# LM-CMA: an Alternative to L-BFGS for Large Scale Black-box Optimization

Ilya Loshchilov

# LM-CMA: an Alternative to L-BFGS for Large Scale Black-box Optimization

**Ilya Loshchilov** ilya.loshchilov@gmail.com
Laboratory of Intelligent Systems (LIS)
École Polytechnique Fédéral de Lausanne (EPFL), Lausanne, Switzerland

**Abstract**

The limited memory BFGS method (L-BFGS) of Liu and Nocedal (1989) is often considered to be the method of choice for continuous optimization when first- and/or second- order information is available. However, the use of L-BFGS can be complicated in a black-box scenario where gradient information is not available and therefore should be numerically estimated. The accuracy of this estimation, obtained by finite difference methods, is often problem-dependent that may lead to premature convergence of the algorithm.

In this paper, we demonstrate an alternative to L-BFGS, the limited memory Covariance Matrix Adaptation Evolution Strategy (LM-CMA) proposed by Loshchilov (2014). The LM-CMA is a stochastic derivative-free algorithm for numerical optimization of non-linear, non-convex optimization problems. Inspired by the L-BFGS, the LM-CMA samples candidate solutions according to a covariance matrix reproduced from $m$ direction vectors selected during the optimization process. The decomposition of the covariance matrix into Cholesky factors allows to reduce the memory complexity to $O(mn)$, where $n$ is the number of decision variables. The time complexity of sampling one candidate solution is also $O(mn)$, but scales as only about 25 scalar-vector multiplications in practice. The algorithm has an important property of invariance w.r.t. strictly increasing transformations of the objective function, such transformations do not compromise its ability to approach the optimum. The LM-CMA outperforms the original CMA-ES and its large scale versions on non-separable ill-conditioned problems with a factor increasing with problem dimension. Invariance properties of the algorithm do not prevent it from demonstrating a comparable performance to L-BFGS on non-trivial large scale smooth and nonsmooth optimization problems.

**Keywords**

LM-CMA, L-BFGS, CMA-ES, large scale optimization, black-box optimization.

## 1 Introduction

In a black-box scenario, knowledge about an objective function $f : \boldsymbol{X} \to \mathbb{R}$, to be optimized on some space $\boldsymbol{X}$, is restricted to the handling of a device that delivers the value of $f(\boldsymbol{x})$ for any input $\boldsymbol{x} \in \boldsymbol{X}$. The goal of black-box optimization is to find solutions with small (in the case of minimization) value $f(\boldsymbol{x})$, using the least number of calls to the function $f$ (Ollivier et al., 2011). In continuous domain, $f$ is defined as a mapping $\mathbb{R}^n \to \mathbb{R}$, where $n$ is the number of variables. The increasing typical number of variables involved in everyday optimization problems makes it harder to supply the search with useful problem-specific knowledge, e.g., gradient information, valid assumptions about problem properties. The use of large scale black-box optimization approaches would seem attractive providing that a comparable performance can be achieved.

The use of well recognized gradient-based approaches such as the Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm (Shanno, 1970) is complicated in the black-box scenario since gradient information is not available and therefore should be estimated by costly finite difference methods (e.g., $n+1$ function evaluations per gradient estimation for forward difference and $2n+1$ for central difference). The latter procedures are problem-sensitive and may require a priori knowledge about the problem at hand, e.g., scaling of $f$, decision variables and expected condition number (Li et al., 2007).

By the 1980s, another difficulty has become evident: the use of quasi-Newton methods such as BFGS is limited to small and medium scale optimization problems for which the approximate inverse Hessian matrix can be stored in memory. As a solution, it was proposed not to store the matrix but to reconstruct it using information from the last $m$ iterations (Nocedal, 1980). The final algorithm called the limited memory BFGS algorithm (L-BFGS or LM-BFGS) proposed by Liu and Nocedal (1989) is still considered to be the state-of-the-art of large scale gradient-based optimization (Becker and Fadili, 2012). However, when a large scale black-box function is considered, the L-BFGS is forced to deal both with a scarce information coming from only $m$ recent gradients and potentially numerically imprecise estimations of these gradients which scale up the run-time in the number of function evaluations by a factor of $n$. It is reasonable to wonder whether the L-BFGS and other derivative-based algorithms are still competitive in these settings or better performance and robustness can be achieved by derivative-free algorithms.

The Covariance Matrix Adaptation Evolution Strategy (CMA-ES) seems to be a reasonable alternative, it is a derivative-free algorithm designed to learn dependencies between decision variables by adapting a covariance matrix which defines the sampling distribution of candidate solutions (Hansen et al., 2003). This algorithm constantly demonstrates good performance at various platforms for comparing continuous optimizers such as the Black-Box Optimization Benchmarking (BBOB) workshop (Finck et al., 2010; Auger et al., 2010; Loshchilov et al., 2013) and the Special Session at Congress on Evolutionary Computation (García et al., 2009; Loshchilov, 2013a). The CMA-ES was also extended to noisy (Hansen et al., 2009), expensive (Kern et al., 2006; Loshchilov et al., 2012) and multi-objective optimization (Igel et al., 2007).

The principle advantage of CMA-ES, the learning of dependencies between $n$ decision variables, also forms its main practical limitations such as $O(n^2)$ memory storage required to run the algorithm and $O(n^2)$ computational time complexity per function evaluation (Ros and Hansen, 2008). These limitations may preclude the use of CMA-ES for computationally cheap but large scale optimization problems if the internal computational cost of CMA-ES is greater than the cost of one function evaluation. On non-trivial large scale problems with $n > 10,000$ not only the internal computational cost of CMA-ES becomes enormous but it is becoming simply impossible to efficiently store the covariance matrix in memory. An open problem is how to extend efficient black-box approaches such as CMA-ES to $n \gg 1000$ while keeping a reasonable trade-off between the performance in terms of the number of function evaluations and the internal time and space complexity. The low complexity methods such as separable CMA-ES (sep-CMA-ES by Ros and Hansen (2008)), linear time Natural Evolution Strategy (R1-NES by Sun et al. (2011)) and VD-CMA by Akimoto et al. (2014) are useful when the large scale optimization problem at hand is separable or decision variables are weakly correlated, otherwise the performance of these algorithms w.r.t. the original CMA-ES may deteriorate significantly.

In this paper, we present a greatly improved version of the recently proposed extension of CMA-ES to large scale optimization called the limited memory CMA-ES (LM-CMA) by Loshchilov (2014). Instead of storing the covariance matrix, the LM-CMA stores $m$ direction vectors in memory and uses them to generate solutions. The algorithm has $O(mn)$ space complexity, where $m$ can be a function of $n$. The time complexity is linear in practice with the smallest constant factor among the presented evolutionary algorithms.

The paper is organized as follows. First, we briefly describe L-BFGS in Section 2 and CMA-ES with its large scale alternatives in Section 3. Then, we present the improved version of LM-CMA in Section 4, investigate its performance w.r.t. large scale alternatives in Section 5 and conclude the paper in Section 6.

## 2 The L-BFGS

An early version of the L-BFGS method, at that time called the SQN method, was proposed by Nocedal (1980). During the first $m$ iterations the L-BFGS is identical to the BFGS method, but stores BFGS corrections separately until the maximum number of them $m$ is used up. Then, the oldest corrections are replaced by the newest ones. The approximate of the inverse Hessian of $f$ at iteration $k$, $\boldsymbol{H}_k$ is obtained by applying $m$ BFGS updates to a sparse symmetric and positive definite matrix $\boldsymbol{H}_0$ provided by the user (Liu and Nocedal, 1989).

Let us denote iterates by $\boldsymbol{x}_k$, $\boldsymbol{s}_k = \boldsymbol{x}_{k+1} - \boldsymbol{x}_k$ and $\boldsymbol{y}_k = \boldsymbol{g}_{k+1} - \boldsymbol{g}_k$, where $\boldsymbol{g}$ denotes gradient. The method uses the inverse BFGS formula in the form

$$\boldsymbol{H}_{k+1} = \boldsymbol{V}_k^T \boldsymbol{H}_k \boldsymbol{V}_k + \rho_k \boldsymbol{s}_k \boldsymbol{s}_k^T, \tag{1}$$

where $\rho_k = 1/\boldsymbol{y}_k^T \boldsymbol{s}_k$, and $\boldsymbol{V}_k = \boldsymbol{I} - \rho_k \boldsymbol{y}_k \boldsymbol{s}_k^T$ (Dennis Jr and Schnabel, 1996; Liu and Nocedal, 1989).

The L-BFGS method works as follows (Liu and Nocedal, 1989):

*Step 1*. Choose $\boldsymbol{x}_0$, $m$, $0 < \beta' < 1/2$, $\beta' < \beta < 1$, and a symmetric and positive definite starting matrix $\boldsymbol{H}_0$. Set $k = 0$.

*Step 2*. Compute

$$\boldsymbol{d}_k = -\boldsymbol{H}_k \boldsymbol{g}_k, \tag{2}$$

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{d}_k, \tag{3}$$

where $\alpha_k$ satisfies the Wolfe conditions (Wolfe, 1969):

$$f(\boldsymbol{x}_k + \alpha_k \boldsymbol{d}_k) \leq f(\boldsymbol{x}_k) + \beta' \alpha_k \boldsymbol{g}_k^T \boldsymbol{d}_k, \tag{4}$$

$$g(\boldsymbol{x}_k + \alpha_k \boldsymbol{d}_k)^T \boldsymbol{d}_k \geq \beta \boldsymbol{g}_k^T \boldsymbol{d}_k. \tag{5}$$

The novelty introduced by Liu and Nocedal (1989) w.r.t. the version given in Nocedal (1980) is that the line search is not forced to perform at least one cubic interpolation, but the unit steplength $\alpha_k = 1$ is always tried first, and if it satisfies the Wolfe conditions, it is accepted.

*Step 3*. Let $\hat{m} = \min(k, m-1)$. Update $\boldsymbol{H}_0$ $\hat{m}+1$ times using the pairs $\left\{\boldsymbol{y}_j, \boldsymbol{s}_j\right\}_{j=k-\hat{m}}^{k}$ as follows:

$$\begin{aligned}\boldsymbol{H}_{k+1} = &(\boldsymbol{V}_k^T \cdots \boldsymbol{V}_{k-\hat{m}}^T)\boldsymbol{H}_0(\boldsymbol{V}_{k-\hat{m}} \cdots \boldsymbol{V}_k)\\ &+\rho_{k-\hat{m}}(\boldsymbol{V}_k^T \cdots \boldsymbol{V}_{k-\hat{m}+1}^T)\boldsymbol{s}_{k-\hat{m}}\boldsymbol{s}_{k-\hat{m}}^T(\boldsymbol{V}_{k-\hat{m}+1} \cdots \boldsymbol{V}_k)\\ &+\rho_{k-\hat{m}+1}(\boldsymbol{V}_k^T \cdots \boldsymbol{V}_{k-\hat{m}+2}^T)\boldsymbol{s}_{k-\hat{m}+1}\boldsymbol{s}_{k-\hat{m}+1}^T(\boldsymbol{V}_{k-\hat{m}+2} \cdots \boldsymbol{V}_k)\\ &\vdots\\ &+\rho_k \boldsymbol{s}_k \boldsymbol{s}_k^T\end{aligned}$$

*Step 4*. Set $k = k + 1$ and go to Step 2.

The algorithm space and time complexity scales as $O(mn)$ per iteration (not per function evaluation), where $m$ in order of 5-40 suggested in the original paper is still the most common setting. An extension to bound constrained optimization called L-BFGS-B has the efficiency of the original algorithm, however at the cost of a significantly more complex implementation (Byrd et al., 1995). Extensions to optimization with arbitrary constraints are currently not available. Satisfactory and computationally tractable handling of noise is at least problematic, often impossible.

Nevertheless, as already mentioned above, when gradient information is available, L-BFGS is competitive to other techniques (Becker and Fadili, 2012; Ngiam et al., 2011) and often can be viewed as a method of choice (Andrew and Gao, 2007) for large scale continuous optimization. However, in the black-box scenario when gradient-information is not available (direct search settings), the advantages of L-BFGS are becoming less obvious and derivative-free algorithms can potentially perform comparable. In this paper, we investigate this scenario in detail.

## 3 Evolution Strategies for Large Scale Optimization

Historically, first Evolution Strategies (Rechenberg, 1973) were designed to perform the search without learning dependencies between variables which is a more recent development that gradually led to the CMA-ES algorithm (Hansen and Ostermeier, 1996; Hansen et al., 2003). In this section, we discuss the CMA-ES algorithm and its state-of-the-art derivatives for large scale optimization. For a recent comprehensible overview of Evolution Strategies, the interested reader is referred to Hansen et al. (2015). More specifically, the analysis of theoretical foundations of Evolution Strategies is provided by Wierstra et al. (2014); Ollivier et al. (2011); Akimoto and Ollivier (2013); Glasmachers (2012); Auger and Hansen (2013); Hansen and Auger (2014); Arnold (2014); Beyer (2014).

### 3.1 The CMA-ES

The Covariance Matrix Adaptation Evolution Strategy (Hansen and Ostermeier, 1996, 2001; Hansen et al., 2003) is probably the most popular and in overall the most efficient Evolution Strategy.

The $(\mu/\mu_w, \lambda)$-CMA-ES is outlined in **Algorithm** 1. At iteration $t$ of CMA-ES, a mean $\boldsymbol{m}^t$ of the mutation distribution (can be interpreted as an estimation of the optimum) is used to generate its $k$-th out of $\lambda$ candidate solution $\boldsymbol{x}_k \in \mathbb{R}^n$ (line 5) by adding a random Gaussian mutation defined by a (positive definite) covariance matrix

**Algorithm 1** The $(\mu/\mu_w, \lambda)$-CMA-ES

```
1: given $n \in \mathbb{N}_+$, $\lambda = 4 + \lfloor 3 \ln n \rfloor$, $\mu = \lfloor \lambda/2 \rfloor$, $\boldsymbol{w}_i = \frac{\ln(\mu+\frac{1}{2}) - \ln i}{\sum_{j=1}^{\mu}(\ln(\mu+\frac{1}{2}) - \ln j)}$ for $i = 1 \ldots \mu$,
   $\mu_w = \frac{1}{\sum_{i=1}^{\mu} w_i^2}$, $c_\sigma = \frac{\mu_w+2}{n+\mu_w+3}$, $d_\sigma = 1 + c_\sigma + 2\max(0, \sqrt{\frac{\mu_w-1}{n+1}} - 1)$, $c_c = \frac{4}{n+4}$, $c_1 =$
   $\frac{2\min(1,\lambda/6)}{(n+1.3)^2+\mu_w}$, $c_\mu = \frac{2\,(\mu_w-2+1/\mu_w)}{(n+2)^2+\mu_w}$
2: initialize $\boldsymbol{m}^{t=0} \in \mathbb{R}^n, \sigma^{t=0} > 0, \boldsymbol{p}_\sigma^{t=0} = \mathbf{0}, \boldsymbol{p}_c^{t=0} = \mathbf{0}, \mathbf{C}^{t=0} = \mathbf{I}, t \leftarrow 0$
3: repeat
4:     for $k \leftarrow 1, \ldots, \lambda$ do
5:         $\boldsymbol{x}_k \leftarrow \boldsymbol{m}^t + \sigma^t \mathcal{N}(\mathbf{0}, \mathbf{C}^t)$
6:         $\boldsymbol{f}_k \leftarrow f(\boldsymbol{x}_k)$
7:     $\boldsymbol{m}^{t+1} \leftarrow \sum_{i=1}^{\mu} \boldsymbol{w}_i \boldsymbol{x}_{i:\lambda}$ // the symbol $i:\lambda$ denotes $i$-th best individual on $f$
8:     $\boldsymbol{p}_\sigma^{t+1} \leftarrow (1-c_\sigma)\boldsymbol{p}_\sigma^t + \sqrt{c_\sigma(2-c_\sigma)}\sqrt{\mu_w}\mathbf{C}^{t^{-\frac{1}{2}}} \frac{\boldsymbol{m}^{t+1}-\boldsymbol{m}^t}{\sigma^t}$
9:     $h_\sigma \leftarrow \mathbb{1}_{\|p_\sigma^{t+1}\| < \sqrt{1-(1-c_\sigma)^{2(t+1)}}(1.4+\frac{2}{n+1})\,\mathbb{E}\|\mathcal{N}(\mathbf{0},\mathbf{I})\|}$
10:    $\boldsymbol{p}_c^{t+1} \leftarrow (1-c_c)\boldsymbol{p}_c^t + h_\sigma\sqrt{c_c(2-c_c)}\sqrt{\mu_w}\frac{\boldsymbol{m}^{t+1}-\boldsymbol{m}^t}{\sigma^t}$
11:    $\mathbf{C}_\mu \leftarrow \sum_{i=1}^{\mu} w_i \frac{\boldsymbol{x}_{i:\lambda}-\boldsymbol{m}^t}{\sigma^t} \times \frac{(\boldsymbol{x}_{i:\lambda}-\boldsymbol{m}^t)^T}{\sigma^t}$
12:    $\mathbf{C}^{t+1} \leftarrow (1-c_1-c_\mu)\mathbf{C}^t + c_1 \underbrace{\boldsymbol{p}_c^{t+1}\boldsymbol{p}_c^{t+1^T}}_{\text{rank-one update}} + c_\mu \underbrace{\mathbf{C}_\mu}_{\text{rank-}\mu\text{ update}}$
13:    $\sigma^{t+1} \leftarrow \sigma^t \exp(\frac{c_\sigma}{d_\sigma}(\frac{\|\boldsymbol{p}_\sigma^{t+1}\|}{\mathbb{E}\|\mathcal{N}(\mathbf{0},\mathbf{I})\|} - 1))$
14:    $t \leftarrow t+1$
15: until stopping criterion is met
```

$\mathbf{C}^t \in \mathbb{R}^{n \times n}$ and a mutation step-size $\sigma^t$ as follows:

$$\boldsymbol{x}_k^t \leftarrow \mathcal{N}\left(\boldsymbol{m}^t, {\sigma^t}^2\mathbf{C}^t\right) \leftarrow \boldsymbol{m}^t + \sigma^t\mathcal{N}\left(\mathbf{0}, \mathbf{C}^t\right) \tag{6}$$

These $\lambda$ solutions then should be evaluated with an objective function $f$ (line 6). The old mean of the mutation distribution is stored in $\boldsymbol{m}^t$ and a new mean $\boldsymbol{m}^{t+1}$ is computed as a *weighted sum* of the best $\mu$ parent individuals selected among $\lambda$ generated offspring individuals (line 7). The weights $\boldsymbol{w}$ are used to control the impact of the selected individuals, the weights are usually higher for better ranked individuals (line 1).

The procedure of the adaptation of the step-size $\sigma^t$ in CMA-ES is inherited from the Cumulative Step-Size Adaptation Evolution Strategy (CSA-ES) (Hansen and Ostermeier, 1996) and is controlled by evolution path $\boldsymbol{p}_\sigma^{t+1}$. Successful mutation steps $\frac{\boldsymbol{m}^{t+1}-\boldsymbol{m}^t}{\sigma^t}$ (line 8) are tracked in the space of sampling, i.e., in the isotropic coordinate system defined by principal components of the covariance matrix $\mathbf{C}^t$. To update the evolution path $\boldsymbol{p}_\sigma^{t+1}$ a decay/relaxation factor $c_\sigma$ is used to decrease the importance of the previously performed steps with time. The step-size update rule increases the step-size if the length of the evolution path $\boldsymbol{p}_\sigma^{t+1}$ is longer than the expected length of the evolution path under random selection $\mathbb{E}\,\|\mathcal{N}(\mathbf{0},\mathbf{I})\|$, and decreases otherwise (line 13). The expectation of $\|\mathcal{N}(\mathbf{0},\mathbf{I})\|$ is approximated by $\sqrt{n}(1 - \frac{1}{4n} + \frac{1}{21n^2})$. A damping parameter $d_\sigma$ controls the change of the step-size.

The covariance matrix update consists of two parts (line 12): *rank-one update* (Hansen and Ostermeier, 2001) and *rank-$\mu$ update* (Hansen et al., 2003). The rank-one update computes evolution path $\boldsymbol{p}_c^{t+1}$ of successful moves of the mean $\frac{\boldsymbol{m}^{t+1}-\boldsymbol{m}^t}{\sigma^t}$ of the mutation distribution in the given coordinate system (line 10), in a similar way as for the evolution path $\boldsymbol{p}_\sigma^{t+1}$ of the step-size. To stall the update of $\boldsymbol{p}_c^{t+1}$ when $\sigma$ increases rapidly, a $h_\sigma$ trigger is used (line 9).

The rank-$\mu$ update computes a covariance matrix $\mathbf{C}_\mu$ as a weighted sum of covariances of successful steps of $\mu$ best individuals (line 11). The update of $\mathbf{C}$ itself is a replace of the previously accumulated information by a new one with corresponding weights of importance (line 12): $c_1$ for covariance matrix $\boldsymbol{p}_c^{t+1}\boldsymbol{p}_c^{t+1^T}$ of rank-one update and $c_\mu$ for $\mathbf{C}_\mu$ of rank-$\mu$ update (Hansen et al., 2003) such that $c_1 + c_\mu \leq 1$. It was also proposed to take into account unsuccessful mutations in the *"active" rank-$\mu$ update* (Hansen and Ros, 2010; Jastrebski and Arnold, 2006).

In CMA-ES, the factorization of the covariance $\mathbf{C}$ into $\mathbf{A}\mathbf{A}^T = \mathbf{C}$ is needed to sample the multivariate normal distribution (line 5). The eigendecomposition with $O(n^3)$ complexity is used for the factorization. Already in the original CMA-ES it was proposed to perform the eigendecomposition every $n/10$ generations (not shown in **Algorithm** 1) to reduce the complexity per function evaluation to $O(n^2)$.

## 3.2 Large Scale Variants

The original $O(n^2)$ time and space complexity of CMA-ES precludes its applications to large scale optimization with $n \gg 1000$. To enable the algorithm for large scale optimization, a linear time and space version called sep-CMA-ES was proposed by Ros and Hansen (2008). The algorithm does not learn dependencies but the scaling of variables by restraining the covariance matrix update to the diagonal elements:

$$c_{jj}^{t+1} = (1 - c_{cov})c_{jj}^t + \frac{1}{\mu_{cov}}\left(\boldsymbol{p}_c^{t+1}\right)_j^2 + c_{ccov}\left(1 - \frac{1}{\mu_{ccov}}\right)\sum_{i=1}^{\mu} w_i c_{jj}^t \left({z_{i:\lambda}}^{t+1}\right)_j^2, j = 1, \ldots, n \quad (7)$$

where, for $j = 1, \ldots, n$ the $c_{jj}$ are the diagonal elements of $\mathbf{C}^t$ and the $\left({z_{i:\lambda}}^{t+1}\right)_j = \left({x_{i:\lambda}}^{t+1}\right)_j / (\sigma^t \sqrt{c_{jj}^t})$.

This update reduces the computational complexity to $O(n)$ and allows to exploit problem separability. The algorithm demonstrated good performance on separable problems and even outperformed CMA-ES on non-separable Rosenbrock function for $n > 100$.

A Natural Evolution Strategy (NES) variant, the Rank-One NES (R1-NES) by Sun et al. (2011), adapts the search distribution according to the natural gradient with a particular low rank parametrization of the covariance matrix,

$$\mathbf{C} = \sigma^2(\mathbf{I} + \boldsymbol{u}\boldsymbol{u}^T), \quad (8)$$

where $u$ and $\sigma$ are the parameters to be adjusted. The adaptation of the predominant eigendirection $\boldsymbol{u}$ allows the algorithm to solve highly non-separable problems while maintaining only $O(n)$ time and $O(\mu n)$ space complexity. The use of the natural gradient in the derivation of the NES algorithm motivated a research which led to the formulation of the Information Geometric Optimization (IGO) framework by Ollivier et al. (2011).

The IGO framework was used to derive a similar to R1-NES algorithm called VD-CMA (Akimoto et al., 2014) with the sampling distribution parametrized by a

Gaussian model with the covariance matrix restricted as follows:

$$\mathbf{C} = D(\mathbf{I} + \boldsymbol{u}\boldsymbol{u}^T)D, \tag{9}$$

where $D$ is a diagonal matrix of dimension $n$ and $\boldsymbol{u}$ is a vector in $\mathbb{R}^n$. This model is able to represent a scaling for each variable by $D$ and a principal component, which is generally not parallel to an axis, by $Dv$ (Akimoto et al., 2014). It has $O(n)$ time and $O(\mu n)$ space complexity but i) typically demonstrates a better performance than sep-CMA-ES and R1-NES and ii) can solve a larger class of functions (Akimoto et al., 2014).

A version of CMA-ES with a limited memory storage also called limited memory CMA-ES (L-CMA-ES) was proposed by Knight and Lunacek (2007). The L-CMA-ES uses the $m$ eigenvectors and eigenvalues spanning the $m$-dimensional dominant subspace of the $n \times n$-dimensional covariance matrix $\mathbf{C}$ . The authors adapted a singular value decomposition updating algorithm developed by Brand (2006) that allowed to avoid the explicit computation and storage of the covariance matrix. For $m < n$ the performance in terms of the number of function evaluations gradually decreases while enabling the search in $\mathbb{R}^n$ for $n > 10,000$. However, the computational complexity of $O(m^2 n)$ practically (for $m$ in order of $\sqrt{n}$ as suggested by Knight and Lunacek (2007)) leads to the same limitations of $O(n^2)$ time complexity as in the original CMA-ES.

The $(\mu/\mu_w, \lambda)$-Cholesky-CMA-ES proposed by Suttorp et al. (2009) is of special interest in this paper because the LM-CMA is based on this algorithm. The Cholesky-CMA represents a version of CMA-ES with rank-one update where instead of performing the factorization of the covariance matrix $\mathbf{C}^t$ into $\mathbf{A}^t{\mathbf{A}^t}^T = \mathbf{C}^t$, the Cholesky factor $\mathbf{A}^t$ and its inverse ${\mathbf{A}^t}^{-1}$ are iteratively updated. From **Theorem 1** (Suttorp et al., 2009) it follows that if $\mathbf{C}^t$ is updated as

$$\mathbf{C}^{t+1} = \alpha \mathbf{C}^t + \beta \boldsymbol{v}^t {\boldsymbol{v}^t}^T, \tag{10}$$

where $\boldsymbol{v} \in \mathbb{R}^n$ is given in the decomposition form $\boldsymbol{v}^t = \mathbf{A}^t \boldsymbol{z}^t$, and $\alpha, \beta \in \mathbb{R}^+$, then for $\boldsymbol{z} \neq \boldsymbol{0}$ a Cholesky factor of the matrix $\mathbf{C}^{t+1}$ can be computed by

$$\mathbf{A}^{t+1} = \sqrt{\alpha}\mathbf{A}^t + \frac{\sqrt{\alpha}}{\|\boldsymbol{z}^t\|^2}\left(\sqrt{1 + \frac{\beta}{\alpha}\|\boldsymbol{z}^t\|^2} - 1\right)[\mathbf{A}^t \boldsymbol{z}^t]{\boldsymbol{z}^t}^T, \tag{11}$$

for $\boldsymbol{z}_t = \boldsymbol{0}$ we have $\mathbf{A}^{t+1} = \sqrt{\alpha}\mathbf{A}^t$. From the **Theorem 2** (Suttorp et al., 2009) it follows that if ${\mathbf{A}^{-1}}^t$ is the inverse of $\mathbf{A}^t$, then the inverse of $\mathbf{A}^{t+1}$ can be computed by

$${\mathbf{A}^{-1}}^{t+1} = \frac{1}{\sqrt{\alpha}}{\mathbf{A}^{-1}}^t - \frac{1}{\sqrt{\alpha}\|\boldsymbol{z}^t\|^2}\left(1 - \frac{1}{\sqrt{1 + \frac{\beta}{\alpha}\|\boldsymbol{z}^t\|^2}}\right)\boldsymbol{z}^t[{\boldsymbol{z}^t}^T {\mathbf{A}^{-1}}^t], \tag{12}$$

for $\boldsymbol{z}^t \neq \boldsymbol{0}$ and by ${\mathbf{A}^{-1}}^{t+1} = \frac{1}{\sqrt{\alpha}}{\mathbf{A}^{-1}}^t$ for $\boldsymbol{z}^t = \boldsymbol{0}$.

The $(\mu/\mu_w, \lambda)$-Cholesky-CMA-ES is outlined in **Algorithm** 2. As well as in the original CMA-ES, Cholesky-CMA-ES proceeds by sampling $\lambda$ candidate solutions (lines 4 - 7) and taking into account the most successful $\mu$ out of $\lambda$ solutions in the evolution path adaptation (lines 10 and 11). However, the eigendecomposition procedure is not required anymore because the Cholesky factor and its inverse are updated incrementally (line 13 and 14). This simplifies a lot the implementation of the algorithm and keeps its time complexity as $O(n^2)$. A postponed update of the Cholesky factors every $O(n)$ iterations would not reduce the asymptotic complexity further (as it does in the

**Algorithm 2** The $(\mu/\mu_w, \lambda)$-Cholesky-CMA-ES

```
1: given $n \in \mathbb{N}_+$, $\lambda = 4 + \lfloor 3 \ln n \rfloor$, $\mu = \lfloor \lambda/2 \rfloor$, $w_i = \frac{\ln(\mu+1)-\ln(i)}{\mu \ln(\mu+1) - \sum_{j=1}^{\mu} \ln(j)}$; $i = 1 \ldots \mu$,
   $\mu_w = \frac{1}{\sum_{i=1}^{\mu} w_i^2}$, $c_\sigma = \frac{\sqrt{\mu_w}}{\sqrt{n}+\sqrt{\mu_w}}$, $d_\sigma = 1 + c_\sigma + 2\max(0, \sqrt{\frac{\mu_w - 1}{n+1}} - 1)$, $c_c = \frac{4}{n+4}$,
   $c_1 = \frac{2}{(n+\sqrt{2})^2}$
2: initialize $\boldsymbol{m}^{t=0} \in \mathbb{R}^n, \sigma^{t=0} > 0, \boldsymbol{p}_\sigma^{t=-1} = \mathbf{0}, \boldsymbol{p}_c^{t=-1} = \mathbf{0}, \mathbf{A}^{t=0} = \mathbf{I}, \mathbf{A}_{inv}^{t=0} = \mathbf{I}, t \leftarrow 0$
3: repeat
4:     for $k \leftarrow 1, \ldots, \lambda$ do
5:         $\boldsymbol{z}_k \leftarrow \mathcal{N}(\mathbf{0}, \mathbf{I})$
6:         $\boldsymbol{x}_k \leftarrow \boldsymbol{m}^t + \sigma^t \mathbf{A} \boldsymbol{z}_k$
7:         $\boldsymbol{f}_k \leftarrow f(\boldsymbol{x}_k)$
8:     $\boldsymbol{m}^{t+1} \leftarrow \sum_{i=1}^{\mu} \boldsymbol{w}_i \boldsymbol{x}_{i:\lambda}$
9:     $\boldsymbol{z}_w \leftarrow \sum_{i=1}^{\mu} \boldsymbol{w}_i \boldsymbol{z}_{i:\lambda}$
10:    $\boldsymbol{p}_\sigma^t \leftarrow (1 - c_\sigma)\boldsymbol{p}_\sigma^{t-1} + \sqrt{c_\sigma(2 - c_\sigma)}\sqrt{\mu_w}\boldsymbol{z}_w$
11:    $\boldsymbol{p}_c^t \leftarrow (1 - c_c)\boldsymbol{p}_c^{t-1} + \sqrt{c_c(2 - c_c)}\sqrt{\mu_w}\mathbf{A}^t \boldsymbol{z}_w$
12:    $\boldsymbol{v}^t \leftarrow \mathbf{A}_{inv}^t \boldsymbol{p}_c^t$
13:    $\mathbf{A}^{t+1} \leftarrow \sqrt{1 - c_1}\mathbf{A}^t + \frac{\sqrt{1-c_1}}{\|\boldsymbol{v}^t\|^2}\left(\sqrt{1 + \frac{c_1}{1-c_1}\|\boldsymbol{v}^t\|^2} - 1\right)\boldsymbol{p}_c^t {\boldsymbol{v}^t}^T$
14:    $\mathbf{A}_{inv}^{t+1} \leftarrow \frac{1}{\sqrt{1-c_1}}\mathbf{A}_{inv}^t - \frac{1}{\sqrt{1-c_1}\|\boldsymbol{v}^t\|^2}\left(1 - \frac{1}{\sqrt{1+\frac{c_1}{1-c_1}\|\boldsymbol{v}^t\|^2}}\right)\boldsymbol{v}^t[{\boldsymbol{v}^t}^T \mathbf{A}_{inv}^t]$,
15:    $\sigma^{t+1} \leftarrow \sigma^t \exp(\frac{c_\sigma}{d_\sigma}(\frac{\|\boldsymbol{p}_\sigma^t\|}{\mathbb{E}\|\mathcal{N}(\mathbf{0},\mathbf{I})\|} - 1))$
16:    $t \leftarrow t + 1$
17: until stopping criterion is met
```

original CMA-ES) because the quadratic complexity will remain due to matrix-vector multiplications needed to sample new individuals.

The non-elitist Cholesky-CMA is a good alternative to the original CMA-ES and demonstrates a comparable performance (Suttorp et al., 2009). While it has the same computational and memory complexity, the lack of rank-$\mu$ update may deteriorate its performance on problems where it is essential.

## 4 The LM-CMA

The LM-CMA is inspired by the L-BFGS algorithm but instead of storing $m$ gradients for performing inverse Hessian requiring operations it stores $m$ direction vectors to reproduce the Cholesky factor $\mathbf{A}$ and generate candidate solutions with a limited time and space cost $O(mn)$ (see Section 4.1). These $m$ vectors are estimates of descent directions provided by evolution path vectors and should be stored with a particular temporal distance to enrich $\mathbf{A}$ (see Section 4.2). An important novelty introduced w.r.t. the original LM-CMA proposed by Loshchilov (2014) is a procedure for sampling from a family of search representations defined by Cholesky factors reconstructed from $m^* \leq m$ vectors (see Section 4.3) and according to the Rademacher distribution (see Section 4.4). These novelties allow to simultaneously reduce the internal time complexity of the algorithm and improve its performance in terms of the number of function evaluations. Before describing the algorithm itself, we gradually introduce all the necessary components.

**Algorithm 3** Az(): Cholesky factor - vector update

```
1: given $\boldsymbol{z} \in \mathbb{R}^n, m \in \mathbb{Z}_+, \boldsymbol{j} \in \mathbb{Z}_+^m, \boldsymbol{i} \in \mathbb{Z}_+^{|\boldsymbol{i}|}, \boldsymbol{P} \in \mathbb{R}^{m \times n}, \boldsymbol{V} \in \mathbb{R}^{m \times n}, \boldsymbol{b} \in \mathbb{R}^m, a \in [0,1]$
2: initialize $\boldsymbol{x} \leftarrow \boldsymbol{z}$
3: for $t \leftarrow 1, \ldots, |\boldsymbol{i}|$ do
4:     $k \leftarrow \boldsymbol{b}^{\boldsymbol{j}_{i_t}} \boldsymbol{V}^{(j_{i_t},:)} \cdot \boldsymbol{z}$
5:     $\boldsymbol{x} \leftarrow a\boldsymbol{x} + k\boldsymbol{P}^{(j_{i_t},:)}$
6: return $\boldsymbol{x}$
```

**Algorithm 4** Ainvz(): inverse Cholesky factor - vector update

```
1: given $\boldsymbol{z} \in \mathbb{R}^n, m \in \mathbb{Z}_+, \boldsymbol{j} \in \mathbb{Z}_+^m, \boldsymbol{i} \in \mathbb{Z}_+^{|\boldsymbol{i}|}, \boldsymbol{d} \in \mathbb{R}^m, c \in [0,1]$
2: initialize $\boldsymbol{x} \leftarrow \boldsymbol{z}$
3: for $t \leftarrow 1, \ldots, |\boldsymbol{i}|$ do
4:     $k \leftarrow \boldsymbol{d}^{\boldsymbol{j}_{i_t}} \boldsymbol{V}^{(j_{i_t},:)} \cdot \boldsymbol{x}$
5:     $\boldsymbol{x} \leftarrow c\boldsymbol{x} - k\boldsymbol{V}^{(j_{i_t},:)}$
6: return $\boldsymbol{x}$
```

### 4.1 Reconstruction of Cholesky Factors

By setting $a = \sqrt{1-c_1}$, $b^t = \frac{\sqrt{1-c_1}}{\|\boldsymbol{v}^t\|^2}\left(\sqrt{1 + \frac{c_1}{1-c_1}\|\boldsymbol{v}^t\|^2} - 1\right)$ and considering the *evolution path* $\boldsymbol{p}_c^t$ (a change of optimum estimate $\boldsymbol{m}$ smoothed over iterations, see line 12 of **Algorithm** 7) together with $\boldsymbol{v}^t = \mathbf{A}^{-1^t}\boldsymbol{p}_c^t$, one can rewrite Equation (11) as

$$\mathbf{A}^{t+1} = a\mathbf{A}^t + b^t \boldsymbol{p}_c^t \boldsymbol{v}^{t^T}, \tag{13}$$

The product of a random vector $\boldsymbol{z}$ and the Cholesky factor $\mathbf{A}^t$ thus can be directly computed. At iteration $t = 0$, $\mathbf{A}^0 = \mathbf{I}$ and $\mathbf{A}^0\boldsymbol{z} = \boldsymbol{z}$, the new updated Cholesky factor $\mathbf{A}^1 = a\mathbf{I} + b^0\boldsymbol{p}_c^0\boldsymbol{v}^{0^T}$. At iteration $t = 1$, $\mathbf{A}^1\boldsymbol{z} = (a\mathbf{I} + b^0\boldsymbol{p}_c^0\boldsymbol{v}^{0^T})\boldsymbol{z} = a\boldsymbol{z} + b^0\boldsymbol{p}_c^0(\boldsymbol{v}^{0^T}\boldsymbol{z})$ and $\mathbf{A}^2 = a(a\mathbf{I} + b^0\boldsymbol{p}_c^0\boldsymbol{v}^{0^T}) + b^1\boldsymbol{p}_c^1\boldsymbol{v}^{1^T}$. Thus, a very simple iterative procedure which scales as $O(mn)$ can be used to sample candidate solutions in $\mathbb{R}^n$ according to the Cholesky factor $\mathbf{A}^t$ reconstructed from $m$ pairs of vectors $\boldsymbol{p}_c^t$ and $\boldsymbol{v}^t$.

The pseudo-code of the procedure to reconstruct $\boldsymbol{x} = \mathbf{A}^t\boldsymbol{z}$ from $m$ direction vectors is given in **Algorithm** 3. At each iteration of reconstruction of $\boldsymbol{x} = \mathbf{A}^t\boldsymbol{z}$ (lines 3 - 4), $\boldsymbol{x}$ is updated as a sum of $a$-weighted version of itself and $b^t$-weighted evolution path $\boldsymbol{p}_c^t$ (accessed from a matrix $\boldsymbol{P} \in R^{m \times n}$ as $\boldsymbol{P}^{(i_t,:)}$ ) scaled by the dot product of $\boldsymbol{v}^t$ and $\boldsymbol{x}$. As can be seen, **Algorithms** 3 and 4 use $\boldsymbol{j}_{i_t}$ indexation instead of $t$. This is a convenient way to have references to matrices $\boldsymbol{P}$ and $\boldsymbol{V}$ which store $\boldsymbol{p}_c$ and $\boldsymbol{v}$ vectors, respectively. In the next subsections, we will show how to efficiently manipulate these vectors.

A very similar approach can be used to reconstruct $\boldsymbol{x} = \mathbf{A}^{t^{-1}}\boldsymbol{z}$, for the sake of reproducibility the pseudo-code is given in **Algorithm** 4 for $c = 1/\sqrt{1-c_1}$ and $d^t = \frac{1}{\sqrt{1-c_1}\|\boldsymbol{v}^t\|^2} \times \left(1 - \frac{1}{\sqrt{1+\frac{c_1}{1-c_1}\|\boldsymbol{v}^t\|^2}}\right)$. The computational complexity of both procedures scales as $O(mn)$.

It is important to note that when a vector $\boldsymbol{p}^\ell$ from a set of $m$ vectors stored in $\boldsymbol{P}$ is replaced by a new vector $\boldsymbol{p}^{t+1}$ (see line 15 in **Algorithm** 7), *all* inverse vectors $\boldsymbol{v}^t$ for $t = \ell, \ldots, m$ should be iteratively recomputed (Krause, 2014). This procedure corresponds to line 16 in **Algorithm** 7.

**Algorithm 5** UpdateSet(): direction vectors update

```
1: given $m \in \mathbb{R}^+, \boldsymbol{j} \in \mathbb{Z}_+^m, \boldsymbol{l} \in \mathbb{Z}_+^m, t \in \mathbb{Z}_+, \boldsymbol{N} \in \mathbb{Z}_+^m, \boldsymbol{P} \in \mathbb{R}^{m \times n}, \boldsymbol{p}_c \in \mathbb{R}^n, T \in \mathbb{Z}_+$
2: $t \leftarrow \lfloor t/T \rfloor$
3: if $t \leq m$ then
4:     $\boldsymbol{j}_t \leftarrow t$
5: else
6:     $i_{min} \leftarrow 1 + argmin_i \left( \boldsymbol{l}_{\boldsymbol{j}_{i+1}} - \boldsymbol{l}_{\boldsymbol{j}_i} - \boldsymbol{N}_i \right), |1 \leq i \leq (m-1)$
7:     if $\boldsymbol{l}_{\boldsymbol{j}_{i_{min}}} - \boldsymbol{l}_{\boldsymbol{j}_{i_{min}-1}} - \boldsymbol{N}_i \geq 0$ then
8:         $i_{min} \leftarrow 1$
9:     $\boldsymbol{j}_{tmp} \leftarrow \boldsymbol{j}_{i_{min}}$
10:    for $i \leftarrow i_{min}, \ldots, m-1$ do
11:        $\boldsymbol{j}_i \leftarrow \boldsymbol{j}_{i+1}$
12:    $\boldsymbol{j}_m \leftarrow \boldsymbol{j}_{tmp}$
13: $j_{cur} \leftarrow \boldsymbol{j}_{\mathtt{min}(t+1,m)}$
14: $\boldsymbol{l}_{j_{cur}} \leftarrow tT$
15: $\boldsymbol{P}^{(j_{cur},:)} \leftarrow \boldsymbol{p}_c$
16: return: $\boldsymbol{j}_{cur}, \boldsymbol{j}, \boldsymbol{l}$
```

### 4.2 Direction Vectors Selection and Storage

The choice to store only $m \ll n$ direction vectors $\boldsymbol{p}_c$ to obtain a comparable amount of useful information as stored in the covariance matrix of the original CMA-ES requires a careful procedure of selection and storage. A simple yet powerful procedure proposed by Loshchilov (2014) is to preserve a certain temporal distance in terms of number of iterations between the stored direction vectors. The procedure tends to store a more unique information in contrast to the case if the latest $m$ evolution path vectors would be stored. The latter case is different from the storage of $m$ gradients as in L-BFGS since the evolution path is gradually updated at each iteration with a relatively small learning rate $c_c$ and from $\mu \ll n$ selected vectors.

The selection procedure is outlined in **Algorithm** 5 which outputs an array of pointers $\boldsymbol{j}$ such that $\boldsymbol{j}_1$ points out to a row in matrices $\boldsymbol{P}$ and $\boldsymbol{V}$ with the oldest saved vectors $\boldsymbol{p}_c$ and $\boldsymbol{v}$ which will be taken into account during the reconstruction procedure. The higher the index $i$ of $\boldsymbol{j}_i$ the more recent the corresponding direction vector is. The index $j_{cur}$ points out to the vector which will be replaced by the newest one in the same iteration when the procedure is called. The rule to choose a vector to be replaced is the following. Find a pair of consecutively saved vectors ($\boldsymbol{P}^{(j_{i_{min}-1},:)}, \boldsymbol{P}^{(j_{i_{min}},:)}$) with the distance between them (in terms of indexes of iterations, stored in $\boldsymbol{l}$) closest to a target distance $N_i$ (line 6). If this distance is smaller than $N_i$ then the index $j_{i_{min}}$ will be swapped with last index of $\boldsymbol{j}$ (lines 9-12) and the corresponding vector $\boldsymbol{P}^{(j_{i_{min}},:)}$ will be replaced by the new vector $\boldsymbol{p}_c$ (line 15), otherwise the oldest vector among $m$ saved vectors will be removed (as a result of line 8). Thus, the procedure gradually replaces vectors in a way to keep them at the distance $N_i$ and with the overall time horizon for all vectors of at most $\sum_i^{m-1} N_i$ iterations. The procedure can be called periodically every $T \in \mathbb{Z}_+$ iterations of the algorithm. The values of $N_i$ are to be defined, e.g., as a function of problem dimension $n$ and direction vector index $i$. Here, however, we set $N_i$ to $n$ for all $i$, i.e., the target distance equals to the problem dimension.

**Algorithm 6** SelectSubset(): direction vectors selection

```
1: given $m \in \mathbb{Z}_+, m_\sigma = 4, k \in \mathbb{Z}_+$
2: if $k = 1$ then
3:     $m_\sigma \leftarrow 10 m_\sigma$
4: $m^* \leftarrow \min(\lfloor m_\sigma |\mathcal{N}(0,1)| \rfloor, m)$
5: $\boldsymbol{i} \leftarrow (m + 1 - m^*), \ldots, m$
6: return $\boldsymbol{i}$
```

## 4.3 Sampling from a Family of Search Representations

At iteration $t$, a new $k$-th solution can be generated as

$$\boldsymbol{x}_k \leftarrow \boldsymbol{m}^t + \sigma^t Az(\boldsymbol{z}_k, \boldsymbol{i}), \tag{14}$$

where $\boldsymbol{z}_k \in \mathbb{R}^n$ is a vector drawn from some distribution and transformed by a Cholesky factor by calling $Az(\boldsymbol{z}_k, \boldsymbol{i})$. The $Az()$ procedure (see **Algorithm** 3) has an input $\boldsymbol{i}$ which defines indexes of direction vectors used to reconstruct the Cholesky factor. It is important to note that $\boldsymbol{P}^{(1,:)}$ refers to the first vector physically stored in matrix $\boldsymbol{P}$, $\boldsymbol{P}^{(j_1,:)}$ refers to the oldest vector, $\boldsymbol{P}^{(j_{i_t},:)}$ refers to the $\boldsymbol{i}_t$-th oldest vector according to an array $\boldsymbol{i}$ with indexes of vectors of interest. Thus, by setting $\boldsymbol{i} = 1, \ldots, m$ all $m$ vectors will be used in the reconstruction. Importantly, omission of some vector in $\boldsymbol{i}$ can be viewed as setting of the corresponding learning rate in Equation (13) to 0.

By varying $\boldsymbol{i}$, one can control the reconstruction of the Cholesky factor used for sampling and in this way explore a family of possible transformations of the coordinate system. The maximum number of vectors defined by $m$ can be associated with the number of degrees of freedom of this exploration.

While in the original LM-CMA (Loshchilov, 2014) the value of $m$ is set to $4 + \lfloor 3 \ln n \rfloor$ to allow the algorithm scale up to millions of variables, we found that greater values of $m$, e.g., $\sqrt{n}$ often lead to better performance (see Section 5.7 for a detailed analysis). Thus, when memory allows, a gain in performance can be achieved. However, due to an internal cost $O(mn)$ of $Az()$, the time cost then would scale as $O(n^{3/2})$ which is undesirable for $n \gg 1000$. This is where the use of $m^*$ out of $m$ vectors can drastically reduce the time complexity. We propose to sample $m^*$ from a truncated half-normal distribution $\left|\mathcal{N}\left(0, m_\sigma^2\right)\right|$ (see line 4 of **Algorithm** 6) and set $\boldsymbol{i}$ to the latest $m^*$ vectors (line 5). For a constant $m_\sigma = 4$, the time complexity of $Az()$ scales as $O(n)$. New value of $m^*$ is generated for each new individual. Additionally, to exploit the oldest information, we force $m^*$ to be generated with $10m_\sigma$ for one out of $\lambda$ individuals. While for $m^* = 0$ the new solution $\boldsymbol{x}_k$ appears to be sampled from an isotropic normal distribution, the computation of $\boldsymbol{v}$ inverses is performed using all $m$ vectors.

## 4.4 Sampling from the Rademacher Distribution

Evolution Strategies are mainly associated with the multivariate normal distribution used to sample candidate solutions. However, alternative distributions such as the Cauchy distribution can be used (Yao and Liu, 1997; Hansen, 2008). Moreover, the Adaptive Encoding procedure proposed by Hansen (2008) can be coupled with any sampling distribution as in Loshchilov et al. (2011), where it was shown that completely deterministic adaptive coordinate descent on principal components obtained with the Adaptive Encoding procedure allows to obtain the performance comparable to the one of CMA-ES.

In this paper, inspired by Loshchilov et al. (2011), we replace the original multivariate normal distribution used in LM-CMA by the Rademacher distribution, where a random variable has 50% chance to be either -1 or +1 (also can be viewed as a Bernoulli distribution). Thus, a pre-image vector of candidate solution $\boldsymbol{z}$ contains $n$ values which are either -1 or +1. Our intention to use this distribution is three-fold: i) to reduce the computation complexity by a constant but rather significant factor, ii) to demonstrate that the Rademacher distribution can potentially be an alternative to the Gaussian distribution at least in large scale settings, iii) to demonstrate that our new step-size adaptation rule (see next section), which does not make assumptions about the sampling distribution, can work well when used with non-Gaussian distributions. As a support for this substitution, we recall that for a $n$-dimensional unit-variance spherical Gaussian, for any positive real number $\beta \leq \sqrt{n}$, all but at most $3\exp^{-c\beta^2}$ of the mass lies within the annulus $\sqrt{n-1} - \beta \leq r \leq \sqrt{n-1} + \beta$, where, $c$ is a fixed positive constant (Hopcroft and Kannan, 2015). Thus, when $n$ is large, the mass is concentrated in a thin annulus of width $O(1)$ at radius $\sqrt{n}$. Interestingly, the sampling from the Rademacher distribution reproduces this effect of large-dimensional Gaussian sampling since the distance from the center of a $n$-dimensional hypercube to its corners is $\sqrt{n}$.

### 4.5 Population Success Rule

The step-size used to define the scale of deviation of a sampled candidate solution from the mean of the mutation distribution can be adapted by the Population Success Rule (PSR) proposed for LM-CMA by Loshchilov (2014). This procedure does not assume that candidate solutions should come from the multivariate normal distribution as it is often assumed in Evolution Strategies including CMA-ES. Therefore, PSR procedure is well suited for the Rademacher distribution.

The PSR is inspired by the *median success rule* (Ait Elhara et al., 2013). To estimate the success of the current population we combine fitness function values from the previous and current population into a mixed set

$$\boldsymbol{f}_{mix} \leftarrow \boldsymbol{f}^{t-1} \cup \boldsymbol{f}^{t} \tag{15}$$

Then, all individuals in the mixed set are ranked to define two sets $\boldsymbol{r}_{t-1}$ and $\boldsymbol{r}_t$ (the lower the rank the better the individual) containing ranks of individuals of the previous and current populations ranked in the mixed set. A normalized success measurement is computed as

$$z_{PSR} \leftarrow \frac{\sum_{i=1}^{\lambda} \boldsymbol{r}^{t-1}(i) - \boldsymbol{r}^{t}(i)}{\lambda^2} - z^*, \tag{16}$$

where $z^*$ is a target success ratio and $\lambda^2$ accounts for the normalization of the sum term and for different possible population size $\lambda$. Then, for $s \leftarrow (1 - c_\sigma)s + c_\sigma z_{PSR}$, the step-size is adapted as

$$\sigma^{t+1} \leftarrow \sigma^t \exp\left(s / d_\sigma\right), \tag{17}$$

where $d_\sigma$ is a damping factor which we set here to 1.

### 4.6 The Algorithm

The improved LM-CMA is given in **Algorithm** 7. At each iteration $t$, $\lambda$ candidate solutions are generated by mutation defined as a product of a vector $\boldsymbol{z}_k$ sampled from the Rademacher distribution and a Cholesky factor $\mathbf{A}^t$ reconstructed from $m^*$ out of

**Algorithm 7** The $(\mu/\mu_w, \lambda)$-LM-CMA

```
1: given $n \in \mathbb{N}_+$, $\lambda = 4 + \lfloor 3 \ln n \rfloor$, $\mu = \lfloor \lambda/2 \rfloor$, $w_i = \frac{\ln(\mu+1)-\ln(i)}{\mu \ln(\mu+1) - \sum_{j=1}^{\mu} \ln(j)}$; $i = 1 \dots \mu$, $\mu_w = \frac{1}{\sum_{i=1}^{\mu} w_i^2}$, $c_\sigma = 0.3$, $z^* = 0.25$, $m = 4 + \lfloor 3 \ln n \rfloor$, $N_{steps} = n$, $c_c = \frac{0.5}{\sqrt{n}}$, $c_1 = \frac{1}{10 \ln(n+1)}$, $d_\sigma = 1, T = \lfloor log(n) \rfloor$
2: initialize $\boldsymbol{m}^{t=0} \in \mathbb{R}^n, \sigma^{t=0} > 0, \boldsymbol{p}_c^{t=0} = \mathbf{0}, s \leftarrow 0, t \leftarrow 0$
3: repeat
4:   for $k \leftarrow 1, \dots, \lambda$ do
5:     if $k \pmod 2 = 1$ then
6:       $\boldsymbol{z}_k \leftarrow Rademacher()$
7:       $\boldsymbol{i} \leftarrow SelectSubset(k)$
8:       $\boldsymbol{x}_k \leftarrow \boldsymbol{m}^t + \sigma^t Az(\boldsymbol{z}_k, \boldsymbol{i})$
9:     else
10:      $\boldsymbol{x}_k \leftarrow \boldsymbol{m}^t - (\boldsymbol{x}_{k-1} - \boldsymbol{m}^t)$
11:    $\boldsymbol{f}_k^t \leftarrow f(\boldsymbol{x}_k)$
12:  $\boldsymbol{m}^{t+1} \leftarrow \sum_{i=1}^{\mu} \boldsymbol{w}_i \boldsymbol{x}_{i:\lambda}$
13:  $\boldsymbol{p}_c^{t+1} \leftarrow (1 - c_c)\boldsymbol{p}_c^t + \sqrt{c_c(2 - c_c)}\sqrt{\mu_w}(\boldsymbol{m}^{t+1} - \boldsymbol{m}^t)/\sigma^t$
14:  if $t \pmod T = 0$ then
15:    $UpdateSet(\boldsymbol{p}_c^{t+1})$
16:    $UpdateInverses()$
17:  $\boldsymbol{r}^t, \boldsymbol{r}^{t-1} \leftarrow$ Ranks of $\boldsymbol{f}^t$ and $\boldsymbol{f}^{t-1}$ in $\boldsymbol{f}^t \cup \boldsymbol{f}^{t-1}$
18:  $z_{PSR} \leftarrow \frac{\sum_{i=1}^{\lambda} \boldsymbol{r}^{t-1}(i) - \boldsymbol{r}^t(i)}{\lambda^2} - z^*$
19:  $s \leftarrow (1 - c_\sigma)s + c_\sigma z_{PSR}$
20:  $\sigma^{t+1} \leftarrow \sigma^t \exp(s/d_\sigma)$
21:  $t \leftarrow t + 1$
22: until stopping criterion is met
```

$m$ vectors (line 4-11) as described in Sections 4.1-4.4. We introduce the mirrored sampling (Brockhoff et al., 2010) to generate the actual $\boldsymbol{x}_k$ only every second time and thus decrease the computation cost per function evaluation by a factor of two by evaluating $\boldsymbol{m}^t + \sigma^t Az(\boldsymbol{z}_k)$ and then its mirrored version $\boldsymbol{m}^t - (\boldsymbol{x}_{k-1} - \boldsymbol{m}^t)$. The latter approach sometimes also improves the convergence rate.

The best $\mu$ out of $\lambda$ solutions are selected to compute the new mean $\boldsymbol{m}^{t+1}$ of the mutation distribution in line 12. The new evolution path $\boldsymbol{p}_c^{t+1}$ is updated (line 13) from the change of the mean vector $\sqrt{\mu_w}(\boldsymbol{m}^{t+1} - \boldsymbol{m}^t)/\sigma^t$ and represents an estimate of descent direction. One vector among $m$ vectors is selected and replaced by the new $\boldsymbol{p}_c^{t+1}$ in *UpdateSet()* procedure described in Section 4.2. All inverses $\boldsymbol{v}$ of evolution path vectors which are at least as recent as the direction vector to be replaced should be recomputed in the *UpdateInverses()* procedure as described in Section 4.1. The step-size is updated according to the PSR rule described in Section 4.5.

## 5 Experimental Validation

The performance of the LM-CMA is investigated comparatively to the L-BFGS (Liu and Nocedal, 1989), the active CMA-ES by Hansen and Ros (2010) and the VD-CMA by Akimoto et al. (2014). The sep-CMA-ES is removed from the comparison due to its similar but inferior performance w.r.t. the VD-CMA observed both in our study and by Akimoto et al. (2014).

Table 1: Test functions, initialization intervals and initial standard deviation (when applied). $\boldsymbol{R}$ is an orthogonal $n \times n$ matrix with each column vector $\boldsymbol{q}_i$ being a uniformly distributed unit vector implementing an angle-preserving transformation (Ros and Hansen, 2008)

| Name | Function | Target $f(x)$ | Init | $\sigma^0$ |
|---|---|---|---|---|
| Sphere | $f_{Sphere}(\boldsymbol{x})= \sum_{i=1}^{n} \boldsymbol{x}_i^2$ | $10^{-10}$ | $[-5,5]^n$ | 3 |
| Ellipsoid | $f_{Elli}(\boldsymbol{x})= \sum_{i=1}^{n} 10^{6\frac{i-1}{n-1}} \boldsymbol{x}_i^2$ | $10^{-10}$ | $[-5,5]^n$ | 3 |
| Rosenbrock | $f_{Rosen}(\boldsymbol{x})= \sum_{i=1}^{n-1} \left(100.(\boldsymbol{x}_i^2 - x_{i+1})^2 + (x_i - 1)^2\right)$ | $10^{-10}$ | $[-5,5]^n$ | 3 |
| Discus | $f_{Discus}(\boldsymbol{x})= 10^6 \boldsymbol{x}_1^2 + \sum_{i=2}^{n} \boldsymbol{x}_i^2$ | $10^{-10}$ | $[-5,5]^n$ | 3 |
| Cigar | $f_{Cigar}(\boldsymbol{x})= \boldsymbol{x}_1^2 + 10^6 \sum_{i=2}^{n} \boldsymbol{x}_i^2$ | $10^{-10}$ | $[-5,5]^n$ | 3 |
| Different Powers | $f_{DiffPow}(\boldsymbol{x})= \sum_{i=1}^{n} \lvert\boldsymbol{x}_i\rvert^{2+4(i-1)/(n-1)}$ | $10^{-10}$ | $[-5,5]^n$ | 3 |
| Rotated Ellipsoid | $f_{RotElli}(\boldsymbol{x})=f_{Elli}(\boldsymbol{Rx})$ | $10^{-10}$ | $[-5,5]^n$ | 3 |
| Rotated Rosenbrock | $f_{RotRosen}(\boldsymbol{x})=f_{Rosen}(\boldsymbol{Rx})$ | $10^{-10}$ | $[-5,5]^n$ | 3 |
| Rotated Discus | $f_{RotDiscus}(\boldsymbol{x})=f_{Discus}(\boldsymbol{Rx})$ | $10^{-10}$ | $[-5,5]^n$ | 3 |
| Rotated Cigar | $f_{RotCigar}(\boldsymbol{x})=f_{Cigar}(\boldsymbol{Rx})$ | $10^{-10}$ | $[-5,5]^n$ | 3 |
| Rotated Different Powers | $f_{RotDiffPow}(\boldsymbol{x})=f_{DiffPow}(\boldsymbol{Rx})$ | $10^{-10}$ | $[-5,5]^n$ | 3 |

We use the L-BFGS implemented in MinFunc library by Schmidt (2005) in its default parameter settings [1], the active CMA-ES (aCMA) without restarts in its default parametrization of CMA-ES MATLAB code version 3.61 [2]. For faster performance in terms of CPU time, the VD-CMA was (exactly) reimplemented in C language from the MATLAB code provided by the authors. For the sake of reproducibility, the source code of all algorithms is available online [3]. The default parameters of LM-CMA are given in **Algorithm** 7.

We use a set of benchmark problems (see Table 1) commonly used in Evolutionary Computation, more specifically in the BBOB framework (Finck et al., 2010). Indeed, many problems are missing including the ones where tested methods and LM-CMA fail to timely demonstrate reasonable performance in large scale settings. We focus on algorithm performance w.r.t. both the number of function evaluations used to reach target values of $f$, CPU time spent per function evaluation and the number of memory slots required to run algorithms. Any subset of these metrics can dominate search cost in large scale settings, while in low scale settings memory is typically of a lesser importance.

In this section, we first investigate the scalability of the proposed algorithm w.r.t. the existing alternatives. While both the computational time and space complexities scale moderately with problem dimension, the algorithm is capable to preserve certain invariance properties of the original CMA-ES. Moreover, we obtain unexpectedly good results on some well-known benchmark problems, e.g., linear scaling of the budget of function evaluations to solve Separable and Rotated Ellipsoid problems. We demonstrate that the performance of LM-CMA is comparable to the one of L-BFGS with exact estimation of gradient information. Importantly, we show that LM-CMA is competitive to L-BFGS in very large scale (for derivative-free optimization) settings with 100,000 variables. Finally, we investigate the sensitivity of LM-CMA to some key internal parameters such as the number of stored direction vectors $m$.

[1] `http://www.cs.ubc.ca/~schmidtm/Software/minFunc.html`
[2] `http://www.lri.fr/~hansen/cmaes.m`
[3] `http://sites.google.com/site/ecjlmcma/`

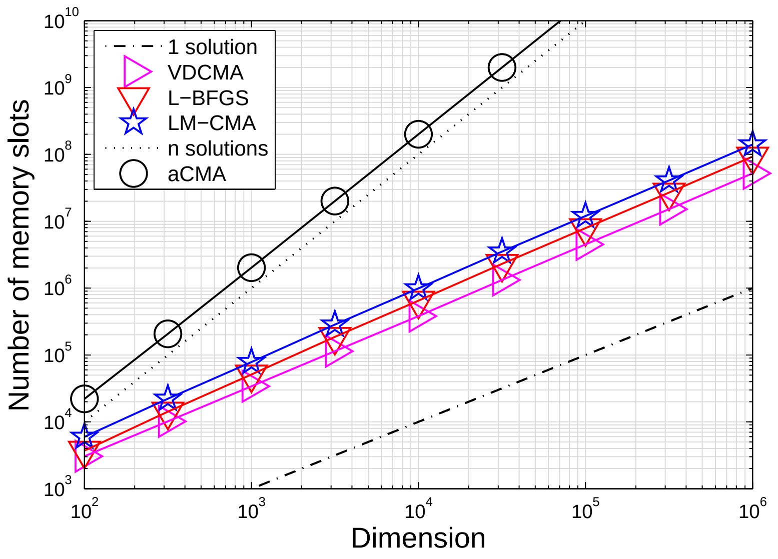

Figure 1: Number of memory slots of floating point variables required to run different optimization algorithms versus problem dimension $n$.

## 5.1 Space Complexity

The number of memory slots required to run optimization algorithms versus problem dimension $n$ referred to as space complexity can limit applicability of certain algorithms to large scale optimization problems. Here, we list the number of slots up to constant and asymptotically constant terms.

The presented algorithms store $\lambda$ generated solutions (LM-CMA, VD-CMA and aCMA with the default $\lambda = 4 + \lfloor 3 \ln n \rfloor$ and $\mu = \lambda/2$, L-BFGS with $\lambda = 1$) and some intermediate information (LM-CMA and L-BFGS with $m$ pairs of vectors, aCMA with at least two matrices of size $n \times n$) to perform the search. Our implementation of VD-CMA requires $(max(\mu + 15, \lambda) + 7)n$ slots compared to $(2.5\lambda + 21)n$ of the original MATLAB code. The LM-CMA requires $(2m + \lambda + 6)n + 5m$ slots, the L-BFGS requires $(2m + 3)n$ slots and aCMA requires $(2n + \lambda + 3)n$ slots.

Figure 1 shows that due to its quadratic space complexity aCMA requires about $2 \times 10^8$ slots (respectively, $8 \times 10^8$ slots) for 10,000-dimensional (respectively, 20,000-dimensional) problems which with 8 bytes per double-precision floating point number would correspond to about 1.6 GB (respectively, 6.4 GB) of computer memory. This simply precludes the use of CMA-ES and its variants with explicit storage of the full covariance matrix or Cholesky factors to large scale optimization problems with $n > 10,000$. LM-CMA stores $m$ pairs of vectors as well as the L-BFGS. For $m = 4 + \lfloor 3 \ln n \rfloor$ (as the default population size in CMA-ES), L-BFGS is 2 times and LM-CMA is 3 times more expensive in memory than VD-CMA, but they all basically can be run for millions of variables.

In this paper, we argue that additional memory can be used while it is allowed and is at no cost. Thus, while the default $m = 4 + \lfloor 3 \ln n \rfloor$, we suggest to use $m = \lfloor 2\sqrt{n} \rfloor$ if memory allows (see Section 5.7). In general, the user can provide a threshold on memory and if, e.g., by using $m = \lfloor 2\sqrt{n} \rfloor$ this memory threshold would be violated,

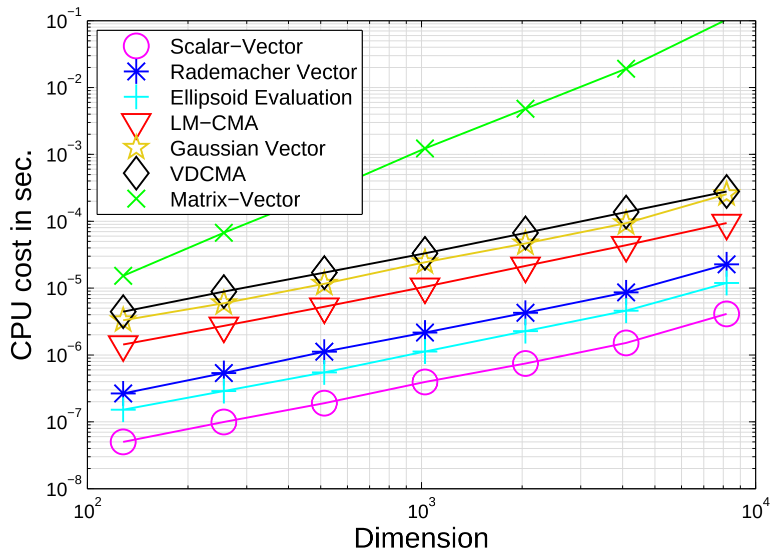

Figure 2: Timing results of LM-CMA and VD-CMA averaged over the whole run on the separable Ellipsoid compared to timing results of simple operations averaged over 100 seconds of experiments. The results for L-BFGS are not shown but for an optimized implementation would be comparable to one scalar-vector multiplication.

the algorithm automatically reduces $m$ to a feasible $m_f$.

### 5.2 Time Complexity

The average amount of CPU time internally required by an algorithm per evaluation of some objective function $f \in \mathbb{R}^n$ (not per algorithm iteration) referred to as time complexity also can limit applicability of certain algorithms to large scale optimization problems. They simply can be too expensive to run, e.g., much more expensive than to perform function evaluations.

Figure 2 shows how fast CPU time per evaluation scales for different operations measured on one 1.8 GHz processor of an Intel Core i7-4500U. Scalar-vector multiplication of a vector with $n$ variables scales linearly with ca. $4 \cdot 10^{-10} n$ seconds, evaluation of the separable Ellipsoid function is 2-3 times more expensive if a temporary data is used. Sampling of $n$ normally distributed variables scales as ca. 60 vectors-scalar multiplications that defines the cost of sampling of unique candidate solutions of many Evolution Strategies such as separable CMA-ES and VD-CMA. However, the sampling of variables according to the Rademacher distribution is about 10 times cheaper. The use of mirrored sampling also decreases the computational burden without worsening the convergence. Finally, the internal computation cost of LM-CMA scales linearly as about 25 scalar-vector multiplications per function evaluation. It is much faster than the lower bound for the original CMA-ES defined by one matrix-vector multiplication required to sample one individual. We observe that the cost of one matrix-vector multiplications costs about $2n$ scalar-vector multiplications, the overhead is probably due to access to matrix members.

The proposed version of LM-CMA is about 10 times faster internally than the original version by Loshchilov (2014) due to the use of mirrored sampling, the Rademacher sampling distribution and sampling with $m^*$ instead of $m$ direction vectors both for $m = 4 + \lfloor 3 \ln n \rfloor$ and $m = \lfloor 2\sqrt{n} \rfloor$. For 8192-dimensional problems it is about 1000 times faster internally than CMA-ES algorithms with the full covariance matrix update (the cost of Cholesky-CMA-ES is given in Loshchilov (2014)).

### 5.3 Invariance under Rank-preserving Transformations of the Objective Function

The LM-CMA belongs to a family of so-called comparison-based algorithms. The performance of these algorithms is unaffected by rank-preserving (strictly monotonically increasing) transformations of the objective function, e.g., whether the function $f$, $f^3$ or $f \times |f|^{-2/3}$ is minimized (Ollivier et al., 2011). Moreover, this invariance property provides robustness to noise as far as this noise does not impact a comparison of solutions of interest (Auger and Hansen, 2013).

In contrast, gradient-based algorithms are sensitive to rank-preserving transformations of $f$. While the availability of gradient information may mitigate the problem that objective functions with the same contours can be solved with a different number of functions evaluations, the lack of gradient information forces the user to estimate it with approaches whose numerical stability is subject to scaling of $f$. Here, we simulate L-BFGS in an idealistic black-box scenario when gradient information is estimated perfectly (we provide exact gradients) but at the cost of $n + 1$ function evaluations per gradient that corresponds to the cost of the forward difference method. Additionally, we investigate the performance of L-BFGS with the central difference method ($2n + 1$ evaluations per gradient) which is twice more expensive but numerically more stable. We denote this method as CL-BFGS.

### 5.4 Invariance under Search Space Transformations

Invariance properties under different search space transformations include translation invariance, scale invariance, rotational invariance and general linear invariance under any full rank matrix $\boldsymbol{R}$ when the algorithm performance on $f(\boldsymbol{x})$ and $f(\boldsymbol{Rx})$ is the same given that the initial conditions of the algorithm are chosen appropriately (Hansen et al., 2011). Thus, the lack of the latter invariance is associated with a better algorithm performance for some $\boldsymbol{R}$ and worse for the others. In practice, it often appears to be relatively simple to design an algorithm specifically for a set of problems with a particular $\boldsymbol{R}$, e.g., identity matrix, and then demonstrate its good performance. If this set contains separable problems, the problems where the optimum can be found with a coordinate-wise search, then even on highly multi-modal functions great results can be easily achieved (Loshchilov et al., 2013). Many derivative-free search algorithms in one or another way exploit problem separability and fail to demonstrate a comparable performance on, e.g., rotated versions of the same problems. This would not be an issue if most of real-world problems are separable, this is, however, unlikely to be the case and some partial-separability or full non-separability are more likely to be present.

The original CMA-ES is invariant w.r.t. any invertible linear transformation of the search space, $\boldsymbol{R}$, if the initial covariance matrix $\mathbf{C}^{t=0} = \boldsymbol{R}^{-1}(\boldsymbol{R}^{-1})^T$, and the initial search point(s) are transformed accordingly Hansen (2006). However, $\boldsymbol{R}$ matrix is often unknown (otherwise, one could directly transform the objective function) and cannot be stored in memory in large scale settings with $n \gg 10,000$. Thus, the covariance matrix adapted by LM-CMA has at most rank $m$ and so the intrinsic coordinate system cannot capture some full rank matrix $\boldsymbol{R}$ entirely. Therefore, the performance of the algorithm on $f(\boldsymbol{Rx})$ compared to $f(\boldsymbol{x})$ depends on $\boldsymbol{R}$. However, in our experiments,

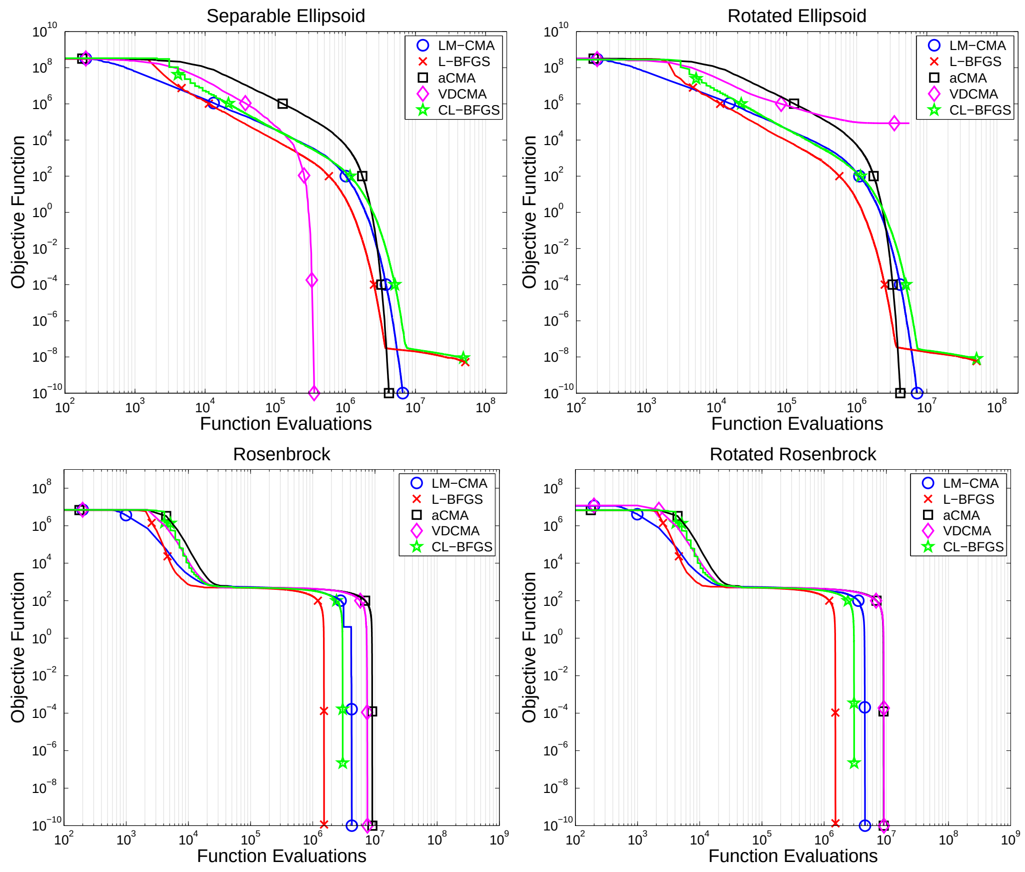

Figure 3: The trajectories show the median of 11 runs of LM-CMA, L-BFGS with exact gradients provided at the cost of $n + 1$ evaluations per gradient, CL-BFGS with central differencing, active CMA-ES and VD-CMA on 512- Separable/Original (Left Column) and Rotated (Right Column) functions.

differences in performance on axis-aligned and rotated ill-conditioned functions were marginal.

Here, we test LM-CMA, aCMA, L-BFGS, CL-BFGS both on separable problems and their rotated versions (see Table 1). It is simply intractable to run algorithms on large scale rotated problems with $n > 1000$ due to the quadratic cost of involved matrix-vectors multiplications (see Figure 2). Fortunately, there is no need to do it for algorithms that are invariant to rotations of the search space since their performance is the same as on the separable problems whose evaluation is cheap (linear in time). Figures 3-4 show that the performance of aCMA on 512-dimensional (the dimensionality still feasible to perform full runs of aCMA) separable (left column) and rotated (right column) problems is very similar and the difference (if any) is likely due to a non-invariant initialization. The invariance in performance is not guaranteed but rather observed for LM-CMA, L-BFGS and CL-BFGS. However, the performance of the VD-CMA degrades significantly on $f_{RotElli}$, $f_{RotDiscus}$ and $f_{RotDiffPow}$ functions due to the restricted form of the adapted covariance matrix of Equation (9). Both $f_{Rosen}$, $f_{Cigar}$ and their rotated

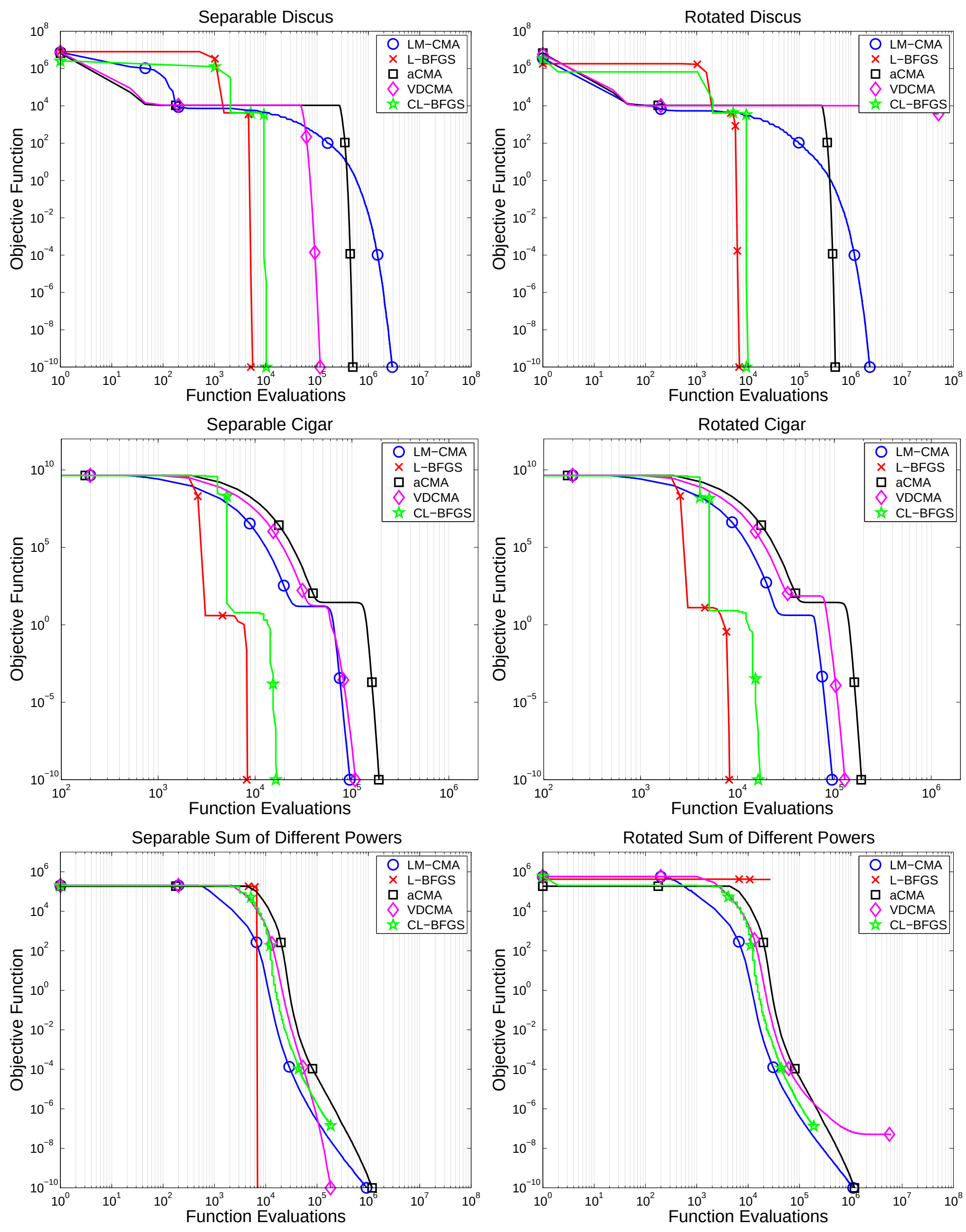

Figure 4: The trajectories show the median of 11 runs of LM-CMA, L-BFGS with exact gradients, CL-BFGS with central differencing, active CMA-ES and VD-CMA on 512-Separable (Left Column) and Rotated (Right Column) functions.

versions can be solved efficiently since they have a Hessian matrix whose inverse can be well approximated by Equation (9) (Akimoto et al., 2014).

An important observation from Figures 3-4 is that even the exact gradient infor-

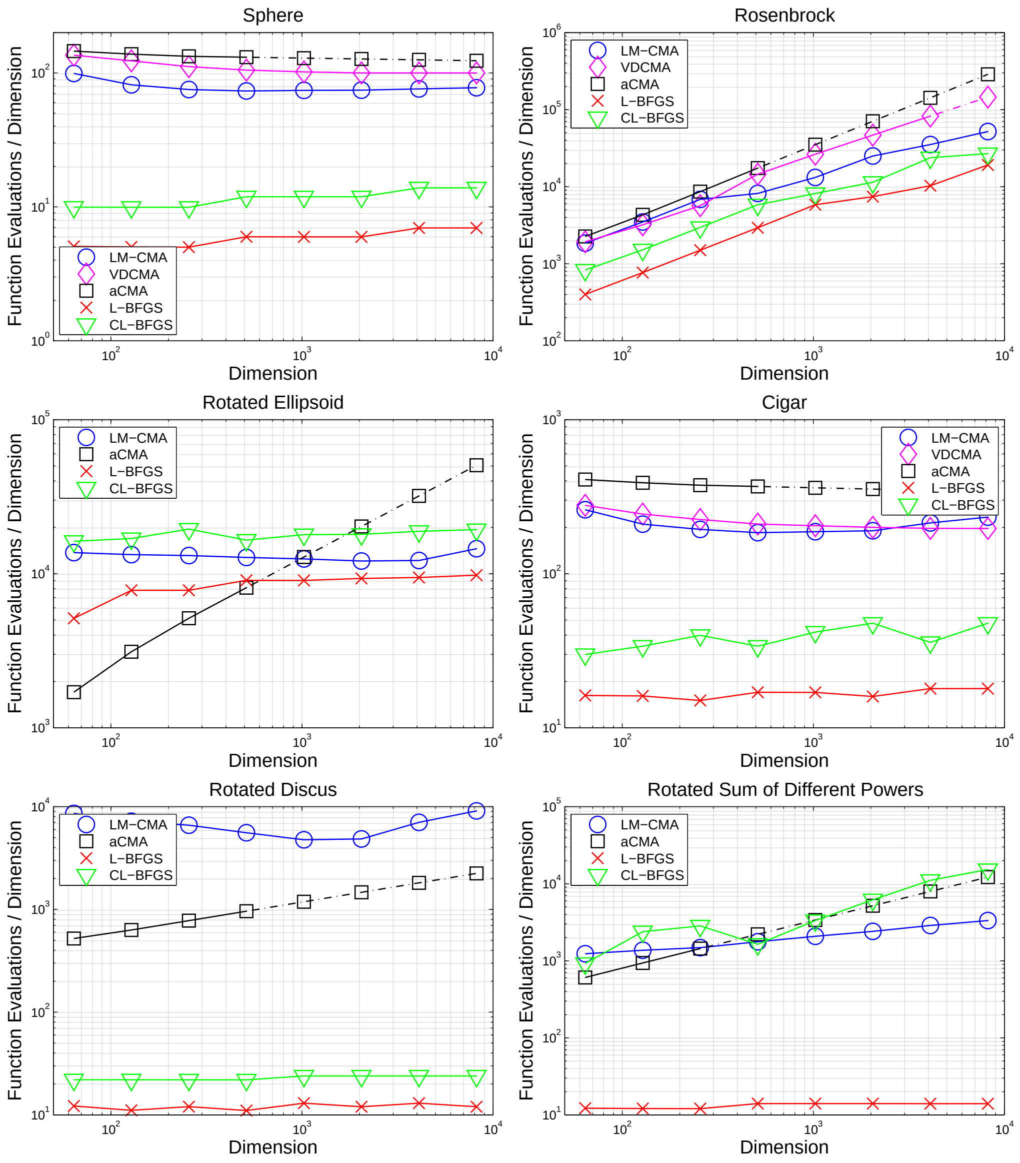

Figure 5: Median (out of 11 runs) number of function evaluations required to find $f(\boldsymbol{x}) = 10^{-10}$ for LM-CMA, L-BFGS with exact gradients, CL-BFGS with central differencing, active CMA-ES and VD-CMA. Dotted lines depict extrapolated results.

mation is not sufficient for L-BFGS to avoid numerical problems which lead to an imprecise estimation of the inverse Hessian matrix and premature convergence on $f_{Elli}$ and $f_{RotElli}$. The L-BFGS with the central difference method (CL-BFGS) experiences an early triggering of stopping criteria on $f_{Rosen}$ and $f_{DiffPow}$. While numerical problems due to imprecise derivative estimations are quite natural for L-BFGS especially on ill-conditioned problems, we assume that with a better implementation of the algorithm (e.g., with high-precision arithmetic) one could obtain a more stable convergence.

Therefore, we extrapolate the convergence curves of L-BFGS and CL-BFGS towards the target $f = 10^{-10}$ after removing the part of the curve which clearly belongs to the stagnation, e.g., $f < 10^{-7}$ on $f_{Elli}$.

## 5.5 Scaling with Problem Dimension

The performance versus the increasing number of problem variables is given in Figure 5. We exclude the results of VD-CMA on some problems because, as can be seen from Figures 3-4, the algorithm does not find the optimum with a reasonable number of function evaluations or/and convergences prematurely. For algorithms demonstrating the same performance on separable and non-separable problems (see Figures 3-4), we plot some results obtained on separable problems as obtained on rotated problems in Figure 5 to avoid possible misunderstanding from designers of separability-oriented algorithms.

The results suggest that L-BFGS shows the best performance, this is not surprising given the form of the selected objective functions (see Table 1). We should also keep in mind that the exact gradients were provided and this still led to premature convergence on some functions (see Figures 3-4). In the black-box scenario, one would probably use L-BFGS with the forward or central (CL-BFGS) difference methods. The latter is often found to lead to a loss by a factor of 2 (as expected due to $2n + 1$ versus $n + 1$ cost per gradient), except for the $f_{RotDiffPow}$, where the loss is increasing with problem dimension.

Quite surprisingly, the LM-CMA outperforms VD-CMA and aCMA on $f_{Sphere}$. This performance is close to the one obtained for (1+1) Evolution Strategy with optimal step-size. Bad performance on $f_{Sphere}$ anyway would not directly mean that an algorithm is useless, but could illustrate its performance in vicinity of local optima when variable-metric algorithms (e.g., CMA-like algorithms) may perform an isotropic search w.r.t. an adapted internal coordinate system. The obtained results are mainly due to the Population Success Rule which deserves an independent study similar to the one by Hansen et al. (2014). Nevertheless, we would like to mention a few key points of the PSR. By design, depending on the target success ratio $z^*$, one can get either biased (for $z^* \neq 0$) or unbiased (for $z^* = 0$) random walk on random functions. It would be a bias to say that either biased or unbiased change of $\sigma$ "is better" on random functions, since the latter depends on the context. Due to the fact that the (weighted) mean of each new population is computed from the best $\mu$ out of $\lambda$ individuals, the $\lambda$ individuals of the new generation are typically as good as the (weighted) best $\mu$ individuals of the previous one, and, thus, if $z^* = 0$ one may expect $z_{PSR} > 0$ from Equation (16). Typically, it is reasonable to choose $z^* \in (0, 0.5)$ lower-bounded by 0 due to random functions and upper-bounded by 0.5 due to linear functions. In this study, we choose 0.3 which lies roughly in the middle of the interval. It is important to mention a striking similarity with the 1/5th success rule (Schumer and Steiglitz, 1968; Rechenberg, 1973). We consider the PSR to be its population-based version.

The performance of LM-CMA on $f_{Elli}$ is probably the most surprising result of this work. In general, the scaling of CMA-ES is expected to be from super-linear to quadratic with $n$ on $f_{Elli}$ since the number of parameters of the full covariance matrix to learn is $(n^2 + n)/2$ (Hansen and Ostermeier, 2001). While aCMA demonstrates this scaling, LM-CMA scales linearly albeit with a significant constant factor. The performance of both algorithms coincides at $n \approx 1000$, then, LM-CMA outperforms aCMA (given that our extrapolation is reasonable) with a factor increasing with $n$. It should be noted that aCMA is slower in terms of CPU time per function evaluation by a factor of

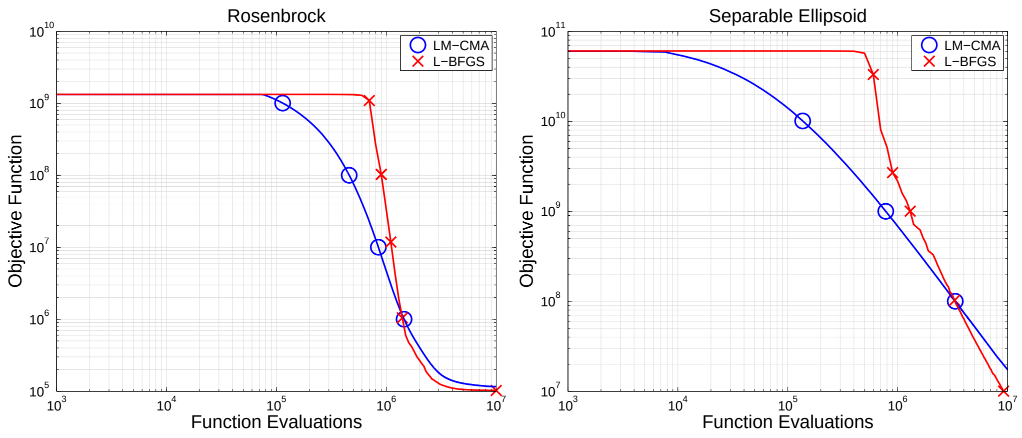

Figure 6: The trajectories show the median of 11 runs of LM-CMA, L-BFGS (with exact gradients provided at the cost of $n+1$ evaluations per gradient) on 100,000-dimensional Rosenbrock and Ellipsoid functions.

$n/10$ (see Figure 2). Another interesting observation is that the L-BFGS is only slightly faster than LM-CMA, while CL-BFGS is actually outperformed by the latter. An insight to these observations can be found in Figure 3 where both LM-CMA and L-BFGS outperform aCMA by a factor of 10 in the initial part of the search, while aCMA compensates this loss by having the covariance matrix well adapted that allows to accelerate convergence close to the optimum. This might be explained as follows: a smaller number of internal parameters defining the intrinsic coordinate system can be learned faster and with greater learning rates, this allows a faster convergence but may slow down the search in vicinity of the optimum if the condition number cannot be captured by the reduced intrinsic coordinate system.

The LM-CMA is better or is as good as VD-CMA on $f_{Rosen}$ and $f_{Cigar}$ where it is expected to be outperformed by the latter due a presumably few principal components needed to be learnt to solve these problems. The scaling on $f_{Rosen}$ suggests that the problem is more difficult (e.g., more difficult than $f_{Elli}$) than one could expect, mainly due to an adaptation of the intrinsic coordinate system required while following the banana shape valley of this function.

The results on 100,000-dimensional problems (see Figure 6) show that LM-CMA outperforms L-BFGS on the first $10n - 20n$ function evaluations which corresponds to the first 10-20 iterations of L-BFGS. This observation suggests that LM-CMA can be viewed as an alternative to L-BFGS when $n$ is large and the available number of function evaluations is limited. While it can provide a competitive performance in the beginning, it is also able to learn dependencies between variables to approach the optimum.

### 5.6 Performance on a nonsmooth variant of Nesterov's function

While designed for smooth optimization, BFGS is known to work well for nonsmooth optimization too. A recent study by Overton (2015) demonstrated the difficulties encountered by BFGS on some nonsmooth functions. We selected one of the test functions from Overton (2015) called the second nonsmooth variant of Nesterov-Chebyshev-Rosenbrock function defined as follows:

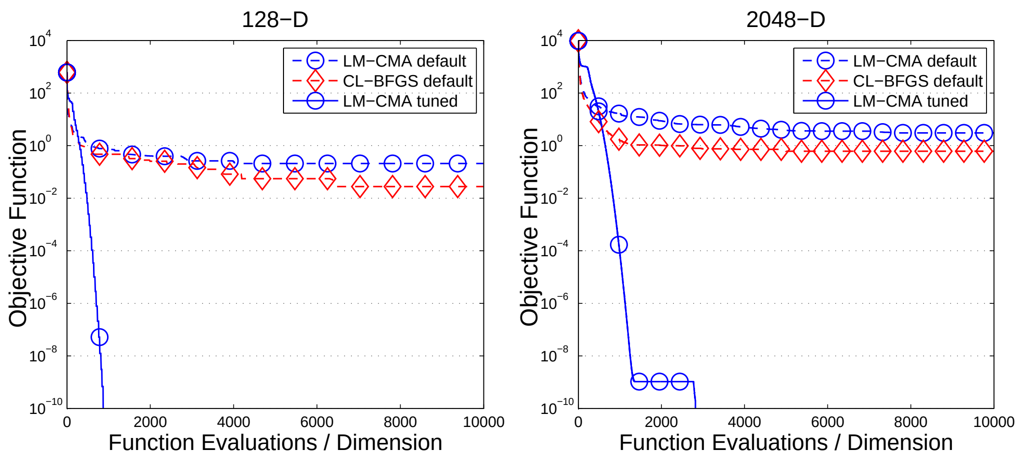

Figure 7: The trajectories show the median of 11 runs of LM-CMA in default settings, CL-BFGS in default settings and tuned LM-CMA (all three algorithms are with restarts) on the second nonsmooth variant of Nesterov-Chebyshev-Rosenbrock function in dimensions 128 and 2048.

$$\hat{N}(x) = \frac{1}{4}\left|x_1 - 1\right| + \sum_{i=1}^{n-1}\left|x_{i+1} - 2\left|x_i\right| + 1\right| \tag{18}$$

This function is nonsmooth (though locally Lipschitz) as well as nonconvex, it has $2^{n-1}$ Clarke stationary points (Overton, 2015). Overton (2015) showed that for $n = 5$ BFGS starting from 1000 randomly generated points finds all 16 Clarke stationary points (for the definition of Clarke stationary points see Abramson and Audet (2006)) and the probability to find the global minimizer is only by about a factor of 2 greater than to find any of the Clarke stationary points. This probability dropped by a factor of 2 for $n = 6$ while and since the number of Clarke stationary points doubled (Overton, 2015). Clearly, the problem becomes extremely difficult for BFGS when $n$ is large.

We launched LM-CMA and CL-BFGS (L-BFGS performed worse) on $\hat{N}(x)$ for $n = 128$ and $n = 2048$. Figure 7 shows that CL-BFGS performs better than LM-CMA, however, both algorithms in default settings and with restarts do not perform well. We tuned both LM-CMA and CL-BFGS but report the results only for LM-CMA since we failed to improve the performance of CL-BFGS by more than one order of magnitude of the objective function value. The tuned parameters for LM-CMA are: i) doubled population size $\lambda$, ii) increased learning rate by 15 to $c_1 = 15/(10\ln(n+1))$, iii) an extremely small learning rate for step-size adaptation $c_\sigma = 0.3/n^2$ instead of $c_\sigma = 0.3$. The last modification is probably the most important, practically, it defines the schedule how step-size decreases. A similar effect can be achieved by reducing $z^*$ or increasing $d_\sigma$. Faster learning of dependencies between variables and slower step-size decrease drastically improve the convergence and the problem can be solved both for $n = 128$ and $n = 2048$ (Figure 7). Interestingly, the number of function evaluations scales almost linearly with problem dimension.

We expected that tuning of CL-BFGS will lead to similar improvements. Surprisingly, our attempts to modify its parameters, often in order to slow down the convergence (e.g., type and number of line-search steps, Wolfe conditions parameters) failed.

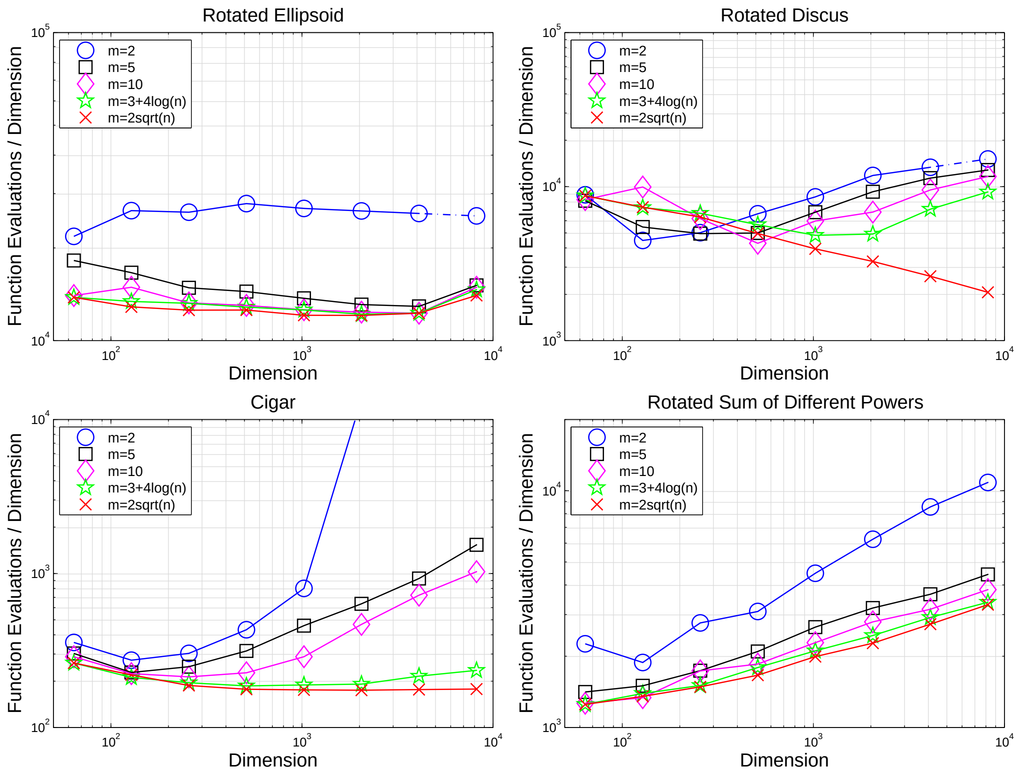

Figure 8: Sensitivity of LM-CMA to different settings of $m$.

We still expect that certain modifications should improve CL-BFGS and thus we leave this question open. The settings tuned for $\hat{N}(x)$ function differ significantly from the default ones. It is of great interest to find an online procedure to adapt them. The next section is aimed at gaining some intuition on parameters importance in LM-CMA.

## 5.7 Sensitivity to Parameters

The black-box scenario implies that the optimization problem at hand is not known, it is therefore hard if even possible to suggest a ”right” parametrization of our algorithm that works best on all problems. Offline tuning in large scale optimization is also computationally expensive. It is rather optimistic to believe that one always can afford enough computational resources to run algorithms till the optimum on very large real-world optimization problems. Nevertheless, we tend to focus on this scenario in order to gain an understanding about scalability on benchmark problems.

Our experience with parameter selection by exclusion of non-viable settings suggests that there exists a dependency between the population size $\lambda$, number of stored vectors $m$, the target temporal distance between them $N_{steps}$, the learning rate $c_c$ for the evolution path and learning rate $c_1$ for the Cholesky factor update. The main reason for this is that all of them impact how well the intrinsic coordinate system defined by the Cholesky factor reflects the current optimization landscape. A posteriori, if $m \ll n$, it seems reasonable to store vectors with a temporal distance in order of $N_{steps} = n$ on problems where a global coordinate system is expected to be constant, e.g., on a class

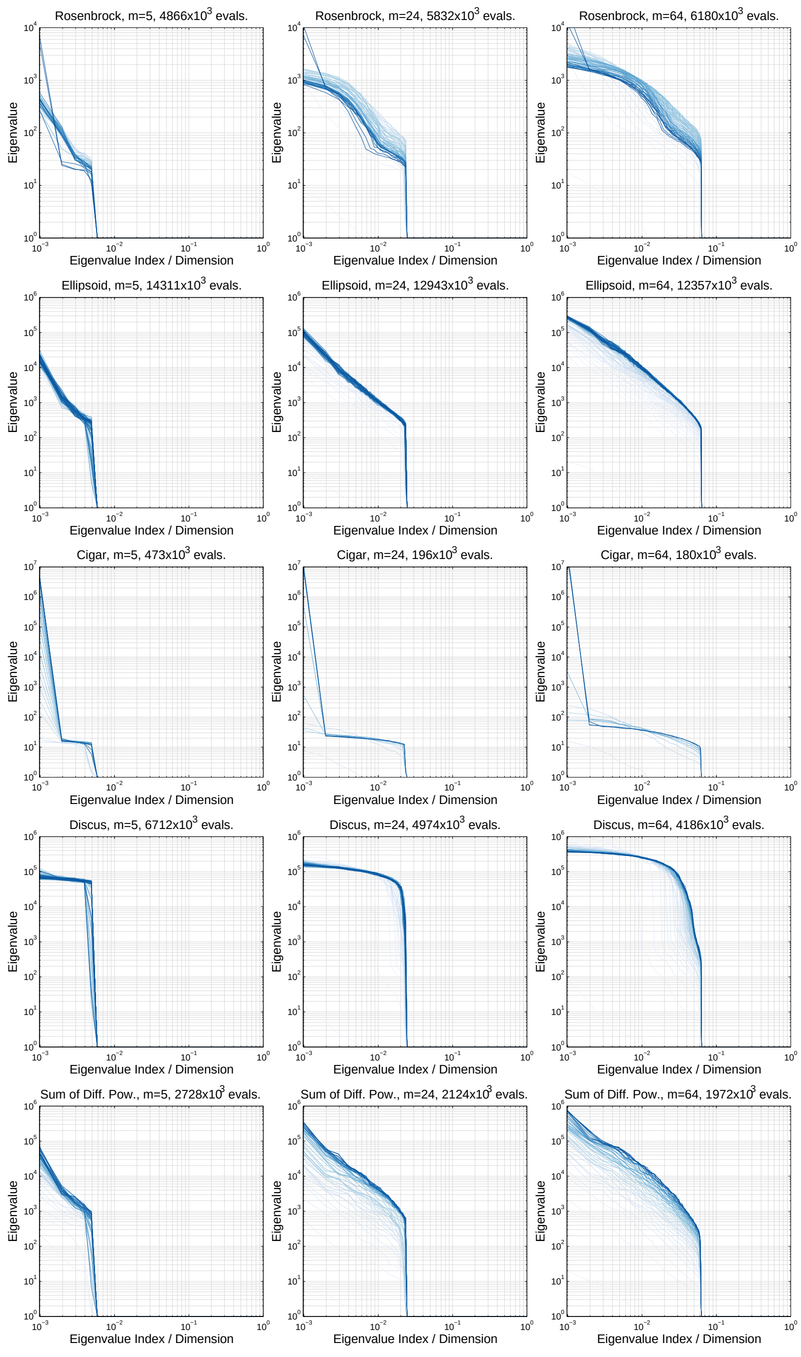

Figure 9: Eigenspectrums of $\mathbf{C}^t = \mathbf{A}^t {\mathbf{A}^t}^T$ for $t$ denoting iteration of LM-CMA with $m$ direction vectors ($m = 5$, $m = 4 + \lfloor 3 \ln 1024 \rfloor = 24$, $m = \lfloor 2\sqrt{n} \rfloor = 64$) on 1024-dimensional problems. Darker (blue) lines correspond to later iterations. The number of function evaluations to reach $f(\boldsymbol{x}) = 10^{-10}$ is given in the title of each sub-figure.

of problems described by the general ellipsoid model (Beyer, 2014). The learning rate for the evolution path is related to both $m$ and $n$, here, we set it to $c_c = \frac{0.5}{\sqrt{n}}$ which is roughly inversely proportional to the (if affordable) suggested $m = \lfloor 2\sqrt{n} \rfloor$. We found that the chosen $c_c$ is still valid for the default $m = 4 + \lfloor 3 \ln n \rfloor$. We do not have a good interpretation for the learning rate $c_1 = \frac{1}{10 \ln(n+1)}$. In general, we are not in favor of strongly arguing for some parameters settings against the others since as already mentioned above they are problem-dependent. A more appropriate approach would be to perform online adaptation of hyper-parameters as implemented for the original CMA-ES by Loshchilov et al. (2014).

We present an analysis for $m$ which directly affects the amount of memory required to run the algorithm, and, thus, is of special interest since the user might be restricted in memory on very large scale optimization problems with $n > 10^6$. Figure 8 shows that the greater the $m$ the better the performance. The results obtained for the default $m = 4 + \lfloor 3 \ln n \rfloor$, i.e., the results demonstrated in the previous sections can be improved with $m = \lfloor 2\sqrt{n} \rfloor$. The improvements are especially pronounced on $f_{Discus}$ functions, where the factor is increasing with $n$ and the overall cost to solve the function reaches the one extrapolated for aCMA at $n = 8192$ (see Figure 5). It is surprising to observe that $m = 5$ and even $m = 2$ are sufficient to solve $f_{Elli}$, $f_{Discus}$ and $f_{DiffPow}$. The latter is not the case for $f_{Cigar}$, where small values of $m$ lead to an almost quadratic growth of run-time. The overall conclusion would be that on certain problems the choice of $m$ is not critical, while greater values of $m$ are preferable in general.

We investigated the eigenspectrum of the covariance matrix $\mathbf{C}^t$ constructed as $\mathbf{A}^t \mathbf{A}^{t^T}$ from the Cholesky factor $\mathbf{A}^t$. The results for single runs on different 1024-dimensional functions and for different $m$ are shown in Figure 9. The evolution of the eigenspectrum during the run is shown by gradually darkening (blue) lines with increasing $t$. Clearly, the number of eigenvalues is determined by $m$. The profiles, e.g., the one of $f_{Cigar}$, also reflect the structure of the problems (see Table 1). The greater the $m$, the greater condition number can be captured by the intrinsic coordinate system as can be see for $f_{Elli}$, $f_{Discus}$ and $f_{DiffPow}$, that in turn leads to a better performance. However, this is not always the case as can be seen for $f_{Rosen}$ that again demonstrates that optimal hyper-parameter settings are problem-dependent.

## 6 Conclusions

We adapt an idea from derivative-based optimization to extend best performing evolutionary algorithms such CMA-ES to large scale optimization. This allows to reduce the cost of optimization in terms of time by a factor of $n/10$ and memory by a factor between $\sqrt{n}$ and $n$. Importantly, it also often reduces the number of function evaluations required to find the optimum. The idea to store a limited number of vectors and use them to adapt an intrinsic coordinate system is not the only but one of probably very few ways to efficiently search in large scale continuous domains. We propose two quite similar alternatives: i) the storage of points and a later estimation of descent directions from differences of these points, and ii) the use of a reduced matrix $m \times n$ as in (Knight and Lunacek, 2007) but with a modified sampling procedure to obtain linear time complexity as proposed for the Adaptive Coordinate Descent by Loshchilov (2013b).

The use of the Population Success Rule is rather optional and alternative step-size adaptation procedures can be applied. However, we find its similarity with the 1/5-th rule quite interesting. The procedure does not make any assumption about the sampling distribution, this allowed to use the Rademacher distribution. When $n$ is large, the sampling from a $n$-dimensional Rademacher distribution resembles the sampling from a $n$-dimensional Gaussian distribution since the probability mass of the latter is concentrated in a thin annulus of width $O(1)$ at radius $\sqrt{n}$.

The presented comparison shows that LM-CMA outperforms other evolutionary algorithms and is comparable to L-BFGS on non-trivial large scale optimization problems when the black-box (derivative-free) scenario is considered. Clearly, the black-box scenario is a pessimistic scenario but a substantial part of works that use finite difference methods for optimization deal with this scenario, and, thus, can consider LM-CMA as an alternative. Importantly, LM-CMA is invariant to rank-preserving transformations of the objective function and therefore is potentially more robust than L-BFGS. The results shown in Figure 7 suggest that the use of a smaller number of direction vectors $m$ can be still efficient, i.e., more efficient algorithms, e.g., with adaptive $m$ (or an adaptive $m \times n$ transformation matrix) can be designed. It seems both promising and feasible to extend the algorithm to constrained, noisy and/or multi-objective optimization, the domains, which are both hardly accessible for L-BFGS and keenly demanded by practitioners. As an important contribution to the success in this direction, it would be helpful to implement online adaptation of internal hyper-parameters as already implemented in the original CMA-ES (Loshchilov, 2014). This would ensure an additional level of invariance and robustness on large scale black-box optimization problems.

## Acknowledgments

I am grateful to Michèle Sebag and Marc Schoenauer for many valuable discussions and insights. I also would like to thank Oswin Krause, Youhei Akimoto and the anonymous reviewers whose interest and valuable comments helped to improve this work.

## References

Abramson, M. A. and Audet, C. (2006). Convergence of mesh adaptive direct search to second-order stationary points. *SIAM Journal on Optimization*, 17(2):606–619.

Ait Elhara, O., Auger, A., and Hansen, N. (2013). A median success rule for non-elitist evolution strategies: Study of feasibility. In *Genetic and Evolutionary Computation Conference*, pages 415–422. ACM.

Akimoto, Y., Auger, A., and Hansen, N. (2014). Comparison-based natural gradient optimization in high dimension. In *Genetic and Evolutionary Computation Conference*, pages 373–380. ACM.

Akimoto, Y. and Ollivier, Y. (2013). Objective improvement in information-geometric optimization. In *Twelfth workshop on Foundations of Genetic Algorithms XII*, pages 1–10. ACM.

Andrew, G. and Gao, J. (2007). Scalable training of l 1-regularized log-linear models. In *24th International Conference on Machine Learning*, pages 33–40. ACM.

Arnold, D. V. (2014). On the behaviour of the (1,λ)-ES for conically constrained linear problems. *Evolutionary Computation*, 22(3):503–523.

Auger, A., Finck, S., Hansen, N., and Ros, R. (2010). BBOB 2010: Comparison Tables of All Algorithms on All Noiseless Functions. Technical Report RR-7215, INRIA.

Auger, A. and Hansen, N. (2013). Linear Convergence of Comparison-based Step-size Adaptive Randomized Search via Stability of Markov Chains. *arXiv preprint arXiv:1310.7697*.

Becker, S. and Fadili, J. (2012). A quasi-newton proximal splitting method. In *Advances in Neural Information Processing Systems*, pages 2618–2626.

Beyer, H.-G. (2014). Convergence analysis of evolutionary algorithms that are based on the paradigm of information geometry. *Evolutionary Computation*, 22(4):679–709.

Brand, M. (2006). Fast low-rank modifications of the thin singular value decomposition. *Linear algebra and its applications*, 415(1):20–30.

Brockhoff, D., Auger, A., Hansen, N., Arnold, D. V., and Hohm, T. (2010). Mirrored sampling and sequential selection for evolution strategies. In *Parallel Problem Solving from Nature–PPSN*, pages 11–21. Springer.

Byrd, R. H., Lu, P., Nocedal, J., and Zhu, C. (1995). A limited memory algorithm for bound constrained optimization. *SIAM Journal on Scientific Computing*, 16(5):1190–1208.

Dennis Jr, J. E. and Schnabel, R. B. (1996). *Numerical methods for unconstrained optimization and nonlinear equations*, volume 16. SIAM.

Finck, S., Hansen, N., Ros, R., and Auger, A. (2010). Real-parameter black-box optimization benchmarking 2010: Experimental setup. Technical Report 2009/21, Research Center PPE.

García, S., Molina, D., Lozano, M., and Herrera, F. (2009). A study on the use of non-parametric tests for analyzing the evolutionary algorithms' behaviour: a case study on the CEC'2005 Special Session on Real Parameter Optimization. *Journal of Heuristics*, 15:617–644.

Glasmachers, T. (2012). Convergence of the IGO-Flow of isotropic gaussian distributions on convex quadratic problems. In *Parallel Problem Solving from Nature–PPSN*, pages 1–10. Springer Berlin Heidelberg.

Hansen, N. (2006). The cma evolution strategy: a comparing review. In *Towards a new evolutionary computation*, pages 75–102. Springer.

Hansen, N. (2008). Adaptive encoding: How to render search coordinate system invariant. In *Parallel Problem Solving from Nature–PPSN*, pages 205–214. Springer.

Hansen, N., Arnold, D. V., and Auger, A. (2015). Evolution strategies. In *Springer Handbook of Computational Intelligence*, pages 871–898. Springer.

Hansen, N., Atamna, A., and Auger, A. (2014). How to Assess Step-Size Adaptation Mechanisms in Randomised Search. In *Parallel Problem Solving from Nature–PPSN*, pages 60–69. Springer.

Hansen, N. and Auger, A. (2014). Principled design of continuous stochastic search: From theory to practice. In *Theory and Principled Methods for the Design of Metaheuristics*, pages 145–180. Springer Berlin Heidelberg.

Hansen, N., Müller, S., and Koumoutsakos, P. (2003). Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES). *Evolutionary Computation*, 11(1):1–18.

Hansen, N., Niederberger, A. S., Guzzella, L., and Koumoutsakos, P. (2009). A method for handling uncertainty in evolutionary optimization with an application to feedback control of combustion. *Evolutionary Computation, IEEE Transactions on*, 13(1):180–197.

Hansen, N. and Ostermeier, A. (1996). Adapting Arbitrary Normal Mutation Distributions in Evolution Strategies: The Covariance Matrix Adaptation. In *International Conference on Evolutionary Computation*, pages 312–317.

Hansen, N. and Ostermeier, A. (2001). Completely Derandomized Self-Adaptation in Evolution Strategies. *Evolutionary Computation*, 9(2):159–195.

Hansen, N. and Ros, R. (2010). Benchmarking a weighted negative covariance matrix update on the BBOB-2010 noiseless testbed. In *Genetic and Evolutionary Computation Conference*, pages 1673–1680. ACM.

Hansen, N., Ros, R., Mauny, N., Schoenauer, M., and Auger, A. (2011). Impacts of invariance in search: When CMA-ES and PSO face ill-conditioned and non-separable problems. *Applied Soft Computing*, 11(8):5755–5769.

Hopcroft, J. and Kannan, R. (2015). *Foundations of Data Science*. to appear.

Igel, C., Hansen, N., and Roth, S. (2007). Covariance matrix adaptation for multi-objective optimization. *Evolutionary computation*, 15(1):1–28.

Jastrebski, G. A. and Arnold, D. V. (2006). Improving Evolution Strategies through Active Covariance Matrix Adaptation. In *IEEE Congress on Evolutionary Computation*, pages 2814–2821.

Kern, S., Hansen, N., and Koumoutsakos, P. (2006). Local Meta-Models for Optimization Using Evolution Strategies. In *PPSN*, pages 939–948. LNCS 4193, Springer Verlag.

Knight, J. N. and Lunacek, M. (2007). Reducing the space-time complexity of the CMA-ES. In *Genetic and Evolutionary Computation Conference*, pages 658–665. ACM.

Krause, O. (2014). Personal communication.

Li, Z.-C., Chien, C.-S., and Huang, H.-T. (2007). Effective condition number for finite difference method. *Journal of computational and applied mathematics*, 198(1):208–235.

Liu, D. C. and Nocedal, J. (1989). On the limited memory BFGS method for large scale optimization. *Mathematical programming*, 45(1-3):503–528.

Loshchilov, I. (2013a). CMA-ES with restarts for solving CEC 2013 benchmark problems. In *Evolutionary Computation (CEC), 2013 IEEE Congress on*, pages 369–376. IEEE.

Loshchilov, I. (2013b). *Surrogate-Assisted Evolutionary Algorithms*. PhD thesis, Université Paris Sud-Paris XI.

Loshchilov, I. (2014). A Computationally Efficient Limited Memory CMA-ES for Large Scale Optimization. In *Genetic and Evolutionary Computation Conference*, pages 397–404. ACM.

Loshchilov, I., Schoenauer, M., and Sebag, M. (2011). Adaptive Coordinate Descent. In *Genetic and Evolutionary Computation Conference*, pages 885–892. ACM.

Loshchilov, I., Schoenauer, M., and Sebag, M. (2012). Self-adaptive Surrogate-Assisted Covariance Matrix Adaptation Evolution Strategy. In *Genetic and Evolutionary Computation Conference*, pages 321–328. ACM.

Loshchilov, I., Schoenauer, M., and Sebag, M. (2013). Bi-population CMA-ES agorithms with surrogate models and line searches. In *Genetic and Evolutionary Computation Conference*, pages 1177–1184. ACM.

Loshchilov, I., Schoenauer, M., Sebag, M., and Hansen, N. (2014). Maximum Likelihood-based Online Adaptation of Hyper-parameters in CMA-ES. In *Parallel Problem Solving from Nature–PPSN*, pages 70–79. Springer International Publishing.

Ngiam, J., Coates, A., Lahiri, A., Prochnow, B., Le, Q. V., and Ng, A. Y. (2011). On optimization methods for deep learning. In *International Conference on Machine Learning*, pages 265–272.

Nocedal, J. (1980). Updating quasi-newton matrices with limited storage. *Math. of computation*, 35(151):773–782.

Ollivier, Y., Arnold, L., Auger, A., and Hansen, N. (2011). Information-geometric optimization algorithms: A unifying picture via invariance principles. *arXiv preprint arXiv:1106.3708*.

Overton, M. L. (2015). Nonsmooth, nonconvex optimization algorithms and examples. Slides of a talk given at Challenges in Optimization for Data Science, July 1–2, 2015, Universite Pierre et Marie Curie – Paris 6, Paris, France.

Rechenberg, I. (1973). *Evolutionsstrategie: optimierung technischer systeme nach prinzipien der biologischen evolution*. Frommann-Holzboog.

Ros, R. and Hansen, N. (2008). A simple modification in CMA-ES achieving and space complexity. In *Parallel Problem Solving from Nature–PPSN*, pages 296–305. Springer.

Schmidt, M. (2005). Minfunc. *Technical report, Laboratoire d'Informatique de l'Ecole Normale Supérieure, Paris*.

Schumer, M. and Steiglitz, K. (1968). Adaptive step size random search. *Automatic Control, IEEE Transactions on*, 13(3):270–276.

Shanno, D. F. (1970). Conditioning of Quasi-Newton Methods for Function Minimization. *Math. of Computation*, 24(111):647–656.

Sun, Y., Gomez, F., Schaul, T., and Schmidhuber, J. (2011). A linear time natural evolution strategy for non-separable functions. *arXiv preprint arXiv:1106.1998*.

Suttorp, T., Hansen, N., and Igel, C. (2009). Efficient covariance matrix update for variable metric evolution strategies. *Machine Learning*, 75(2):167–197.

Wierstra, D., Schaul, T., Glasmachers, T., Sun, Y., Peters, J., and Schmidhuber, J. (2014). Natural evolution strategies. *The Journal of Machine Learning Research*, 15(1):949–980.

Wolfe, P. (1969). Convergence conditions for ascent methods. *SIAM review*, 11(2):226–235.

Yao, X. and Liu, Y. (1997). Fast evolution strategies. In *Evolutionary Programming VI*, pages 149–161. Springer.