A More Efficient Rank-one Covariance Matrix Update for Evolution Strategies

Oswin Krause
Department of Computer Science
University of Copenhagen
2100 Copenhagen, Denmark
oswin.krause@di.ku.dk

Christian Igel
Department of Computer Science
University of Copenhagen
2100 Copenhagen, Denmark
igel@di.ku.dk

ABSTRACT

Learning covariance matrices of Gaussian distributions is at the heart of most variable-metric randomized algorithms for continuous optimization. If the search space dimensionality is high, updating the covariance or its factorization is computationally expensive. Therefore, we adopt an algorithm from numerical mathematics for rank-one updates of Cholesky factors. Our methods results in a quadratic time covariance matrix update scheme with minimal memory requirements. The numerically stable algorithm leads to triangular Cholesky factors. Systems of linear equations where the linear transformation is defined by a triangular matrix can be solved in quadratic time. This can be exploited to avoid the additional iterative update of the inverse Cholesky factor required in some covariance matrix adaptation algorithms proposed in the literature. When used together with the (1+1)-CMA-ES and the multi-objective CMA-ES, the new method leads to a memory reduction by a factor of almost four and a faster covariance matrix update. The numerical stability and runtime improvements are demonstrated on a set of benchmark functions.

Keywords

CMA-ES, covariance matrix adaptation, rank-one update, Cholesky factorization

1. INTRODUCTION

Adapting the covariance matrix of additive variation operators is a key concept in evolution strategies (ES, [3]) and estimation of distribution algorithms [11]. In many applications, the time needed for the covariance matrix update can be neglected, because the algorithm's computation time is dominated by the objective function evaluations. However, for high dimensional problems and/or objective functions that can be evaluated quickly, the covariance matrix update may become a bottleneck. For example, in the covariance matrix adaptation ES (CMA-ES, [7, 6]), the number of objective function evaluations per iteration typically increases sub-linearly with the search space dimensionality n (e.g., $\lambda = 3 + \lfloor 3 \ln n \rfloor$, see [6]), while the cost of updating the full $n \times n$ covariance matrix Σ of the search distribution (or a factorization of Σ) is $\Omega(n^2)$. Most often the update further requires a decomposition of Σ in cubic time.

For large n, also the memory requirements can become relevant, in particular in the multi-objective CMA-ES (MO-CMA-ES, [8, 17]), where an individual covariance matrix is stored for each individual.

Against this background, several ways to reduce the complexity of the covariance matrix adaptation have been proposed. The most basic strategy to reduce the time complexity is to update the covariance matrix only every $o(n)$ iterations [7, 6]. However, this implies that the algorithm does not optimally exploit the gathered information. And indeed, postponing updates can lead to an increase in the number of objective function evaluations needed to achieve a desired objective function value [16].

Another approach is to impose restrictions on the covariance matrices (e.g., [13, 14, 15, 1, 12]), that is, on the dependencies (i.e., representations or encodings [5]), that can be learnt. This way, linear time and space complexity can be achieved. For example, Ros and Hansen present a CMA-ES variant restricting the covariance matrices to be diagonal [14], which works fine on separable objective functions. The recent variant proposed in [1] additionally adapts one principle component, which results in $2n$ parameters for representing Σ. Recently, Loshchilov proposed a different method [12], which stores a subset of vectors that keep track of previous successful steps and adapts the matrix to make big steps in these directions. These approaches restrict the generality of the CMA-ES and can lead to a significant decrease in performance if the objective function does not match the imposed restrictions (e.g., if there are many strong dependencies between the variables).

To speed up the general covariance update, Igel et al. [10] proposed a fast rank-one update of a non-triangular Cholesky factor L of Σ (i.e., $\Sigma = LL^T$) when the update vector is a previously sampled point from a normal distribution with covariance Σ. The result was generalized in [16] to handle arbitrary directions, which is necessary because it has been proven to be beneficial to consider updates that do not just depend on the last step in the search space (e.g., to use the concept of an *evolution path*), by simultaneously updating the inverse factor. Arnold and Hansen [2] extended the results to the case of the rank-one *downdate*, which can be used to penalize certain search directions. The computation time for the update of Σ with this type of algorithms is asymptot-

FOGA'15, January 17–20, 2015, Aberystwyth, UK.

ACM 978-1-4503-3434-1/15/01 ...$15.00.
http://dx.doi.org/10.1145/2725494.2725496.

ically optimal and a rank m update requires $O(mn^2)$ computation time. These Cholesky-update based methods considerably speed up the (1+1)-CMA-ES [10, 2] and in particular the MO-CMA-ES [8, 17], which can be viewed as running several (1+1)-CMA-ESs at the same time. However, they have two obvious drawbacks. First, instead of Σ they maintain the not necessarily triangular Cholesky factor L. The methods allowing for general updates additionally store the inverse factor L^{-1}. That is, instead of $n(n+1)/2$ parameters, $2n^2$ values have to be kept in memory. Second, L and L^{-1} are updated independently of each other. This is not elegant and bears the risk that the stored matrices L and L^{-1} diverge in the sense that one is not the inverse of the other due to numerical problems. However, the latter has not been observed to cause problems in practice, see [16].

Both issues do not occur if L is triangular. First, then L has only $n(n+1)/2$ parameters. Second, instead of storing and iteratively updating L^{-1}, which is only needed in the update to compute $\boldsymbol{w} = L^{-1}\boldsymbol{v}$, one can now solve $L\boldsymbol{w} = \boldsymbol{v}$ for $\boldsymbol{w}$ in quadratic time. Furthermore, the above-mentioned divergence problem cannot occur.

It turns out that the problem of a fast and efficient rank-one update of triangular Cholesky factors is a classical question in numerical mathematics. It has been solved in 1974 by Gill et al. [4]. In their work, the authors present various $O(n^2)$ updates resulting in triangular Cholesky factors. Versions of the algorithms have been implemented in most major software-libraries manipulating matrices, including Matlab, LAPACK and the Eigen C++ library.

This paper shows how to adopt Gill et al.'s algorithms for time and memory efficient as well as numerically stable covariance matrix up- and downdates. We will introduce the notation and recall the covariance matrix update currently used in the (1+1)-CMA-ES in section 2. In section 3, one variant of the triangular Cholesky update is described. The next section shows how to use this update as part of the (1+1)-CMA-ES. Section 4 presents timing experiments, and we conclude in section 5.

2. COVARIANCE MATRIX UPDATES AND THE CHOLESKY FACTOR

In the following, we will denote vectors by lower-case bold italics (e.g., $\boldsymbol{v} \in \mathbb{R}^N$), scalar values by lower-case italics (e.g., $v \in \mathbb{R}$), and matrices by captal letters (e.g., $V \in \mathbb{R}^{n\times n}$). Sampling from an n-dimensional multi-variate normal distribution $\mathcal{N}(\boldsymbol{m}, \Sigma)$, $\boldsymbol{m} \in \mathbb{R}^n$,$\Sigma \in \mathbb{R}^{n\times n}$ is usually done using a decomposition of the covariance matrix Σ. This could be the square root of the matrix $\Sigma = HH \in \mathbb{R}^{n\times n}$ or some arbitrary (triangular or non-triangular) Cholesky factorization $\Sigma = LL^T$, which is related to the square root by $L = HE$ where E is an orthogonal matrix. We can sample a point $\boldsymbol{z}$ from $\mathcal{N}(\boldsymbol{m}, \Sigma)$ using a sample $\boldsymbol{z} \sim \mathcal{N}(\boldsymbol{0}, I)$ by

$$\boldsymbol{x} = L\boldsymbol{z} + \boldsymbol{m} = HE\boldsymbol{z} + \boldsymbol{m} = H\boldsymbol{y} + \boldsymbol{m} \ ,$$

where we set $\boldsymbol{y} = E\boldsymbol{z}$. This holds as $\boldsymbol{y} \sim \mathcal{N}(\boldsymbol{0}, I)$ since E is orthogonal. Thus, as long as we are only interested in the value of $\boldsymbol{x}$ and do not need $\boldsymbol{y}$, we can sample using the Cholesky factor instead of the matrix square root.

Adaptation of the covariance matrix is an important building block of evolutionary strategies. For example, the (1+1)-CMA-ES creates a sequence of covariance matrices $\Sigma_k \in \mathbb{R}^{n\times n}$, $k = 1, \ldots, T$, which are related by a rank-one-update

$$\Sigma_{k+1} = \alpha_k \Sigma_k + \beta_k \boldsymbol{v}_k \boldsymbol{v}_k^T \ ,$$

where $\alpha_k, \beta_k \in \mathbb{R}$ and $\boldsymbol{v}_k \in \mathbb{R}^n$, $k = 1, \ldots, T$. As we are not interested in Σ_k itself but only in acquiring samples from the normal distribution $\mathcal{N}(\boldsymbol{0}, \Sigma_k)$, our goal is to find a simple update to H_k or L_k such that $\Sigma_{k+1} = H_{k+1}H_{k+1} = L_{k+1}L_{k+1}^T$ under the constraint of $O(n^2)$ time and memory requirement. It turns out that there exists a fast update for the Cholesky factor L_k but to date no formula for H_k is known. The update of L_k is known from numerical mathematics

$$L_{k+1} = \sqrt{\alpha_k} L_k \left(I + \gamma_k \frac{\boldsymbol{w}_k \boldsymbol{w}_k^T}{||\boldsymbol{w}_k||^2} \right) \ ,$$

where $\boldsymbol{w}_k = L_k^{-1}\boldsymbol{v}_k$, I is the $n \times n$ identity matrix and $\gamma_k = \sqrt{1 + \frac{\beta_k}{\alpha_k}||\boldsymbol{w}_k||^2} - 1$, and has been applied in the context of the (1+1)-CMA-ES in [10].

If the update direction $\boldsymbol{v_k}$ corresponds to a sample $L_k\boldsymbol{z}$, $\boldsymbol{z} \sim \mathcal{N}(\boldsymbol{0}, I)$, then we get $\boldsymbol{w} = \boldsymbol{z}$ for free. However, this is in general not the case, in particular when an evolution path is used. The evolution path is a weighted average of successful steps, and the potential sample that could have created this step has to be found and we have to compute $\boldsymbol{w} = L_k^{-1}\boldsymbol{v}_k$.

The update above has the disadvantage that the Cholesky factor L_{k+1} is not triangular, even if L_k was. Thus, L_{k+1}^{-1} has to be updated and stored separately for being able to calculate $\boldsymbol{w}$ in $O(n^2)$ time [16]:

$$L_{k+1}^{-1} = \left(I - \frac{\gamma_k}{\gamma_k + 1} \frac{\boldsymbol{w}_k \boldsymbol{w}_k^T}{||\boldsymbol{w}_k||^2} \right) \frac{1}{\sqrt{\alpha_k}} L_k^{-1} \qquad (2.1)$$

3. A MORE EFFICIENT, TRIANGULAR UPDATE

Even though the $\Theta(n^2)$ time and space complexity of the update rule described above is asymptotically optimal, it is not fully satisfactory. Maintaining two non symmetric $\mathbb{R}^{n\times n}$ matrices for updating the $n(n+1)/2$ parameters of Σ is inefficient.

Thus, we propose to replace this strategy by an update rule that ensures that the Cholesky factors are triangular. Then one has just to store the $n(n+1)/2$ parameters of the Cholesky factor L_k, because it is possible to solve systems of equations of the form $L_k\boldsymbol{w} = \boldsymbol{v}$ using backwards substitution efficiently in quadratic time.

Gill et al. [4] found solutions to the problem of updating triangular Cholesky factors in 1974. The authors offer a number of algorithms for update and downdate realizing different trade-offs between speed and numerical accuracy. Interestingly, one of the algorithms can already be derived by applying the standard decomposition algorithm. Other updates using Gibbs or Householder transformations can be developed as well, giving more numeric accuracy or having faster implementations. We derive the triangular Cholesky update using the Cholesky factorisation algorithm as a simple example in the following Lemma 1. Without loss of generality, we will now assume $\alpha = 1$.

LEMMA 1. *Let $A = LL^T \in \mathbb{R}^{n\times n}$ be symmetric positive definite and L be a lower triangular matrix. Define a partition of A and L into blocks*

$$A = \begin{pmatrix} a_{11} & \boldsymbol{a}_{21}^T \\ \boldsymbol{a}_{21} & A_{22} \end{pmatrix} = \begin{pmatrix} l_{11} & 0 \\ \boldsymbol{l}_{21} & L_{22} \end{pmatrix} \begin{pmatrix} l_{11} & \boldsymbol{l}_{21}^T \\ 0 & L_{22}^T \end{pmatrix} \ ,$$

with $a_{11}, l_{11} \in \mathbb{R}$, $\boldsymbol{a}_{21}, \boldsymbol{l}_{21} \in \mathbb{R}^{n-1}$ *and* $A_{22}, L_{22} \in \mathbb{R}^{n-1\times n-1}$. *Let*

$$A' = A + \beta \boldsymbol{v}\boldsymbol{v}^T \in \mathbb{R}^{n\times n}$$

with $\beta \in \mathbb{R}$ *and* $\boldsymbol{v} = (v_1, \boldsymbol{v}_2) \in \mathbb{R}^n$, $v_1 \in \mathbb{R}$, $\boldsymbol{v}_2 \in \mathbb{R}^{n-1}$. *There exists a lower triangular matrix* $L' \in \mathbb{R}^{n\times n}$ *such that* $A' = L'L'^T$. *Let* A' *and* L' *have the same block structure as* A *and* L, *respectively. It holds for* L'*:*

$$\begin{aligned} l'_{11} &= \sqrt{l_{11}^2 + \beta v_1^2} \\ \boldsymbol{l'}_{21} &= \frac{l_{11}\boldsymbol{l}_{21} + \beta v_1 \boldsymbol{v}_2}{l'_{11}} \\ L'_{22}L'^T_{22} &= L_{22}L_{22}^T + \beta \frac{l_{11}^2}{l'^2_{11}} \left(\boldsymbol{v}_2 - \frac{v_1}{l_{11}}\boldsymbol{l}_{21}\right)\left(\boldsymbol{v}_2 - \frac{v_1}{l_{11}}\boldsymbol{l}_{21}\right)^T \end{aligned}$$

Proof. Using the block definition of A' the product $L'L'^T$ reads:

$$A' = \begin{pmatrix} l'^2_{11} & l'_{11}\boldsymbol{l'}^T_{21} \\ l'_{11}\boldsymbol{l'}_{21} & L'_{22}L'^T_{22} + \boldsymbol{l'}_{21}\boldsymbol{l'}^T_{21} \end{pmatrix} .$$

Using this we can now determine l'_{11} from a'_{11}, $\boldsymbol{l'}_{21}$ from A'_{21} and L'_{22} from A'_{22}. For l'_{11} holds

$$l'_{11} = \sqrt{a'_{11}} = \sqrt{a_{11} + \beta v_1^2} = \sqrt{l_{11}^2 + \beta v_1^2}$$

and for $\boldsymbol{l'}_{21}$ holds

$$\boldsymbol{l'}_{21} = \frac{A'_{21}}{l'_{11}} = \frac{\boldsymbol{a}_{21} + \beta v_1 \boldsymbol{v}_2}{l'_{11}} = \frac{l_{11}\boldsymbol{l}_{21} + \beta v_1 \boldsymbol{v}_2}{l'_{11}} .$$

Applying the above formulas to $L'_{22}L'^T_{22}$ leads to:

$$\begin{aligned} L'_{22}L'^T_{22} &= A'_{22} - \boldsymbol{l'}_{21}\boldsymbol{l'}^T_{21} \\ &= L_{22}L_{22}^T + \boldsymbol{l}_{21}\boldsymbol{l}_{21}^T + \beta \boldsymbol{v}_2\boldsymbol{v}_2^T \\ &\quad - \frac{l_{11}\boldsymbol{l}_{21} + \beta v_1\boldsymbol{v}_2}{l'_{11}}\left(\frac{l_{11}\boldsymbol{l}_{21} + \beta v_1\boldsymbol{v}_2}{l'_{11}}\right)^T \\ &= L_{22}L_{22}^T + \beta\left(1 - \frac{\beta v_1^2}{l'^2_{11}}\right)\boldsymbol{v}_2\boldsymbol{v}_2^T + \left(1 - \frac{l_{11}^2}{l'^2_{11}}\right)\boldsymbol{l}_{21}\boldsymbol{l}_{21}^T \\ &\quad - \frac{\beta v_1 l_{11}}{l'^2_{11}}\left(\boldsymbol{l}_{21}\boldsymbol{v}_2^T + \boldsymbol{v}_2\boldsymbol{l}_{21}^T\right) \\ &= L_{22}L_{22}^T + \frac{\beta l_{11}^2}{l'^2_{11}}\boldsymbol{v}_2\boldsymbol{v}_2^T + \frac{\beta v_1^2}{l'^2_{11}}\boldsymbol{l}_{21}\boldsymbol{l}_{21}^T \\ &\quad - \frac{\beta v_1 l_{11}}{l'^2_{11}}\left(\boldsymbol{l}_{21}\boldsymbol{v}_2^T + \boldsymbol{v}_2\boldsymbol{l}_{21}^T\right) \\ &= L_{22}L_{22}^T + \beta\frac{l_{11}^2}{l'^2_{11}}\left(\boldsymbol{v}_2 - \frac{v_1}{l_{11}}\boldsymbol{l}_{21}\right)\left(\boldsymbol{v}_2 - \frac{v_1}{l_{11}}\boldsymbol{l}_{21}\right)^T \end{aligned}$$

□

Thus, considering a rank-1 update of an $n \times n$ matrix A, we can update one row and one column of a triangular Cholesky decomposition of A in linear time. The remaining entries can be computed by a rank-1 update of an $(n-1)\times(n-1)$ matrix. Thus, if we apply this idea iteratively, we get the full update in quadratic time:

Corollary 1. *Let* $A = LL^T \in \mathbb{R}^{n\times n}$ *and* $A' = L'L'^T \in \mathbb{R}^{n\times n}$ *as in Lemma 1. Let* $j \in \{1, \ldots, n\}$ *and define a partition of* L *and* L' *into blocks*

$$L = \begin{pmatrix} B_{j,j} & 0 \\ B_{n-j,j} & B_{n-j,n-j} \end{pmatrix}, L' = \begin{pmatrix} B'_{j,j} & 0 \\ B'_{n-j,j} & B'_{n-j,n-j} \end{pmatrix}$$

where $B_{j,j}, B'_{j,j} \in \mathbb{R}^{j\times j}$, $B_{n-j,j}, B'_{n-j,j} \in \mathbb{R}^{n-j\times j}$ *and* $B_{n-j,n-j}, B'_{n-j,n-j} \in \mathbb{R}^{n-j\times n-j}$. *There exists* $\beta_{j+1} \in \mathbb{R}$ *and* $\boldsymbol{w}_{j+1} \in \mathbb{R}^{n-j}$ *such that*

$$B'_{n-j,n-j}B'^T_{n-j,n-j} = B_{n-j,n-j}B^T_{n-j,n-j} + \beta_{j+1}\boldsymbol{\omega}_{j+1}\boldsymbol{\omega}^T_{j+1}$$

Proof. In the following, we sketch a proof by induction. Setting $j = 1$ we can apply Lemma 1 directly, by setting $l'_{11} = B'_{j,j}$, $\boldsymbol{l'}_{21} = B'_{n-j,j}$ and $L'_{22} = B'_{n-j,n-j}$. We see from Lemma 1 that $L'_{22}L'^T_{22}$ is again a rank-1 update to the submatrix $L_{22}L_{22}^T$ of A by setting $\beta_2 = \beta\frac{l_{11}^2}{l'^2_{11}}$ and $\boldsymbol{\omega}_2 = \boldsymbol{v}_2 - \frac{v_1}{l_{11}}\boldsymbol{l}_{21}$. We can thus apply Lemma 1 iteratively on the block $B'_{n-j,n-j}$, which prooves the result. □

From the lemma above it follows that L' can be computed in a column-wise fashion, as for example by Algorithm 3.1.

Algorithm 3.1: Triangular rank-one update used in our experiments

input : triangular Cholesky factor $L \in \mathbb{R}^{n\times n}$ of matrix Σ, $\alpha, \beta \in \mathbb{R}$, $\boldsymbol{v} \in \mathbb{R}^n$;
output: updated triangular Cholesky factor L' of $\alpha\Sigma + \beta\boldsymbol{v}\boldsymbol{v}^T$

```
1  ω ← v ;
2  b ← 1;
3  for j = 1, ..., n do
4      l'_jj ← sqrt(α l_jj^2 + (β/b) ω_j^2);
5      γ ← α l_jj^2 b + β ω_j^2;
6      for k = j+1, ..., n do
7          ω_k ← w_k − (ω_j / l_jj) sqrt(α) l_kj;
8          l'_kj = sqrt(α) (l'_jj / l_jj) l_kj + (l'_jj β ω_j / γ) ω_k;
9      end
10     b ← b + β ω_j^2 / (α l_jj^2);
11 end
```

Lemma 2. *Algorithm 3.1 computes* L' *from* L *as defined in Lemma 1.*

Proof. For simplicity, we will again assume $\alpha = 1$. We will denote the value of b in the jth iteration before update as b_j. It is easy to see that the update of ω in line 7 is the same as ω_{j+1} in Corollary 1. Therefore, it remains to show that the updates of b, l'_{kj} and l'_{jj} are the same. For $\frac{\beta}{b_{j+1}}$ it holds by the update rule in line 10

$$\frac{\beta}{b_{j+1}} = \frac{\beta}{b_j + \beta\frac{\omega_j^2}{\alpha l_{jj}^2}} = \frac{\beta}{b_j}\frac{l_{jj}^2}{l_{jj}^2 + \frac{\beta}{b_j}\omega_j^2} = \frac{\beta}{b_j}\frac{l_{jj}^2}{l'^2_{jj}} .$$

Thus, as $b_1 = 1$ it follows by induction that the l'_{jj} have the correct value and $\frac{\beta}{b_{j+1}} = \beta_{j+1}$. This leaves to show that the

update of l'_{kj} is correct. Note that $\gamma = b{l'}_{jj}^2$ and thus

$$\begin{aligned} l'_{kj} &= \frac{l'_{jj}}{l_{jj}} l_{kj} + \frac{\beta\omega_j}{b_j l'_{jj}} \left(w_k - \frac{\omega_j}{l_{jj}} l_{kj} \right) \\ &= \left(\frac{l'_{jj}}{l_{jj}} - \frac{\beta\omega_j^2}{b_j l'_{jj} l_{jj}} \right) l_{kj} + \frac{\beta\omega_j}{b_j l'_{jj}} w_k \\ &= \frac{l_{jj}}{l'_{jj}} \left(\frac{{l'}_{jj}^2 - \frac{\beta}{b_j}\omega_j^2}{l_{jj}^2} \right) l_{kj} + \frac{\beta\omega_j}{b_j l'_{jj}} w_k \\ &= \frac{l_{jj}}{l'_{jj}} l_{kj} + \frac{\beta\omega_j}{b_j l'_{jj}} w_k \ , \end{aligned}$$

which is identical to the update of the off-diagonal elements in Lemma 1. □

Variants of this algorithm are implemented for example in Matlab as `cholUpdate` or in the Eigen C++ library as `LLT::rankUpdate`. The pseudocode in Algorithm 3.1 corresponds to the `choleskyUpdate` routine in the Shark library [9] which is based on the Eigen implementation. All implementations are in a form that can directly be used in methods such as the (1+1)-CMA-ES (see Section 4.1) or the MO-CMA-ES. It is interesting to note that this update does not even require to solve a system of equations $L_k \boldsymbol{w} = \boldsymbol{v}$.

Note that the update rule proposed by Lemma 1 is numerically very unstable for $\beta < 0$. Algorithm 3.1, however, is more stable. To understand this, note that the initially proposed update rule relies on the fraction $\frac{l_{11}}{l'_{11}}$. As we perform the update $l'_{11} = \sqrt{l_{11}^2 + \beta v_1^2}$, in finite precision arithmetic, the computed value of l'_{11}, $\tilde{l}'_{11}$ will differ from the true value, that is, $|\tilde{l}'_{11} - l'_{11}| < \epsilon$ for some $\epsilon > 0$. The error of the computation of $\frac{l_{11}}{l'_{11}}$ is then bounded by

$$\left| \frac{l_{11}}{l'_{11}} - \frac{l_{11}}{\tilde{l}'_{11}} \right| < \frac{l_{11}}{l'_{11}} \left| \frac{\epsilon}{\tilde{l}'_{11}} \right| < \frac{l_{11}}{l'_{11}} \left| \frac{1}{\frac{l'_{11}}{\epsilon} - 1} \right| \ .$$

This error can become large in finite precision as with $l_{11}^2 + \beta v_1^2 \approx 0$ the problem of *catastrophic cancellation* leads to greatly reduced precision when $l_{11} \gg l'_{11}$ and thus ϵ becomes large in relation to l'_{11}. This problem becomes more severe as the updated β' is also computed based on this fraction, which allows the error to accumulate over several iterations leading to significant errors.

Algorithm 3.1 instead relies on $\frac{l'_{11}}{l_{11}}$. In this case, the error is bounded by

$$\left| \frac{l'_{11}}{l_{11}} - \frac{\tilde{l}'_{11}}{l_{11}} \right| < \left| \frac{\epsilon}{l_{11}} \right| \ ,$$

which remains small even in the presence of catastrophic cancellation. In the case of ill-conditioned matrices, the update scheme may lead to round-off errors, still, there exist update schemes that guarantee positive definiteness for $\beta > 0$ [4].

As all operations in Algorithm 3.1 are in $O(n)$ time in every iteration of the outer loop, the whole update is $O(n^2)$, where $3/2n^2 + O(n)$ multiplications need to be computed, which are split in $n^2/2$ multiplications to update ω_k and n^2 multiplications to update L_{kj}. In comparison, the original (1+1)-CMA-ES update [16, 2] takes $2n^2 + O(n)$ multiplications for updating L_{k+1}, where the operations are evenly split between the computation of $\boldsymbol{w}_k$ and the outer product of L_{k+1}. Moreover, the original (1+1)-CMA-ES additionally needs to update the inverse L_{k+1} which costs $2n^2 + O(n)$ multiplications, leading to a total cost of $4n^2 + O(n)$ multiplications, which makes the new update less expensive.

This suggests that one can expect a speed-up of the covariance matrix update by a factor of $4/(3/2) = 8/3 \approx 2.667$. This is only to be expected for small matrices which reside fully in CPU-cache, because for large matrices the costs of reading and writing to memory will dominate. As the new algorithm requires less storage, it is also expected to reduce the running time of the algorithm in the regime of large matrices.

4. EXPERIMENTS & RESULTS

The new update rule is obviously more memory efficient than the original (1+1)-CMA-ES update. To verify that it is also faster in practice – as counting the basic operations suggests – we experimentally compared efficient implementations of both update rules. We considered the (1+1)-CMA-ES as an example. Next, we outline how the new update rule can be applied within the (1+1)-CMA-ES with active covariance matrix update. Then the experiments are described and the results are presented.

4.1 (1+1)-CMA-ES with triangular Cholesky update

For a detailed description of the (1+1)-CMA-ES we refer to [16, 2]. The new (1+1)-CMA-ES using the triangular Cholesky update is given in Algorithm 4.1, the values of the constants are the same as suggested in [2] and are listed in Table 4.1 for completeness.

In the kth iteration, an offspring individual $\boldsymbol{x}_{\text{offspring}}$ is sampled from the current parent position $\boldsymbol{x}_k$ using the current Cholesky factor L_k from the covariance matrix Σ_k. If the newly sampled individual has a better fitness than the old, it is accepted and the expected success probability $\overline{p}_{\text{succ},k}$ is updated accordingly. Based on the current success rate, the step size σ_k is updated in an exponential fashion. The remainder of the algorithm is devoted to the update of the covariance matrix, which has three cases:

- if $\overline{p}_{\text{succ}} \geq p_{\text{thresh}}$, it is assumed that some directions of the covariance matrix are too small, so that offspring are not sampled exploratory enough around the current point. To compensate this, the evolution path $\boldsymbol{p}_{c,k}$ is shrunk and the covariance matrix C_k is updated to get overall a rounder shape.

- Otherwise, if the new offspring was successful, the evolution path is updated to reflect the move in the direction of $\boldsymbol{x}_{k+1} - \boldsymbol{x}_k$ and a rank-one update in the direction of the evolution path is performed.

- In the case that the individual is particularly unsuccessful, an active update takes place, which updates C_k with a negative update weight to make far steps in that direction unlikely.

  We regard an individual to be unsuccessful when it is worse than its 5th order ancestor, that is the x_k that is five *successful* steps in the past, not counting any unsuccessful offspring.

  Care has to be taken to ensure that C_k remains positive definite, which is the case when $1 - \frac{\bar{c}_{\text{cov}}}{1+\bar{c}_{\text{cov}}} ||L_k \boldsymbol{z}_k||^2 > 0$,

Algorithm 4.1: (1+1)-CMA-ES with active covariance matrix update

external parameters: initial $\boldsymbol{x}_0$, σ_0, L_0

$\overline{p}_{\text{succ},0} \leftarrow p_{\text{succ}}^{\text{target}}$, $\boldsymbol{p}_{c,k} \leftarrow \boldsymbol{0}$, $k \leftarrow 0$, $c_L \leftarrow 1 - c_{\text{cov}} + c_{\text{cov}} c_{\text{c}}(2 - c_{\text{c}})$;

repeat

  $\boldsymbol{z}_k \sim \mathcal{N}(\boldsymbol{0}, I)$;

  $\boldsymbol{x}_{\text{offspring}} \leftarrow \boldsymbol{m}_k + \sigma_k L_k \boldsymbol{z}_k$;

  if $f(\boldsymbol{x}_{\text{offspring}}) \leq f(\boldsymbol{m}_k)$ **then**

    $\boldsymbol{m}_{k+1} \leftarrow \boldsymbol{x}_{\text{offspring}}$;

    $\overline{p}_{\text{succ},k+1} \leftarrow (1 - c_p)\,\overline{p}_{\text{succ},k} + c_p$;

  end

  else

    $\overline{p}_{\text{succ},k+1} \leftarrow (1 - c_p)\,\overline{p}_{\text{succ},k}$;

  end

  $\sigma_{k+1} \leftarrow \sigma_k \cdot \exp\left(\frac{1}{d} \frac{\overline{p}_{\text{succ},k+1} - p_{\text{succ}}^{\text{target}}}{1 - p_{\text{succ}}^{\text{target}}}\right)$;

  if $\overline{p}_{\text{succ},k+1} \geq p_{thresh}$ **then**

    $\boldsymbol{p}_{c,k+1} \leftarrow (1 - c_{\text{c}})\boldsymbol{p}_{c,k}$;

    $L_{k+1} \leftarrow \texttt{choleskyUpdate}(L_k, c_L, c_{\text{cov}}, \boldsymbol{p}_{c,k+1})$;

  end

  else if $\boldsymbol{x}_{\text{offspring}}$ *is unsuccessful* **then**

    $c_{\text{cov}}^{-} = \min\{c_{\text{cov}}^{-,\max}, \frac{1}{2||\boldsymbol{z}_k||^2 - 1}\}$

    $L_{k+1} \leftarrow \texttt{choleskyUpdate}(L_k, 1 + c_{\text{cov}}^{-}, -c_{\text{cov}}^{-}, L_k \boldsymbol{z}_k)$;

  end

  $t \leftarrow k + 1$;

until *stopping criterion is met*;

$p_{\text{succ}}^{\text{target}} = \frac{1}{5 + \frac{1}{2}}$	$c_p = \frac{p_{\text{succ}}^{\text{target}}}{2 + p_{\text{succ}}^{\text{target}}}$
$d = 1 + \frac{n}{2}$	$c_{\text{c}} = \frac{2}{2+n}$
$c_{\text{cov}} = \frac{2}{n^2 + 6}$	$c_{\text{cov}}^{-,\max} = \frac{0.4}{n^{1.6}+1}$
$\sigma_0 = \frac{1}{\sqrt{n}}$	$\overline{p}_{\text{succ},0} = p_{\text{succ}}^{\text{target}}$

Table 4.1: Constants used for the (1+1)-CMA-ES

as otherwise γ_k is undefined. To prevent numerical instabilities, we choose $c_{\text{cov}}^{-} = \min\left\{c_{\text{cov}}^{-,\max}, \frac{1}{2||\boldsymbol{z}_k||^2 - 1}\right\}$.

The function `updateCovariance`(L, α,β, $\boldsymbol{v}$) implements the rank-one update algorithm for L to replace an update of the form $\alpha\Sigma + \beta\boldsymbol{v}\boldsymbol{v}^T$. In the experiments presented in the next section, we used the variant described in Algorithm 3.1. Again, note that the inverse of the Cholesky factor does not occur in the algorithm in contrast to the variants presented in [10, 16, 2].

4.2 Experimental setup

First, we conducted an experiment on the run-time of the algorithm itself. We performed 100000 covariance matrix update using the old formula and the new formula, respectively, with varying dimensionality $n \in \{100, 200, 400, 800\}$ and measured the running time of both algorithms.

Second, we conducted experiments on 5 different randomly rotated benchmark functions as well as their unrotated counterpart: Rosenbrock, Cigar, Discus, Ellipsoid and DiffPowers as defined in Table 4.3. For each function, we ran the (1+1)-CMA-ES with active updates [2] for 10000 iterations and 100 trials with both update rules. We measured the total time summed up over all trials. We repeated every experiment using different dimensionalities of the benchmark functions varying $n \in \{100, 200, 400, 800\}$. In the experiments we used the fact that the new algorithm produces triangular matrices to speed up the sampling of the vectors by using a matrix-vector product implementation, which takes the triangular structure into account. However, to keep both algorithms as comparable as possible, we maintained in both cases the full Cholesky factor matrix with n^2 entries.

We used Algorithm 4.1 as briefly described in Section 4.1. and the Cholesky update algorithm given in Algorithm 3.1. Our implementation is part of the open-source machine learning library Shark [9] and the code to run the experiments is available in the suplementary material.

4.3 Results

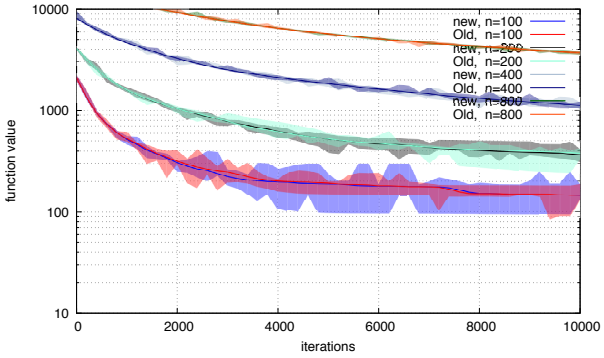
The results for the first experiment are given in Table 4.2. we see that the new algorithm always ran faster than the old one. To measure the speed-up, we calculated the quotient of the running time of the old algorithm and the new one. For $n = 100$ the speed-up is almost exactly the value 8/3 suggested by the analysis of the running time. When n increases, the differences become even larger with a factor of 5.39 with $n = 800$. This is because accessing the RAM takes a lot longer than accessing the CPU-Cache and with growing dimensionality the matrix size exceeds the size of the cache, in which case memory accesses become the bottleneck of the algorithm. As the new algorithm only accesses 1/4th the memory of the old algorithm, this is measureable in running-time.

The results are given in Table 4.4 and the speed-up factors are given in Table 4.5. When comparing the results of the unrotated to the more expensive rotated functions, it can be seen that the impact of the new algorithm becomes less pronounced as the function evaluations become more expensive, as is to be expected. We can see again that the new update rule has a growing impact with increasing n which leads to a cummulated speed-up factor of 2 for the unrotated functions. The impact is not as strong as was suggested by the stand-alone experiment as with growing n less offspring are successful and thus less covariance matrix updates are performed.

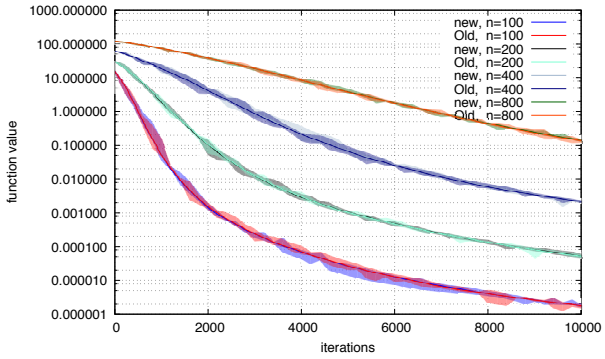
Both algorithms produced similar learning curves, which can be seen from the median and the 25 and 75 percentiles of the function values over the 100 trials as plotted in Figure 4.1.

n	original update	new update	speed-up factor
100	3.11	1.16	2.68
200	12.18	3.88	3.14
400	51.36	13.75	3.74
800	284.19	52.69	5.39

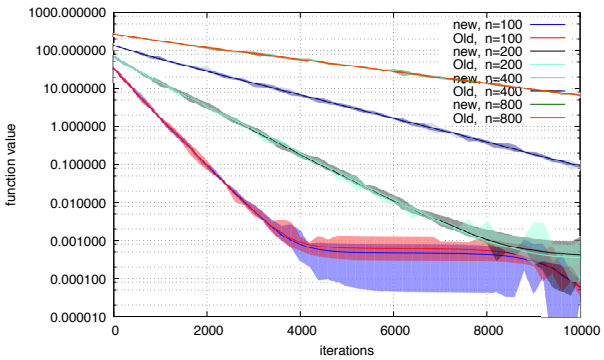
Table 4.2: Results of the timing Experiments of 100000 updates of the cholesky factor L in seconds. Speed-up is calculated as $\text{time}_{\text{orig}}/\text{time}_{\text{new}}$.



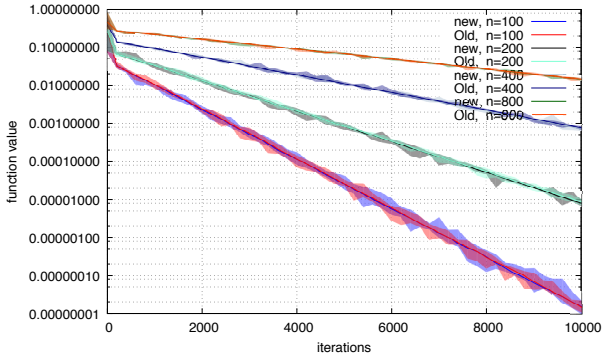
(a) Rosenbrock



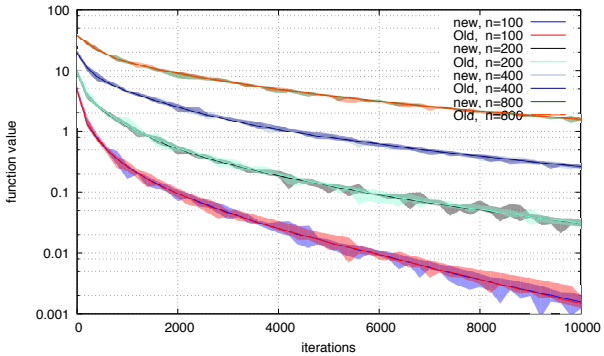
(b) DiffPowers



(c) Cigar



(d) Discus



(e) Ellipsoid

Figure 4.1: Median of the function values for the old and new update rule applied to the five test functions with different dimensionalities n. The shaded areas indicate the 25 and 75 percentiles. Note that the graphs of the medians are largely overlapping

name	formula
Rosenbrock	$f(\boldsymbol{x}) = \sum_{i=1}^{n} 100(x_{i+1} - x_i^2)^2 + (1 - x_i)^2$
Cigar	$f_\alpha(\boldsymbol{x}) = \alpha x_1^2 + \sum_{i=2}^{n} x_i^2$
Discus	$f_\alpha(\boldsymbol{x}) = x_1^2 + \alpha \sum_{i=2}^{n} x_i^2$
Ellipsoid	$f_\alpha(\boldsymbol{x}) = \sum_{i=1}^{n} \alpha^{\frac{i}{n}} x_i^2$
DiffPowers	$f(\boldsymbol{x}) = \sum_{i=1}^{n} \lvert x_i \rvert^{2+10\frac{i-1}{n}}$

Table 4.3: The formulas for the unrotated benchmark functions, where in all experiments $\alpha = 10^{-3}$. The rotated functions have the form $g(\boldsymbol{x}) = f(R\boldsymbol{x})$, where $R \in \mathbb{R}^{n \times n}$ is a random rotation matrix.

5. CONCLUSION

We presented a result on updating Cholesky factors while retaining a triangular shape and adopted it for covariance matrix adaption in randomized search and optimization. When employed within the (1+1)-CMA-ES and the the MO-CMA-ES, it decreases the memory usage by almost a factor of 4 and reduces computation time. The first is a direct result of storing only a triangular matrix instead of two full square matrices, while the latter was confirmed by numerical experiments. The new update is easy to implement and already part of major numeric libraries. Additionally it also allows for less expensive sampling of points, because of the triangular shape of the Cholesky factor. In summary, we see no reason for preferring the original Cholesky factor update rules (e.g., [10, 16, 2]) over the proposed type of methods.

Acknowledgements

We gratefully acknowledge support from the Danish National Advanced Technology Foundation through project "Personalized breast cancer screening".

6. REFERENCES

[1] Y. Akimoto, A. Auger, and N. Hansen. Comparison-based natural gradient optimization in high dimension. In *Proceedings of the 16th Annual Genetic and Evolutionary Computation Conference (GECCO)*, pages 373–380. ACM, 2014.

[2] D. V. Arnold and N. Hansen. Active covariance matrix adaptation for the (1+1)-CMA-ES. In *Proceedings of the 12th Annual Genetic and Evolutionary Computation Conference (GECCO)*, pages 385–392. ACM, 2010.

[3] H.-G. Beyer and H.-P. Schwefel. Evolution strategies–A comprehensive introduction. *Natural Computing*, 1(1):3–52, 2002.

[4] P. E. Gill, G. H. Golub, W. Murray, and M. A. Saunders. Methods for modifying matrix factorizations. *Mathematics of Computation*, 28(126):505–535, 1974.

[5] N. Hansen. Adaptive encoding: How to render search coordinate system invariant. In G. Rudolph et al., editors, *Proceedings of the 10th International Conference on Parallel Problem Solving from Nature (PPSN X)*, LNCS, pages 205–214, 2008.

[6] N. Hansen. The CMA evolution strategy: A tutorial, 2011.

[7] N. Hansen and A. Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, 9(2):159–195, 2001.

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

[9] C. Igel, V. Heidrich-Meisner, and T. Glasmachers. Shark. *Journal of Machine Learning Research*, 9:993–996, 2008.

[10] C. Igel, T. Suttorp, and N. Hansen. A computational efficient covariance matrix update and a (1+1)-CMA for evolution strategies. In *Proceedings of the 8th Annual Genetic and Evolutionary Computation Conference (GECCO)*, pages 453–460. ACM, 2006.

[11] P. Larrañaga. A review on estimation of distribution algorithms. In P. Larrañaga and J. Lozano, editors, *Estimation of distribution algorithms*, pages 57–100. Springer, 2002.

[12] I. Loshchilov. A computationally efficient limited memory CMA-ES for large scale optimization. In *Proceedings of the 16th Annual Genetic and Evolutionary Computation Conference (GECCO)*, pages 397–404. ACM, 2014.

[13] J. Poland and A. Zell. Main vector adaptation: A CMA variant with linear time and space complexity. In *Proceedings of the 10th Annual Genetic and Evolutionary Computation Conference (GECCO)*, pages 1050–1055. Morgan Kaufmann Publishers, 2001.

[14] R. Ros and N. Hansen. A simple modification in CMA-ES achieving linear time and space complexity. In G. Rudolph et al., editors, *Proceedings of the 10th International Conference on Parallel Problem Solving from Nature (PPSN X)*, LNCS, pages 296–305. Springer, 2008.

[15] Y. Sun, T. Schaul, F. Gomez, and J. Schmidhuber. A linear time natural evolution strategy for non-separable functions. In *15th Annual Conference on Genetic and Evolutionary Computation Conference Companion*, pages 61–62. ACM, 2013.

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

[17] T. Voß, H. Hansen, and C. Igel. Improved step size adaptation for the MO-CMA-ES. In *Proceedings of the 12th Annual Genetic and Evolutionary Computation Conference (GECCO)*, pages 487–494. ACM Press, 2010.

	original update					new update				
n	Rosenbrock	Discus	Cigar	Ellipsoid	DiffPowers	Rosenbrock	Discus	Cigar	Ellipsoid	DiffPowers
unrotated										
100	18.91	18.75	19.38	27.06	27.50	16.32	16.43	16.60	25.15	25.31
200	55.70	54.13	51.87	73.45	73.83	42.28	41.56	40.96	59.33	60.21
400	168.4	154.8	151.5	195.2	192.1	119.2	119.3	117.5	154.2	163.2
800	623.1	650.5	650.5	686.4	687.9	290.7	302.5	298.7	352.3	359.3
rotated										
100	21.98	22.44	23.19	31.8	32.87	19.19	19.04	18.99	27.01	27.76
200	70.47	68.49	66.18	87.19	90.5	55.68	55.97	55.21	74.19	75.90
400	213.0	200.0	196.4	232.6	227.9	176.8	188.4	182.3	215.7	227.2
800	1036	1049	1052	1087	1098	712.4	720.1	721.6	776.3	798.8

Table 4.4: Results of the timing experiments in seconds with varying number of dimensionality n. Left: the original update rule, Right: the new update rule. Experiments where run on the rotated as well as the unrotated functions.

n	Rosenbrock	Discus	Cigar	Ellipsoid	DiffPowers
unrotated					
100	1.16	1.14	1.17	1.08	1.09
200	1.32	1.30	1.27	1.28	1.23
400	1.41	1.30	1.29	1.27	1.18
800	2.14	2.15	2.18	1.95	1.91
rotated					
100	1.15	1.18	1.22	1.18	1.18
200	1.27	1.22	1.20	1.18	1.19
400	1.20	1.06	1.08	1.08	1.00
800	1.45	1.46	1.46	1.40	1.37

Table 4.5: Speed-up timing experiments in seconds with varying number of dimensionality n. Speed-up is calculated as $\text{time}_{\text{orig}}/\text{time}_{\text{new}}$ using the results from table 4.4. Experiments where run on the rotated as well as the unrotated functions.