just typing a few lines and running an example, we provide a quick start section here. You can compile pCMALib without MATLAB and MPI (only LAPACK is required).

1. (unzip pCMALib.zip)
2. (cd pCMALib/)
3. emacs make.inc (adapt to the specification of your environment, leave MPI and MATLAB variables on default)
4. make new (compiles pCMALib)
5. cd bin/
6. ./libcma ../example_inputs/rastrigin_ipop.txt (run a example)
7. cd rast_ipop
8. ls (check the output data)

When MPI is available you can try the following test case

1. (unzip pCMALib.zip)
2. (cd pCMALib/)
3. emacs make.inc (adapt to the specification of your environment, adapt MPI related flags to your system and set HAS_MPI = 1)
4. make clean
5. make new (compiles pCMALib)
6. cd bin/
7. mpirun -np 4 ./libcma ../example_inputs/water_pscma.txt (minimize a water cluster)
8. cd water_pscma
9. ls (check the output data, should contain files for each process)

To test the Matlab interface try the following

1. (unzip pCMALib.zip)
2. (cd pCMALib/)
3. emacs make.inc (adapt to the specification of your environment, adept MAT related flags to your system and set HAS_MAT = 1, MPI flags should be empty and HAS_MPI = 0)
4. make clean
5. make new (compiles pCMALib)
6. cd bin/

7. ./libcma ../example_inputs/matlab_test.txt (minimize a water cluster)
8. cd mat_test
9. ls (check the output data, should contain a .mat file)

3 The Evolution Strategy with Covariance Matrix Adaptation

3.1 Standard CMA-ES

We consider the standard CMA-ES [1, 2, 3] with weighted intermediate recombination, step size adaptation, and a combination of rank-μ update and rank-one update [4]. At each iteration of the algorithm, the members of the new population are sampled from a multivariate normal distribution $\mathcal{N}$ with mean $\mathbf{m} \in \mathbb{R}^n$ and covariance $\mathbf{C} \in \mathbb{R}^{n \times n}$. The sampling radius is controlled by the overall standard deviation (step size) σ. Let $\mathbf{x}_k^{(g)}$ the k^{th} individual at generation g. The new individuals at generation $g+1$ are sampled as:

$$\mathbf{x}_k^{(g+1)} \sim \mathbf{m}^{(g)} + \sigma^{(g)} \mathcal{N}\left(\mathbf{0}, \mathbf{C}^{(g)}\right) \quad k = 1, \ldots, \lambda \,. \tag{1}$$

Fig. 2 sketches the sampling process. The λ sampled points are ranked in order of ascending fitness, and

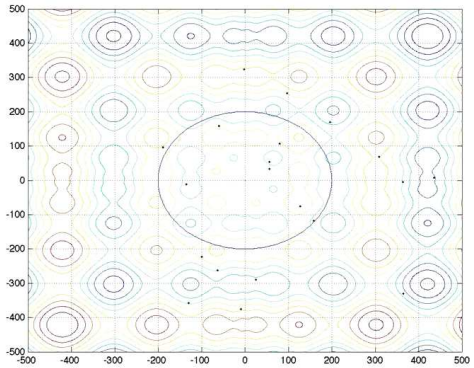


Figure 2: Example of the contour of a covariance matrix and sample points on a 2D fitness landscape

the μ best are selected. The mean of the sampling distribution given in Eq. 1 is updated using *weighted intermediate recombination* of these selected points:

$$\mathbf{m}^{(g+1)} = \sum_{i=1}^{\mu} w_i \mathbf{x}_{i:\lambda}^{(g+1)} \,, \tag{2}$$

with

$$\sum_{i=1}^{\mu} w_i = 1, \quad w_1 \geq w_2 \geq \ldots \geq w_\mu > 0 \,, \tag{3}$$

where the w_i are positive weights, and $\mathbf{x}_{i:\lambda}^{(g+1)}$ denotes the i^{th} ranked individual of the λ sampling points $\mathbf{x}_k^{(g+1)}$. We use the standard CMA-ES implementation, where the sample weights are decreased superlinearly as $w_i = \log(\frac{\lambda-1}{2} + 1) - \log(i)$. We adapt the covariance matrix for the next generation using a combination of rank-μ and rank-one update, thus:

$$\begin{aligned} \mathbf{C}^{(g+1)} = (1 - c_{\text{cov}})\mathbf{C}^{(g)} + \frac{c_{\text{cov}}}{\mu_{\text{cov}}} \underbrace{\mathbf{p}_c^{(g+1)} \mathbf{p}_c^{(g+1)^T}}_{\text{rank-one update}} + \\ c_{\text{cov}} \left(1 - \frac{1}{\mu_{\text{cov}}}\right) \underbrace{\sum_{i=1}^{\mu} w_i \mathbf{y}_{i:\lambda}^{(g+1)} \left(\mathbf{y}_{i:\lambda}^{(g+1)}\right)^T}_{\text{rank-}\mu\text{ update}} , \end{aligned} \tag{4}$$

with $\mu_{\text{cov}} \geq 1$ weighting between rank-μ and rank-one update, $c_{\text{cov}} \in [0, 1]$ the learning rate for the covariance matrix update, and $\mathbf{y}_{i:\lambda}^{(g+1)} = (\mathbf{x}_{i:\lambda}^{(g+1)} - \mathbf{m}^{(g)})/\sigma^{(g)}$. The evolution path $\mathbf{p}_c^{(g+1)}$ and the step size $\sigma^{(g)}$ are determined by elaborate adaptation formulae [5]. All strategy parameters can be externally controlled by the input setting file (see 5).

Two proposed variants of CMA-ES use different standard settings for optimization in a bounded subset $[A, B]^n$: The local restart CMA-ES (LR-CMA-ES) [6] restarts a converged CMA-ES from a random position within the subset $[A, B]^n$ with a purely local initial step size of $\sigma = 10^{-2}(B - A)/2$.

The CMA-ES with iteratively increasing population size (IPOP-CMA-ES) [7] uses a large initial step size of $\sigma = (B - A)/2$, and the population size λ is increased by a factor $incPopSize$ in each sequential *random* restart until the maximum number of function evaluations is reached. The standard setting is $incPopSize = 2$. In addition to these restart settings, pCMALib also allows a restart at the converged position, a strategy that proved valuable when exploring funneled landscapes.

3.2 Parallel island CMA-ES: the Particle Swarm CMA-ES

Today's ever increasing availability of multi-core platforms also proliferates the use of parallel island models in evolutionary computation. There, several instances of an evolutionary algorithm are run in parallel and exchange information. Many migration models and communication topologies have been suggested. For a recent summary we refer to [8]. In order to enhance the performance of CMA-ES on multimodal functions, in particular functions with multiple funnels, we introduced the Particle Swarm CMA-ES [9] where exchange of information between parallel CMA-ES runs is inspired by ideas from Particle Swarm optimization (PSO). Fig. 3 sketches the general concept of PS-CMA-ES. Global swarm

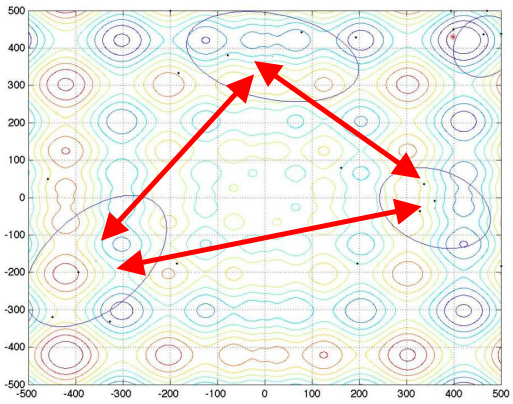


Figure 3: Example of four CMA-ES instances (represented by their covariance matrices and sample points) on a 2D fitness landscape. The arrows illustrate the exchange of non-local information among three instances.

information is included both in the covariance adaptation mechanism of CMA-ES and in the placement of the population mean. In what follows we review the algorithmic details, the implementation concepts can be found in the appendix.

3.2.1 Adapting the Covariance Matrix

We adjust the covariance matrix of the CMA-ES such that it is more likely to sample good candidates, considering the current global best position $\mathbf{p}_{\mathrm{g,best}} \in \mathbb{R}^n$ in the swarm. This is achieved by mixing the CMA covariance matrix from Eq. (4) with a PSO covariance matrix that is influenced by global information:

$$\mathbf{C}^{(g+1)} = c_{\mathrm{p}}\, \mathbf{C}^{(g+1)}_{\mathrm{CMA}} + (1 - c_{\mathrm{p}})\, \mathbf{C}^{(g+1)}_{\mathrm{PSO}}\,, \tag{5}$$

where the mixing weight $c_{\mathrm{p}} \in [0, 1]$ is a new strategy parameter. $\mathbf{C}^{(g+1)}_{\mathrm{CMA}}$ follows the original adaptation rule from Eq. (4). $\mathbf{C}^{(g+1)}_{\mathrm{PSO}}$ is a rotated version of $\mathbf{C}^{(g)}_{\mathrm{CMA}}$ such that the principal eigenvector $\mathbf{b}_{\mathrm{main}}$ of $\mathbf{C}^{(g)}_{\mathrm{CMA}}$ is aligned with the vector $\mathbf{p}_{\mathrm{g}} = \mathbf{p}_{\mathrm{g,best}} - \mathbf{m}^{(g)}$ that points from the current mean $\mathbf{m}^{(g)}$ toward the global best position $\mathbf{p}_{\mathrm{g,best}}$. $\mathbf{C}^{(g)}_{\mathrm{CMA}}$ can be decomposed as $\mathbf{C}^{(g)}_{\mathrm{CMA}} = \mathbf{B}\,\mathbf{D}^2\,\mathbf{B}^T$, such that the rotated covariance matrix can be constructed by rotating the eigenvectors (columns of $\mathbf{B}$), yielding the orthogonal matrix $\mathbf{B}^{(g)}_{\mathrm{rot}} = \mathbf{R}\,\mathbf{B} \in \mathbb{R}^{n\times n}$ of the rotated eigenvectors. $\mathbf{C}^{(g+1)}_{\mathrm{PSO}}$ is then given by:

$$\mathbf{C}^{(g+1)}_{\mathrm{PSO}} = \mathbf{B}^{(g)}_{\mathrm{rot}} \left(\mathbf{D}^{(g)}\right)^2 \left(\mathbf{B}^{(g)}_{\mathrm{rot}}\right)^T. \tag{6}$$

The rotation matrix $\mathbf{R} \in \mathbb{R}^{n\times n}$ is uniquely and efficiently computed using *Givens Rotations* [10]. The Givens rotation matrix $\mathbf{G}$ describes a unique rotation of a vector onto one axis. An n-dimensional rotation

is performed as a sequence of two-dimensional rotations for all possible pairs (i, j) of axes [11]:

$$\mathbf{R}^{(n\times n)} = \prod_{i=1}^{n-1} \prod_{j=i+1}^{n} \mathbf{R}_{ij}\,. \tag{7}$$

$\mathbf{R}_{ij}$ is an $n \times n$ matrix that describes the rotation in the plane spanned by the axes (i, j). It can be considered as a rank-two correction to the identity:

$$\mathbf{R}_{ij,\mathbf{R}_{\text{plane}}} = \begin{array}{c} \\ \\ i \\ \\ j \\ \\ \\ \end{array} \begin{pmatrix} 1 & \dots & 0 & \dots & 0 & \dots & 0 \\ \vdots & \ddots & \vdots & & \vdots & & \vdots \\ 0 & \dots & \mathbf{R}_{\text{plane}}(1,1) & \dots & \mathbf{R}_{\text{plane}}(1,2) & \dots & 0 \\ \vdots & & \vdots & \ddots & \vdots & & \vdots \\ 0 & \dots & \mathbf{R}_{\text{plane}}(2,1) & \dots & \mathbf{R}_{\text{plane}}(2,2) & \dots & 0 \\ \vdots & & \vdots & & \vdots & \ddots & \vdots \\ 0 & \dots & 0 & \dots & 0 & \dots & 1 \end{pmatrix} \begin{array}{l} 1 \\ \\ i \\ \\ j \\ \\ n \end{array}, \tag{8}$$
$$\begin{array}{ccccccc} 1 & & i & & j & & n \end{array}$$

where the 2×2 matrix $\mathbf{R}_{\text{plane}} = \mathbf{G}_{\text{p}}^T\, \mathbf{G}_{\text{b}}$. $\mathbf{G}_{\text{p}}$ is the Givens rotation of the elements i and j of $\mathbf{p}_{\text{g}}$ and $\mathbf{G}_{\text{b}}$ the Givens rotation of the elements i and j of $\mathbf{b}_{\text{main}}$. The complete procedure to compute the rotation matrix $\mathbf{R} \in \mathbb{R}^{n\times n}$ is summarized in Algorithm 1.

Input: Two n-dimensional vectors $\mathbf{b}_{\text{main}}$ and $\mathbf{p}_{\text{g}}$
Result: Rotation matrix $\mathbf{R}$, such that $\mathbf{R}\,\mathbf{b}_{\text{main}} = a\,\mathbf{p}_{\text{g}}$
Initialization: $\mathbf{p} = \mathbf{p_g}$, $\mathbf{b} = \mathbf{b_{main}}$
for $i = (n-1), -1, 1$ **do**
  for $j = n, -1, (i+1)$ **do**

$$\mathbf{p_{plane}} = \begin{pmatrix} p(i) \\ p(j) \end{pmatrix} \qquad \mathbf{b_{plane}} = \begin{pmatrix} b(i) \\ b(j) \end{pmatrix}$$
$$\mathbf{G_p} = \mathbf{Givens}(\mathbf{p_{plane}}) \quad \mathbf{G_b} = \mathbf{Givens}(\mathbf{b_{plane}})$$
$$\mathbf{p} \leftarrow \mathbf{R}_{ij,\mathbf{G_p}}\,\mathbf{p} \qquad \mathbf{b} \leftarrow \mathbf{R}_{ij,\mathbf{G_b}}\,\mathbf{b}$$
$$\mathbf{R_p} \leftarrow \mathbf{R}_{ij,\mathbf{G_p}}\,\mathbf{R_p} \qquad \mathbf{R_b} \leftarrow \mathbf{R}_{ij,\mathbf{G_b}}\,\mathbf{R_b}$$

  end
end
$\mathbf{R} = \mathbf{R_p}^T\mathbf{R_b}$

Algorithm 1: Efficient computation of the n-dimensional rotation matrix using Givens rotations

3.2.2 Biasing the Mean Value

In order to enable individual swarm particles of a PS-CMA-ES to escape local minima, we also bias the mean value in addition to rotating the covariance matrix in direction of the global best solution. After the recombination step of each CMA-ES generation, the updated mean value for the next generation $g + 1$ is biased as:

$$\mathbf{m}^{(g+1)} \leftarrow \mathbf{m}^{(g+1)} + \mathbf{bias}\,. \tag{9}$$

Note that the bias changes the evolution path $\mathbf{p}_c^{(g+1)}$ update for future generations, since the path will be computed with respect to the biased mean value. The biasing rules can be used to include prior knowledge about the structure of the problem, e.g. the correlation length of the fitness landscape. In our benchmark tests, we discriminate the following 3 exploration scenarios that are coupled to the step size σ of each individual CMA-ES instance, a natural measure for its mode of exploration:

1. The CMA-ES instance that produced $\mathbf{p}_{\text{g,best}}$ and all CMA-ES instances with step sizes $\sigma > ||\mathbf{p}_{\text{g}}||$ are not biased at all, thus: $\mathbf{bias} = 0$. Using no bias in these cases avoids catapulting the global

best member out of the funnel with the potential global optimum and prevents very explorative runs from overshooting the target.

2. CMA-ES instances that have converged to local minima far from $\mathbf{p}_{\mathrm{g,best}}$ are characterized by a ratio $\sigma/||\mathbf{p}_{\mathrm{g}}||$ that is smaller than $t_{\mathrm{c}}||\mathbf{p}_{\mathrm{g}}||$, $t_{\mathrm{c}} < 1$. These instances are strongly biased in order to allow them to escape from the current funnel, thus: $\mathbf{bias} = b\,\mathbf{p}_{\mathrm{g}}$. Using a large bias prevents these instances from converging back into the same local minimum again.

3. CMA-ES instances that are not converged, and thus still exploring the space, are given a bias equal to the step size to distance ratio: $\mathbf{bias} = \sigma/||\mathbf{p}_{\mathrm{g}}||\,\mathbf{p}_{\mathrm{g}}$. Using such a small bias prevents clustering of swarm members and preserves the explorative power of the method.

This set of biasing rules, summarized in Algorithm 2, introduces two new strategy parameters, the convergence threshold t_{c} and the biasing factor b.

Input: Mean value $\mathbf{m}^{(g+1)}$ and global best direction $\mathbf{p}_{\mathrm{g}}$
Result: Biased mean value $\mathbf{m}^{(g+1)}$
if $\sigma < ||\mathbf{p}_g||$ **then**
  if $\frac{\sigma}{||\mathbf{p}_g||} \leq t_c||\mathbf{p}_g||$ **then**
    $\mathbf{bias} = b\,\mathbf{p}_{\mathrm{g}}$
  else
    $\mathbf{bias} = \frac{\sigma}{||\mathbf{p}_{\mathrm{g}}||}\,\mathbf{p}_{\mathrm{g}}$
  end
else
  $\mathbf{bias} = 0$
end
$\mathbf{m}^{(g+1)} = \mathbf{m}^{(g+1)} + \mathbf{bias}$

Algorithm 2: **Standard rules for biasing the mean value**

3.2.3 Strategy Parameters

In PS-CMA-ES, all swarm members communicate with each other. A swarm of size S thus requires $S(S-1)$ communication steps. The PSO updates (broadcasting the global best solution, rotating the covariance matrices, and biasing the mean values of all CMA instances) must, however, not be performed at each iteration. Too frequent updates would prevent the CMA-ES instances from evolving and learning the local covariance matrix. Too infrequent updates would lead to premature convergence with several CMA-ES instances stopping in local minima before the first swarm information is exchanged. Clearly, a problem-specific tradeoff has to be found for the communication interval I_{c} between PSO updates, constituting the main strategy parameter of the PS-CMA-ES. In the limit of $I_{\mathrm{c}} \to \infty$, PS-CMA-ES is equivalent to S parallel standard CMA-ES runs.

Other strategy parameters are the swarm size S, the biasing parameters t_{c} and b, and the mixing weight c_{p} in Eq. (5). Both t_{c} and b have been considered random variables at first, but the setting $t_{\mathrm{c}} = 0.1$ and $b = 0.5$ was found a more robust choice on most test functions. The weight c_{p} can, e.g., be randomized, self-adapted, or determined by a preliminary grid search. In the current implementation c_{p} is a constant that can be set in the input file ($PSOWEIGHT$). If it is set to 1, i.e., the PSO covariance is not considered, pCMALib does not compute the $\mathbf{C}_{\mathrm{PSO}}$ which significantly speeds up the code in high dimensions (say $n > 50$).

The swarm size could be chosen so as to reflect the dimensionality of the problem or also using a grid search. It determines the overall population size and the computational overhead of the algorithm. Therefore, S should be chosen as low as possible, but as high as necessary to significantly increase exploration power. In the limit case $S = 1$, PS-CMA-ES is equivalent to standard CMA-ES. In pCMALib, the user can specify all strategy parameters in the input file except t_{c} and b. S is equivalent to the number of processes of an MPI run.

Note that you can combine IPOP-CMA-ES and PS-CMA-ES in pCMALib. This means that if a swarm member converges it is not excluded from the swarm but restarted. This setting can be beneficial if a large number of function evaluations are accessible, i.e. one is not limited by the function evaluation budget.

3.3 Low Discrepancy CMA-ES

Sampling from a multivariate normal distribution relies, in practice, on the use of an efficient pseudo-random number generator such as, e.g., the Mersenne Twister. In recent years, *low-discrepancy* (LD) or *quasi-random* number generators gained, however, some attention. Consider for now the task of uniformly sampling in the box $[0, 1]^n$. Low-discrepancy sequences do not constitute i.i.d samples from the uniform distribution in the box, but rather try to cover the space as uniformly as possible, i.e. to achieve low discrepancy between samples. Sampling from distributions other than the uniform one (in our case from the normal distribution) is achieved by first generating uniform LD sequences and then applying a suitable transformation to the target distribution. Teytaud and Gelly [12] suggested three versions of applying quasi-randomness to CMA-ES:

- substitution of the random mutations with quasi-random mutations
- derandomizing the step size
- random rotation of the offspring's.

Since their results show that substitution of the random mutations yields the biggest performance increase, pCMALib embeds quasi-random number generators that can replace the traditional pseudo-random sampler for this purpose. Available sequences are, e.g., Halton and Sobol sequences. In addition, several different transformation from uniform to normal can be selected. We refer to CMA-ES variants that use LD sequences as LD-CMA-ES.

3.4 Memetic CMA-ES

For some optimization problems, such as, e.g., potential energy minimization in atomic clusters, not only zeroth-order, i.e. objective function values, but also first-order gradient informations may be calculated without additional costs. When gradient information is available, it can be efficiently utilized by, e.g., Quasi-Newton (QN) methods to perform **local** minimization. The most popular QN method is the Broyden-Fletcher-Goldfarb-Shanno (BFGS) method that has been included in pCMALib [**?**]. A crucial point is how to couple gradient optimization with CMA-ES. Such a memetic CMA-ES (from Dawkin's expression for the cultural gene, the meme) has, so far, not been reported in the literature. Our memetic extension for CMA-ES is hence still in experimental phase. Currently, we provide two couplings between CMA-ES and BFGS, a Lamarckian and a Baldwinian approach.

Jean-Baptiste Lamarck was a biologist, who nowadays is mainly remembered for his believe in inheritance of acquired characteristics. In the context of the hybridization of an EA, acquired characteristics relate to the new function value, found via local minimization and the inheritance of the information relates to the position of the found local minimum. An algorithm implementing Lamarckian Evolution is as follows:

- sample population $(\mathbf{x}_1, \ldots, \mathbf{x}_\lambda)$
- minimize each individual $\mathbf{x}_i$ of the population with local search (i.e. with BFGS)
- replace the fitness value **and** the position of the original individiual by the local minimum

The other approach to incorporate the local search is named after James Mark Baldwin, who also became famous for his theories on evolution. In Baldwinian evolution, individuals are not passing on their knowledge to the next generation. The fact that an individual can acquire certain knowledge, given its genetic/phenotypic information, is enough that, under selective pressure, the population will move towards this genetic/phenotypic coding. In the EA context, this relates to a replacement of the fitness of each individiual. A Baldwinian algorithm is as follows:

- sample population $(\mathbf{x}_1, \ldots, \mathbf{x}_\lambda)$
- minimize each individual $\mathbf{x}_i$ of the population with the local search
- replace the fitness value of the original individual by the fitness of the local minimum, but leave the position unchanged.

Applying local minimization to all individuals in CMA-ES is equivalent to a transformation of the underlying fitness landscape (see Wales et al. for details [13]). The transformation removes local multimodality from the landscape. Therefore, search performance might improve. How to use memetic CMA-ES in pCMALib is outlined in 5.

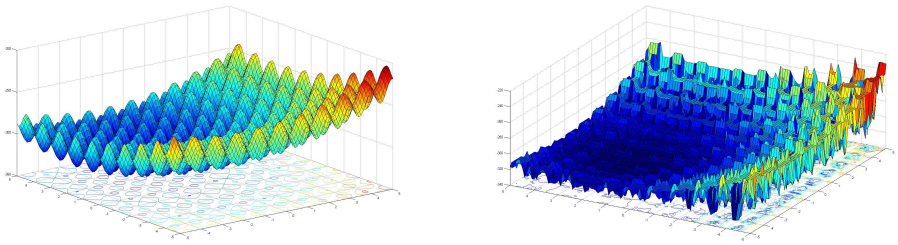


Figure 4: Rastrigin's function before and after applying local minimization (the spikes between neighbouring steps are due to numerical errors of the local search applied here).

4 pCMALib: Features and Structure

pCMALib is a parallel FORTRAN library that implements the Evolution Strategy with Covariance Matrix Adaptation (CMA-ES). The software and additional documents including this manual can be retrieved from `http://www.mosaic.ethz.ch/Downloads/pCMALib`.

We first summarize the key features of pCMALib and then give an introduction to the code design and the library structure.

4.1 General features

pCMALib includes the following features:

- Optimizing objective functions with standard CMA-ES and IPOP-CMA-ES
- Running embarrassingly parallel CMA-ES instances and PS-CMA-ES in a distributed memory environment with MPI
- Efficient Linear Algebra calculation with LAPACK/BLAS
- Sampling with pseuo-random numbers and low discrepancy sequences
- Coupling CMA-ES with Quasi-Newton (BFGS) methods
- Easy control of a large set of strategy parameters via a single input file
- Interfacing with MATLAB, both for objective function call and binary output files
- Benchmarking with the IEEE CEC 2005 (in FORTRAN) and BBOB (in C) benchmark test suite
- Potential Energy calculation for Lennard-Jones and TIPnP water clusters (taken from GMIN **??**)
- Easily extendable to user-specific objective functions

4.2 Code design and Library structure

The core of pCMALib is written in standard FORTRAN 90. A fundamental goal in the code design has been to keep the code structured, easy-to-read and documented. pCMALib's algorithmic flow follows Niko Hansen's MATLAB CMA-ES version (v. 2.54 from 2006) [1]. The user is able to control all relevant algorithmic parameters via an input text file. Output can be generated in text format or MATLAB binary format. All Linear Algebra calculations such as, e.g., matrix multipications, are conducted with LAPACK/BLAS routines. pCMALib is able to run several CMA-ES runs in parallel using MPI. There, each CMA-ES instance is a unique MPI process that are mapped onto the available cores. pCMALib includes third-party software, e.g., for generation of Quasi-/pseudorandom numbers and for gradient descent. Fig. 5 summarizes the library structure. A JAVA-like API will be available soon. We shortly describe the content of all directories and their purpose.

[1] available at http://www.lri.fr/~hansen/cmaesintro.html

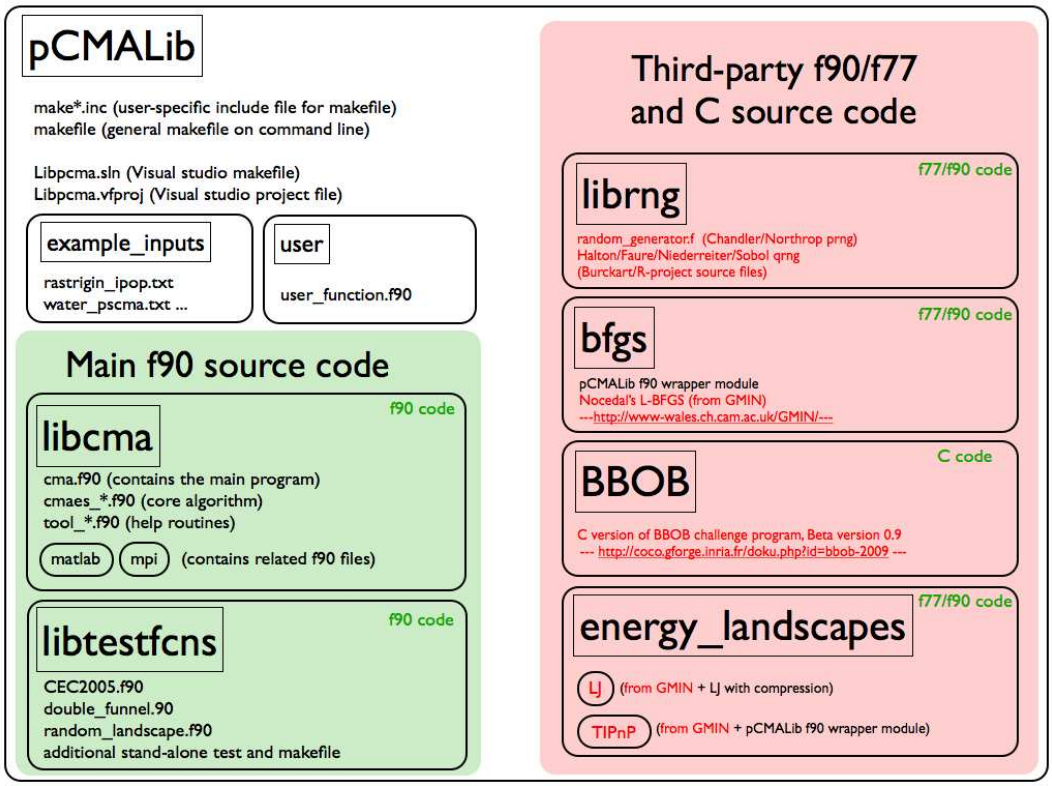


Figure 5: pCMALib directory structure

4.2.1 General compilation and control files

On the top level pCMALib comprises the key compilation files. The files `make*.inc` allow the user to specify all compilation related settings, e.g., the name of the executable, logicals for the use of MATLAB and MPI and their respective location in the system etc. The `make*.inc` is used by the `makefile` via an include statement. This file serves as input to the `make` command (see 5.2 for details). For WIndows Visual Studio users we also included `Libpcma.sln`, a Visual studio makefile, and a Visual Studio project file `Libpcma.vfproj`. The directory *example_inputs* contains several text files that serve as example input to pCMALib's executable (see 5.3 for details).

4.2.2 libcma

The directory **libcma** contains the core f90 source code of pCMALib. Module files that contain key variables and data types needed throughout the source code are named `*_mod.f90`. `cmaes_*.f90` source files contain the core CMA-ES algorithm while `tool_*.f90` files comprise necessary help and support code. The folder **matlab** contains .f90 files that need FORTRAN libraries provided by MATLAB. The folder **mpi** includes source that handles all MPI-related commands. The **main program** is included in `cma.f90`.

4.2.3 libtestfcns

The directory **libtestfcns** provides several benchmark functions that can be used to test the efficienvy of CMA-ES variants. The `CEC2005.f90` provides, e.g., a stand-alone FORTRAN version of the IEEE CEC 2005 benchmark test suite (also with make and test files) [14, 9] and `double_funnel.f90` Lunacek's double funnel problem [15]. If the user intends to code further test functions for CMA-ES, they should be included in this folder.

4.2.4 librng

This directory contains third-party software that is used for random number generation. For pseudo-random numbers we use the Chandler/Northrop f77 implementation of the Marsaglia-Zaman type subtract-with-borrow generator (`rand_generator.f`). For Low discrepancy sequences we included several implementations for Halton, Faure, Niederreiter and Sobol sequences (see files for references).

4.2.5 BBOB

This directory includes the C version of BBOB challenge program, Beta version 0.9 that is available on `http://coco.gforge.inria.fr/doku.php?id=bbob-2009`. BBOB is the GECCO 2009 benchmark test suite and also provides an excellent test bed for algorithm development and comparison.

4.2.6 bfgs

This directory comprises Nocedal's ACM TOMS implementation of the Limited Memory BFGS (L-BFGS) algorithm in the version that is included in GMIN (see `http://www-wales.ch.cam.ac.uk/GMIN/` for details). It is wrapped by a pCMALib module.

4.2.7 energy_landscapes

The folder **energy_landscapes** currently contains potential energy functions for Lennard-Jones clusters and TIPnP water clusters. With slight modification these files are equivalent to the ones included in GMIN (see `http://www-wales.ch.cam.ac.uk/GMIN/` for details). Users that want to implement and test different cluster energies should include their source files here.

5 Getting started

5.1 System requirements

We summarize the necessary and optional system requirements for pCMALib here.

5.1.1 Platform

The current implementation of pCMALib has been successfully tested on desktop computers with Windows XP SP2, Windows Vista, Windows 7, Mac OS X 10.4.x, Ubuntu Linux 8.10 and OpenSolaris 2008.11. Calculations on clusters have been successfully conducted on Gentoo Linux 2.6.25 and Redhat Linux CentOS 5.4. Both 32 bit and 64 bit architectures have been tested. However, different combinations of Fortran compilers and MPI distributions have been used on the various machines, and hence, platform-independence cannot be guaranteed.

5.1.2 Compiler

The **recommended** compiler for pCMALib is the Intel Fortran compiler, version 9.1or higher. Successful compilation with PGI fortran and gfortran is not guaranteed at the current stage. Before compilation a number of preprocessing statements have to be resolved. We recommend to use the C preprocessor `cpp`. When using the BBOB test suite a C compiler is needed. We tested both the GNU C compiler `gcc` and Intel's `icc`.

5.1.3 LAPACK/BLAS

Using pCMALib requires a working LAPACK/BLAS installation. We both tested the LAPACK available at `http://www.netlib.org/lapack/` and the one included in the Intel MKL (Math Kernel Library). Benchmark runs revealed that, on Intel processsors, MKL's LAPACK is considerably faster [16] and should be used.

5.1.4 MPI

Running pCMALib in the parallel setting requires the installation of an MPI library. We tested and recommend both the OpenMPI (1.2.6, 1.2.8, 1.3) and the Intel MPI library. Again, running pCMALib with Intel MPI jointly with Intel MKl on Intel cores gives superior performance [16].

5.2 Compiling pCMALib

Before compilation, some system-specific configurations have to be set. This is achieved by adapting variables in the one of the provided `make*.inc` files.

Listing 1: Snippet of the make.inc file

```
############################################################
# PCMALIB make include file
# December 2009
#
# Georg Ofenbeck,
# MOSAIC Group, ETH Zurich, Switzerland
############################################################

# Name of the program
LIB_CMA := libpcma.a

######### SHELL OPTIONS ################

SHELL    := /bin/sh

######## OUTPUT Folder ################
# will create directories BUILDDIR/bin, BUILDDIR/objects and BUILDDIR/include here
BUILDDIR:= .
...
```

We have a specific `make_brutus.inc` for the ETH Zurich cluster Brutus and a user-specific `make.inc`. There, the user can specify all relevant variables, e.g., the name of the executable by `LIB_CMA := libpcma.a`, the directory where to build the object and binary files, the FORTRAN compiler, variables for location of LAPACK/MPI/MATLAB distribution etc.. The `make.inc` is included in the `makefile`.

Listing 2: Beginning of the makefile

```
#-------------------------------------------------------------------------------
# pCMALib: a parallel fortran 90 library for the evolution strategy with
#          covariance matrix adaptation
# Christian L. Mueller, Benedikt Baumgartner, Georg Ofenbeck
# MOSAIC group, ETH Zurich, Switzerland
#-------------------------------------------------------------------------------
include make.inc

# Default value for the dependency pre-processor
# = same as code-generating pre-processor
DEPCPP ?= $(CPP)
$(CPP) = cpp
...
```

The makefile first resolves all source dependencies and preprocessor statements. Then, the resulting sources are compiled. The makefile contains four targets:

Listing 3: makefile targets

```
.DEFAULT: ;
# Default target is everything
all: $(TARGET)

...

# at the install part we copy the input files and the programm itself to
install:
        $(shell mkdir $(INSTALL_DIR))
        $(shell mv $(TARGET) $(INSTALL_DIR))
        $(shell cp -r $(CEC2005_DIR)/supportData $(INSTALL_DIR))


# Get rid of the .d's too. Notice this may produce weird behavior during
# the next invocation, since the makefile itself needs the .d's.
clean:
        rm -f $(OBJECTS)
        rm -f $(MODULES)
        rm -f $(TARGET)
        rm -f $(DEPENDENCIES)

# Make new
new: clean all install
```

Execute `make new` in the main folder. The makefile should generate a **objects** and a **bin** folder. In the **bin** folder you will find now the executable with the specified name, e.g., libpcma.a. Depending on your specification this file can now be run on single or multiple cores using `(o)mpirun`. The program is controlled by a single text file as outlined in the next section.

5.3 Controlling pCMALib: the program input file

pCMALib's program is controlled by a input file that specifies all algorithmic settings. The **example_inputs** folder contains several examples. We show here the example used in the Quickstart section where IPOP-CMA-ES is run on the shifted 10D Rastrigin function (Function f9 from the CEC 2005 benchmark test suite [14]).

Listing 4: Example input file

```
#-------------------------------------------------------------------------
# pCMALib: a parallel fortran 90 library for the evolution strategy with
#          covariance matrix adaptation
# Christian L. Mueller, Benedikt Baumgartner, Georg Ofenbeck
# MOSAIC group, ETH Zurich, Switzerland
#-------------------------------------------------------------------------

# output data is saved into this folder (relative to workdir)
OUTPUT_FOLDER = rast_ipop
# which function of the CEC2005 or BBOB benchmark suite to use
BENCHFCTNR = 9
# Dimension of the problem
DIMENSIONS = 10
# Upper bounds on all dimensions
ALLDIM_UBOUNDS = 5
# Lower bounds on all dimensions
ALLDIM_LBOUNDS = -5
#the global optimum
GLOBAL_MIN = -330
# use the CEC2005 benchmark suite as target function
USE_CEC = true
# usage of Quasi Random Sampling
QR_SAMPLING = true
#  this only works with Sobol R implementation! (0) no scrambling
# (1)owen type scrambling, (2)faure-tezuka type scrambling,
# (3)owen,faure-tezuka type scrambling
QR_SCRAMBLING = 0
# (0)Moros Inverse, (1)Peter J. Acklam.s Inverter, (2)Inverter from R
QR_INVERTER = 1
#  (0)Sobol, (1)Sobol R implementation, (2)Halton, (3)Halton R
#  implementation, (4)Faure (buggy!), (5)Niederreiter
QR_SAMPLER = 1
# Successful run if global_min -f(x) < accuracy
ACCURACY = 1.E-8
# if multi restart CMA (IPOP) should be used
RESTART_CMA = true
# (0) restart randomly within bounds, (1) restart from point of
# convergence, (2) restart from same startpoint all the time
RESTART_TYPE = 1
# factor by which the population size is increased every restart
INCPOPSIZE = 1.3
#the folder where to find the supportData folder
CECFOLDERS = ./
```

It is important that the input parameters are spelled correctly. There is no parser included so far that checks input parameters for correct spelling. Hence, wrongly spelled input parameters are ignored.

As in the MATLAB version of CMA-ES there are many options that can be set. The following tables summarize all available options. In the code, most of the default settings are set in `libcma/cmaes_opts_mod.f90`.

Name	Default Value	Explanation
Stop Criteria		
STOPFITNESS	-Inf	Stop if $f(x) <$ StopFitness
STOPMAXFUNEVALS	Inf	Maximal number of FES
STOPMAXITER	$\frac{1e3\,(n+5)^2}{\sqrt{\text{PopSize}}}$	Maximal number of iterations
STOPTOLX	$1e\text{-}11\,max(\sigma_{initial})$	Stop if x-change < StopTolX
STOPTOLUPX	$1e3\,max(\sigma_{initial})$	Stop if x-change > StopTolUpX
STOPTOLFUN	$1e$-12	Stop if fun-change < StopTolFun
STOPTOLHISTFUN	$1e$-13	Stop if back fun-change < StopTolFun
STOPTIME	true	usage of a stop time
STOPTIMEHH	0	stop after given hours
STOPTIMEMM	0	stop after given minutes
STOPTIMESS	0	stop after given seconds
STOPONWARNINGS	true	stop if any of the warnings occur
CMA-ES Strat. Params.		
ABS_SIGMA	0	size of the inital σ as absolut value
REL_SIGMA	0.2	size of the inital σ relative to the size of the box constraints - only used if ABS_SIGMA is not set
POPSIZE	$4 + \lfloor 3 \ln(n) \rfloor$	Population Size λ
PARENTNUMBER	$\lfloor \frac{\lambda}{2} \rfloor$	Parent Number μ
RECOMBINATIONWEIGHTS	3	Super–linear (3), linear (2) or equal (1)
PS-CMA-ES Strat. Params.		
PSCMA	false	Switch PS-CMA-ES on or off
PSOWEIGHT	0.7	weights between PSO-based and local covariance matrix (equivalent to c_{p} in 5)
PSOFREQ	200	Intervall length I_{c} between PSO updates (see 3.2.3)
Sampling Options		
QR_SAMPLING	true	usage of Quasi Random sampling
QR_SAMPLER	1	(0)Sobol, (1)Sobol R implementation (2)Halton (3)Halton R implementation (4)Faure (buggy!) (5)Niederreiter
QR_SCRAMBLING	0	this only works with Sobol R implementation! (0)no scrambling (1)owen type scrambling (2)faure-tezuka type scrambling (3)owen,faure-tezuka type scrambling
QR_INVERTER	1	(0)Moros Inverse (1)Peter J. Acklam's Inverter (2)Inverter from the R Implementation
Restart Settings		
RESTART_CMA	false	Flag if restart CMA (IPOP) should be used
RESTART_TYPE	0	(0) restart randomly within bounds (1) restart from point of convergence (2) restart from same startpoint all the time
RESTARTS	0	limit on how many restarts are allowed, 0 = unlimited
INCPOPSIZE	1.25	factor by which the population size is increased every restart
MAXINCFAC	100	the maximum fold increase of the population compared to the initial population size
Benchmark Opts.		
BENCHMARK	false	Switch Benchmark on or off, this causes pCMALib to record and keep track of several variables that are required in the CEC 2005 benchmark protocols
RECORD_ACCURACY	0.0	record when the CMA-ES reaches this level of accuracy
RECORD_BESTHIST	false	record a history of the fitness over time
RECORD_MODULO	100	$FE\_MAX/RECORD\_MODULO$ gives the number of records

Table 1: Available pCMALib options

Others		
DIMENSIONS		Dimension n of the problem
GLOBAL_MIN	0.0	Global minimum (if available)
ACCURACY	0.0	Successful run if $\|$GLOBAL_MIN $-$ f(x)$\|$ $\leq$ accuracy
EVALINITIALX	true	Evaluate initial solution
WARNONEQUALFUNCTIONVALUES	true	Report warning if all function values in a generation are identical
FLGGENDATA	true	Flag if complete output data should be generated (can result in huge files!)
INTGENDATA	1	Integer interval to log output data
FLGOUTTXT	true if compiled without matlab, otherwise false	saves the results of the pCMAlib run into textfiles
FLGGENTRACE	false	saves only trace of best solutions and their fitness values into textfiles
FUNCNAME	' '	Name of the function that will be reported in output
VERBOSEMODULO	100	Messaging after every i-th iteration in the console
OUTPUT_FOLDER	'out'	output data is saved into this folder (relative to workdir)
SILENTMPI	true	flag if only process with rank 0 should report output in the console
USE_SEED	false	if a seed for the RNG should be used
SEED_FOLDER	'false'	folder containing the seed file 'seed.txt'
Boundary setting		
ALLDIM_LBOUNDS	-Inf	Lower bounds on all dimensions
ALLDIM_UBOUNDS	Inf	Upper bounds on all dimensions
USE_INIT_BOUNDS	false	if special bounds should be used for initalization of population
INIT_UBOUNDS	Inf	Upper bounds for initialization
INIT_LBOUNDS	-Inf	Lower bounds for initialization
Objective functions		
USE_LJ_COMP	false	use the Lennard Jones potential with compression as target function
USE_DF	false	use the DoubleFunnel benchmark as target function
USE_MATFUNC	false	use the template for a Matlab target as function
USE_RANDOM_LANDSCAPE	false	use the random landscape test as target function
USE_CEC	false	use the CEC2005 benchmark suite as target function
USE_BBOB	false	use the BBOB benchmark suite as target function
USE_LJ	false	use the Lennard Jones potential as target function
USE_TIP	false	use the TIP4P water potential as target function
Add. objective function settings		
WRITE_PDB	false	if a pdb file representing the current best solution is written in intervals according to the VerboseModulo setting
LJ_COMP	1	compression parameter μ_{comp} for the Lennard Jones potential with compression
DF_S	0	setting for parameter 's' for the Double Funnel benchmark
DF_RAST	true	if rastrigin function should be applied to the Double Funnel benchmark
CECFOLDERS	"	where to find the supportData folder for the CEC2005 benchmark suite relative to the working directory
BENCHFCTNR	1	which function of the CEC2005 or BBOB benchmark suite to use

Table 2: Available pCMALib options

BFGS Settings (experimental)		
BFGS_USE	false	if BFGS should be used to assist CMA. This is still in development!
BFGS_FACTR		not used at the moment
BFGS_PGTOL		not used at the moment
BFGS_GRAD_STEPSIZE		step size used for the gradient approximation
BFGS_POSITION	2	1 = replace X values by local minimum X 2 = replace F values with F values at local minimum
BFGS_CENTRAL_DIFFERENCE	false	if central difference should be used, otherwise backward difference is utilized
BFGS_DGUESS		the guess for the inital step size for the line search
BFGS_STPMAX		uper bounds for the step in the line search
BFGS_STPMIN		specify lower for the step in the line search
BFGS_GTOL		controls the accuracy of the line search routine MCSRCH

Table 3: Available pCMALib options

5.4 Output files

There are several modes of output generation in pCMALib which are controlled by the four input parameters FLGGENDATA, INTGENDATA, FLGOUTTXT and FLGGENTRACE. The files will be generated in the folder specified by the input parameter OUTPUT FOLDER.

If the FLGGENDATA is true, pCMALib will write out every INTGENDATA **generation** almost all CMA-ES data, such as e.g. the populations, the covariance matrices, the Cholesky matrices, the evolution path etc. Depending on the dimensionality, the run length and the write out interval length, this may result in **HUGE** text files. So when doing production runs with pCMALib, the user is advised to handle the settings with care. In case MATLAB is available, in addition to the text file, a binary MATLAB file cmaesData.mat is generated that includes the summary data given below but **NOT** the traces along the optimization run.

In case FLGGENDATA is false, the user has the possibility to just write the summary data (see below) in text files by setting FLGOUTTXT true. No MATLAB is required for this.

This is the list of summary output files that are provided. In case MATLAB is available, these data are stored in a structured binary MATLAB .mat file. It follows the standard output from the Hansen's CMA-ES MATLAB version plus some pCMALib specific output.

out_bestever_f.txt	the best fitness value found during the optimization
out_bestever_x.txt	the best input vector x found during the optimization
out_bestever_evals.txt	the evaluation where the best fitness value has been found
out_countEvalNaN.txt	the number of invalid samples drawn during the optimization run
out_countEval.txt	the number of samples drawn during the optimization
out_countIter.txt	the number of CMA Iteration done during the optimization
out_countOutOfBounds.txt	the number of samples that fell outside of the Bounds given to CMA
out_funcName.txt	the name of the function like given in the input file if a benchmark is utilized, it returns the name of the function evaluated
out_insigma.txt	the initial sigma CMA starts with in absolute numbers
out_lambda.txt	the initial lambda CMA starts with
out_mueff.txt	effective population weight
out_mu.txt	number of selected individuals per generation
out_N.txt	dimension of the problem
out_settings.txt	all settings turned on/off by pCMALib. Also those values that are turned off via flags (e.g. QR_SAMPLING = .false. and all the other QR Settings are reported)
out_stopflag.txt	the reason why the CMA optimization stopped (e.g. time limit reached, global minimum found etc.)
out_weights.txt	the weights used for the ranking
out_xstart.txt	the initial x_mean of CMA-ES - not utilized at the moment
seed.txt	the initial random number seed - can be used for reproducing the same results

Table 4: Summary output from pCMALib

Finally, if the user is only interested in the trace of current best candidate solutions and its corresponding fitness values, these can be generated by setting the FLGGENTRACE true. Then, the text files besteverF.txt and besteverX.txt contain these values at every INTGENDATA **generation**.

If pCMALib is run in MPI mode, the output files are generated for each process. The resulting file names are the same as in the single process case but with the extension $\_RANK$, i.e. the cmaesData.mat for the process with rank 0 is called *cmaesData_0.mat*.

For the specific case where LJ or TIPnP water clusters are optimized, the user has the possibility to write out PDB files with the LJ or Water atom coordinates. For this purpose the flag $WRITE\_PDB$ should be set true. The write out frequency is controlled by the input parameter VERBOSEMODULO.

6 Test example

6.1 IPOP-CMA-ES

- Unzip / checkout
- Change all parameters in the make.inc file. MPI and MATLAB Include/Library Paths are not required for this example. Make sure that HAS_MPI, HAS_MAT and BBOB are set to 0.
- in the main folder execute make new
- cd to the newly created bin folder
- execute *./libpcma.a ../example_inputs/rastrigin_ipop.txt*
- you should get output on the console similar to the one listed below

Listing 5: Console output generated by pCMAlib on the rastrigin IPOP example

```
   ****************************************************
Warnings:
n= 10: ( 5 , 10 )-CMA-ES on function Shifted Rastrigins Function
       Iterat ,     #Fevals:    Function Value
          1 ,           12 :  -134.484715127431
        100 ,         1002 :  -319.957567812663
        200 ,         2002 :  -320.050414465281
        262 ,         2622 :  -320.050414466886
        262 ,         2622 :  -320.050414466886
------- Restart #           1  Reason: warnequalfunvals
n= 10: ( 6 , 13 )-CMA-ES on function Shifted Rastrigins Function
-----------------------------------
Initial sigma   2.00000000000000
-----------------------------------
       Iterat ,     #Fevals:    Function Value
        300 ,         3116 :  -260.736124312701
        400 ,         4416 :  -326.020162842026
        473 ,         5365 :  -326.020163771627
        473 ,         5365 :  -326.020163771627
------- Restart #           2  Reason: warnequalfunvals

.
.
.

------- Restart #           2  Reason: warnequalfunvals

n= 10: ( 45 , 91 )-CMA-ES on function Shifted Rastrigins Function
-----------------------------------
Initial sigma   2.00000000000000
-----------------------------------
       Iterat ,     #Fevals:    Function Value
       7000 ,       524992 :  -281.725617082436
```

```
       7071 ,       531453 :   -329.999999995965

GLOBAL Bestever.f:   -329.999999995965
GLOBAL Bestever.x:
     1.900E+00
 -1.564E+00
 -9.788E-01
 -2.254E+00
  2.499E+00
 -3.285E+00
  9.759E-01
 -3.666E+00
  9.850E-02
 -3.246E+00
Stopflag:fitness
```

- *ls rast_test* should yield a list of files generated by the optimization run as shown in the previous section.

6.2 PS-CMA-ES

- Unzip / checkout
- Change all parameters in the make.inc file. MPI Paths are required for this example. Make sure that HAS_MAT and BBOB are set to 0, while HAS_MPI is set to 1
- in the main folder execute make new
- cd to the newly created bin folder
- execute *mpirun -n 4 ./libcma ../example_inputs/water_pscma.txt*. This command structure might vary depending on the installed MPI software.
- you should get output on the console similar to the one listed below

Listing 6: Console output generated by pCMAlib on the rastrigin IPOP example

```
*****************************************************
Warnings:
*****************************************************
Started MPI-CMA
To guarantee a decent console output only Process 0 is shown
 All output data is saved to
folder water_pscma


*****************************************************
-----------------------------------
PSO configuration:
Weight:    0.800000000000000
Frequency:         1000
-----------------------------------
Initial sigma    2.40000000000000
-----------------------------------
n= 48: ( 7 , 15 )-CMA-ES on function
        Iterat ,     #Fevals:     Function Value
           1 ,           17 :   -24.7688838771480
         100 ,         1502 :   -91.9960387430328
         200 ,         3002 :   -141.671714969162
         300 ,         4502 :   -149.296251112378
         400 ,         6002 :   -194.206246961448
         500 ,         7502 :   -203.078431185465
```

```
         600  ,         9002  :   -207.834413560923
         700  ,        10502  :   -208.866055139447
         800  ,        12002  :   -209.640611386233
         900  ,        13502  :   -210.402527129655
        1000  ,        15002  :   -211.520111342927
        1100  ,        16502  :   -212.422564260838
        1200  ,        18002  :   -212.949637423201
        1300  ,        19502  :   -213.668174681497
        1400  ,        21002  :   -226.061157943448
        1500  ,        22502  :   -237.363561233379
        1600  ,        24002  :   -239.168766472368
.
.
.
 GLOBAL Bestever.f:   -243.051022815494
GLOBAL Bestever.x:
     1.423E+00
  1.402E+00
 -2.535E+00
 -2.564E-01
 -6.437E-01
 -3.328E+00
 -8.623E-01
 -9.006E-01
 -6.475E-01
 -9.176E-01
  1.923E+00
 -2.113E-01
  2.389E+00
 -6.220E-01
 -4.128E+00
 -1.737E+00
 -2.249E-01
 -5.593E+00
  2.254E-01
  1.494E+00
  2.318E+00
  1.417E+00
  4.412E-01
  4.009E-02
  2.217E+00
  6.194E-01
  9.304E-01
 -4.445E-01
  1.224E-01
  3.384E+00
 -2.908E+00
 -3.839E+00
  5.877E+00
  2.826E+00
 -3.180E+00
 -2.960E+00
  3.403E+00
 -1.536E+00
 -9.409E-01
 -2.305E-01
 -3.482E+00
 -6.000E+00
  9.542E-01
  4.354E+00
  1.263E+00
 -2.616E+00
 -1.269E+00
```

```
  -1.310E+00
Stopflag : tolfun
```

- *ls water_pscma* should yield a list of files generated by the optimization run as shown in the previous section, but this time one file for each process should exist of the form FILENAME_{Process#.txt}

7 Adding new objective functions

Adding user-specific objective functions is rather simple. In the user folder you have a template that you can adapt for your needs. If the name of the template function is renamed, you have to rename `EXTERNAL user_function` in `cmaes.f90` and its call further down in the main as well. Further test functions can also be located in either **testfcns** or the **energy_landscapes** folder. They also have to be known as `EXTERNAL user_function` in `cmaes.f90` in order to work properly.

Listing 7: Snippet of the main.f90

```
!-----------------------------------------------------------------------
!   Routine         :              user_function
!-----------------------------------------------------------------------
!
!   Purpose       :
!
!   Remark        : at the moment pcmalib will only call single values at once
!                   meaning it will send 'vars' of size m x 1 and expect a vector of size
!                   n=1 back. This might change in the future and therefore it is recommended
!                   to write the function in a way that it can handle multiple input vectors in form
!                   of a matrix and return a vector of results
!
!   Input         :
!                      m        (I) m Dimension of the matrix (Rows)
!                                   = Dimension of Vectors
!                      n        (I) n Dimension of the matrix (Columns)
!                                   = # of Vectors to process
!   Input(optional): lbounds / ubounds - m dimensional array with boundaries given to CMA-ES
!
!
!   Input/Output:   vars     (R) the matrix with the input values of size m*n
!
!   Output:         res      (R) the vector with the results of size n
!
!-----------------------------------------------------------------------


SUBROUTINE user_function(res,vars,m,n,lbounds,ubounds)
USE cmaes_param_mod
!-----------------------------------------------------------------------
!   Parameters
!-----------------------------------------------------------------------
REAL(MK),DIMENSION(n),INTENT(out)                :: res
REAL(MK),DIMENSION(m,n),INTENT(in)               :: vars
INTEGER,INTENT(in)                               :: m
INTEGER,INTENT(in)                               :: n
REAL(MK),DIMENSION(m),OPTIONAL                   :: lbounds
REAL(MK),DIMENSION(m),OPTIONAL                   :: ubounds


!your code here!!!!!
WRITE(*,*) 'no_code_provided_in_the_user_function_yet'
STOP


END SUBROUTINE user_function
```

As can be seen from the current template for objective functions it is not possible to provide additional parameters to the objective functions explicitly. If the user function needs additional parameters they have to be provided via global/MODULE variables. An example how to provide function parameters can be found in the double funnel test function case (USE_DF). This objective function has a parameter s that tunes the size and depth ratio of two funnels. It is provided with the input parameter DF_S in the input file.

8 Known issues

- Scrambling caused some troubles on the ETH cluster after some million iterations which could not be reproduced on a local machine (returning 0 values for all samples)
- Ipop (Restart) settings can cause troubles if not limited by the MAXINCFAC control input. This is due to the exponential increase in population size and the therefore linked exponential increase in memory requirements

- the matlab engine is not the most reliable - it happened sometimes that the program would just freeze during the write out of the .mat file
- very long paths (>200 characters) are cut off and might cause troubles
- if the FLGGENDATA is set to true and INTGENDATA small, the output files can become very big due to the fact that it is written as text file
- after the setting file (make.inc) is altered, manual cleaning of the previous make run might be required (deleting the objects folder manually)

9 License and further reading

ToDo

References

[1] Nikolaus Hansen and Andreas Ostermeier. Completely Derandomized Self-Adaption in Evolution Strategies. *Evolutionary Computation*, 9(2):159–195, 2001.

[2] Nikolaus Hansen, Sibylle D. Müller, and Petros Koumoutsakos. Reducing the Time Complexity of the Derandomized Evolution Strategy with Covariance Matrix Adaptation (CMA-ES). *Evolutionary Computation*, 11(1):1–18, 2003.

[3] Nikolaus Hansen and Stefan Kern. Evaluating the CMA Evolution Strategy on Multimodal Test Functions. In *Lecture Notes in Computer Science*, volume 3242 of *Parallel Problem Solving from Nature - PPSN VIII*, pages 282–291, Berlin, Heidelberg, 2004. Springer.

[4] Nikolaus Hansen. The CMA Evolution Strategy. http://www.lri.fr/ hansen/cmaesintro.html.

[5] Nikolaus Hansen. The CMA Evolution Strategy: A Tutorial. http://www.lri.fr/ hansen/cmatutorial.pdf, 2007.

[6] Anne Auger and Nikolaus Hansen. Performance Evaluation of an Advanced Local Search Evolutionary Algorithm. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2005)*, volume 2, pages 1777–1784, 2005.

[7] Anne Auger and Nikolaus Hansen. A Restart CMA Evolution Strategy with Increasing Population Size. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2005)*, volume 2, pages 1769–1776, 2005.

[8] Enrique Alba. *Parallel Metaheuristics: A New Class of Algorithms*. Wiley-Interscience, 2005.

[9] Christian L. Müller, B. Baumgartner, and I. F. Sbalzarini. Particle Swarm CMA Evolution Strategy for the Optimization of Multi-Funnel Landscapes. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2009)*, 2009.

[10] Gene H. Golub and Charles F. Van Loan. *Matrix Computations*. Johns Hopkins University Press, 3rd edition, 1996.

[11] Günther Rudolph. On Correlated Mutations in Evolution Strategies. In R. Männer and B. Manderick, editors, *Parallel Problem Solving from Nature 2 (Proc. 2nd Int. Conf. on Parallel Problem Solving from Nature, Brussels 1992)*, pages 105–114, Amsterdam, 1992. Elsevier.

[12] Olivier Teytaud and Sylvain Gelly. DCMA: yet another derandomization in covariance-matrix-adaptation. In *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 955–963, New York, NY, USA, 2007. ACM.

[13] David J. Wales and Jonathan P. K. Doye. Global Optimization by Basin-Hopping and the Lowest Energy Structures of Lennard-Jones Clusters Containing up to 110 Atoms. *The Journal of Physical Chemistry A*, 101(28):5111–5116, 1997.

[14] P. N. Suganthan, N. Hansen, J. J. Liang, K. Deb, Y.-P. Chen, A. Auger, and S. Tiwari. Problem Definitions and Evaluation Criteria for the CEC 2005 Special Session on Real-Parameter Optimization. Technical report, Nanyang Technological University, Singapore, May 2005.

[15] Monte Lunacek, Darrell Whitley, and Andrew Sutton. The Impact of Global Structure on Search. In *Proceedings of the 10th international conference on Parallel Problem Solving from Nature*, pages 498–507, Berlin, Heidelberg, 2008. Springer-Verlag.

[16] Christian L. Müller. Benchmark runs of pCMALib on Nehalem and Shanghai nodes. Technical report, ETH Zurich, 2009.

[17] Lawrence Livermore National Laboratory. Message Passing Interface (MPI). https://computing.llnl.gov/tutorials/mpi/, September 2007.

APPENDIX

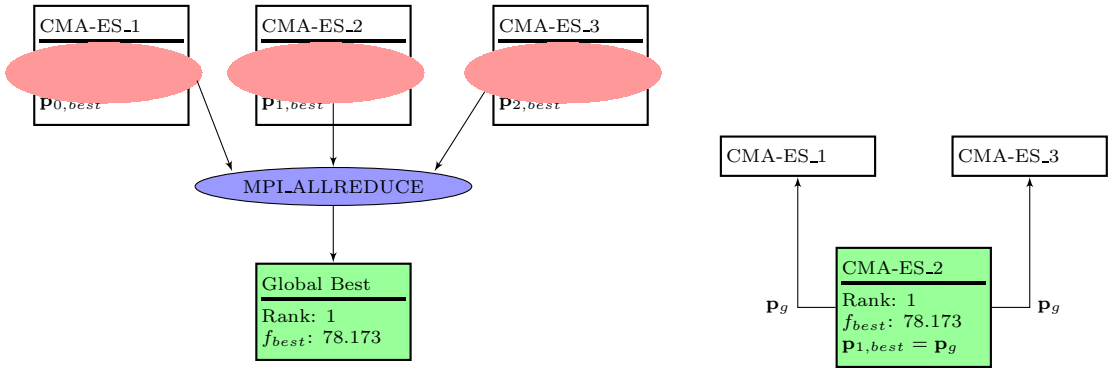
A MPI structure in PS-CMA-ES

A.1 MPI protocol for PS-CMA-ES

The parallel CMA-ES code is developed based on MPI. Multiple CMA-ES instances need to communicate in order to integrate parallel inofrmation. MPI is designed to tackle such tasks on an implementational level. Each CMA-ES instance represents a computational process with separate address spaces. Via MPI, these processes can be distributed to multiple processors (for example to different PCs in a Local Area Network (LAN)). If such a topology is not available, MPI allows to execute parallel code on single processors.
Unfortunately, the developed algorithm is computationally demanding. Especially when multiple CMA-ES units have to share hardware facilities, computation speed decreases. Therefore, an elaborate communication strategy is essential to reduce complexity.

A.2 GLOBAL_BEST-Communication

In MPI, each process is assigned a unique number: the process rank. Multiple processes can be distinguished by their rank. In section 3 it has been outlined, that each CMA-ES instance has to inform the other swarm members on its current best candidate solution. Figure 6 illustrates our procedure. A 2-dimensional array, called F_BEST, is introduced. It holds the current best fitness value, as well as the process rank (illustrated by light red ellipses in Figure 6(a)). The MPI collective communication routine *MPI_ALLREDUCE* is able to find the global best function value within all F_BEST arrays. Since the array also contains the process rank, the corresponding CMA-ES instance is known. In a second step (Figure 6(b)), this process broadcasts the position of its current optimal solution to the other swarm members.
To reduce communication expenses, the broadcast is only performed, when the current optimum has changed.



(a) MPI Allreduce (b) MPI Broadcast

Figure 6: MPI Communication of the Global Best Position $\mathbf{p}_g$

A.3 Excluding processes from communication

If a CMA-ES instance has converged and stopped its search, it needs to be excluded from communication. There are two possible scenarios, how a safe program termination can be ensured, and a communication deadlock can be avoided. One is, that the process waits until all other running processes have reached the same state, such that all processes can be finalized synchronously. Such an approach heavily influences performance, because resources keep on being allocated, until the last process has converged.
Therefore, another, more dynamic approach is favoured: whenever a process stops, communication is adapted, such that this specific process is excluded. To describe our method, definitions of MPI groups and communicators have to be given [17]:

Definition A.3.1 *A **group** is an ordered set of processes. Each process in a group is associated with a unique integer rank. Rank values start at zero and go to N-1, where N is the number of processes in the group. A group is always associated with a communicator object.*

Definition A.3.2 *A **communicator** encompasses a group of processes that may communicate with each other. All MPI messages must specify a communicator. For example, the communicator that comprises all tasks is MPI_COMM_WORLD.*

In MPI, communicators can not be adapted directly. Therefore, a little workaround is needed to adapt communications. At the end of each CMA-ES generation, it is checked, whether one or more processes have met a stopping criterion and are about to terminate. If this is the case, the following rules are applied:

1. Ranks of terminating processes are collected.
2. *MPI_COMM_GROUP()* and *MPI_GROUP_EXCL()* are used to build a new communication group, that excludes the specified ranks.
3. A new rank for each process in the group (*MPI_GROUP_RANK()*) is assigned.
4. A group communicator using *MPI_COMM_CREATE()* is created.
5. Calculations using the newly created communicator are continued. All processes, that are not in the scope of the communicator, are terminated.

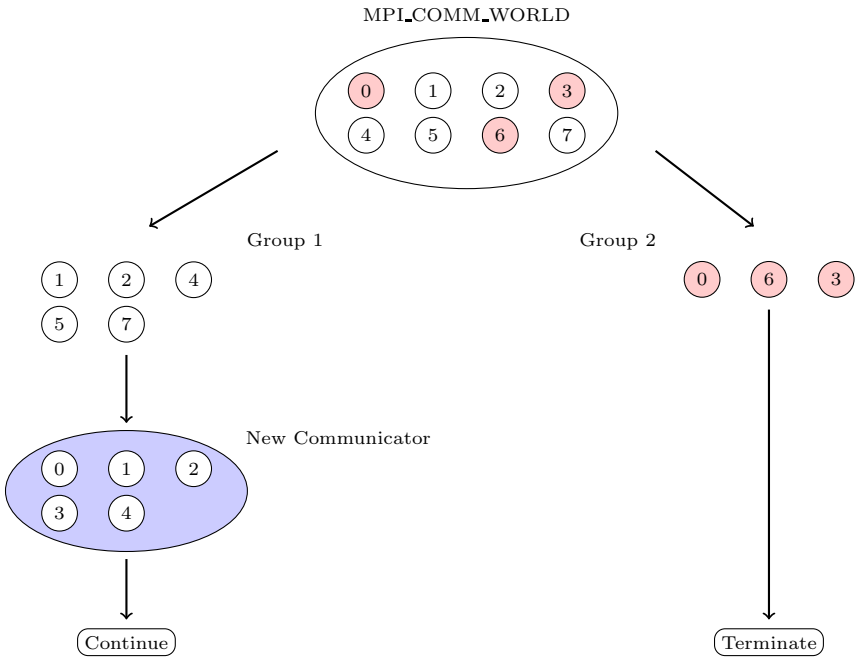


Figure 7: Excluding processes from communication

Figure 7 illustrates an example for the MPI group and communicator management. At the very top of the figure, all processes are within the same communicator environment (MPI_COMM_WORLD). Colored in a light red are processes that will stop. Following the chart, two groups are formed. One group contains running processes (Group 1), the other contains stopping processes (Group 2)[2]. Based on Group 1, a new communicator can be created. Note, that the processes are assigned a new rank starting from 0 again.

[2]Group 2 is not a valid communication group. Instead it contains processes with rank value *MPI_UNDEFINED*