1 Memetic CMA-ES

1.1 Introduction

While a decade ago, genetic algorithms were the ones that allowed new findings in conformational optimization, state-of-the-art algorithms these days are mainly so called 'Basin-Hopping' algorithms. One of the main reasons why Basin-Hopping algorithms are much more successful than classical genetic algorithms is due to the fact that Basin-Hopping utilizes Quasi-Newton methods like BFGS to do local minimization. BFGS and similar algorithms are capable to reach a local minimum magnitudes faster than CMA-ES. The reason why CMA-ES is much slower for this task is simply found in the fact, that it was developed for Black Box optimization. Quasi-Newton algorithms all require derivative information of the fitness function to operate, which is not given in Black Box optimization. While one can approximate gradients, to be able to use those algorithms for black box optimization, the approximation process slows down the speed of those algorithms significantly. For example, if one uses central differences to do this approximation, BFGS will require two times the dimension of the search space additional function evaluations to perform and a certain approximation error will occur, which might slow down convergence even more. But our goal within this paper is to try to achieve competitive results in the area of conformational optimization and not Black Box optimization. Therefore we had to extend CMA-ES to also incorporate the derivative information, which is available for this kind of problems. One way to do this is simply to couple BFGS with CMA. This kind of coupling is known as memetic algorithm within the field of evolutionary computation. Commonly, a local search method is included within the EA in such a way that sampled individuals within the EA are modified by the local search method. This modification of the samples can happen in two ways called the Lamarckian and the Baldwinian approach. Both approaches run the local search for each individual, but they differ in the way that they incorporate the information from the local search back into the EA. Jean-Baptiste Lamarck was a biologist, who nowadays is mainly remembered for his believe in inheritance of acquired characteristics. In the context of the hybridization of an EA, acquired characteristics relate to the new function value, found via local minimization and the inheritance of the information relates to the position of this new minimum. An algorithm implementing Lamarckian Evolution looks like this:

- sample population \((x_1, \ldots, x_n)\)
- minimize each individual of the population with the local search
- replace the fitness value of the original individual by the local minimum (both fitness and position).
The other approach to incorporate the local search is named after James Mark Baldwin, who also became famous for his theories on evolution. In Baldwinian evolution, individuals are not passing on their knowledge to the next generation. The fact that an individual can acquire certain knowledge, given its genetic information, is enough that, under selective pressure, the population will move towards this genetic coding. In the EA context, this relates to a replacement of the fitness of each individual. A Baldwinian algorithm looks like this:

- sample population \((x_1, \ldots, x_n)\)
- minimize each individual of the population with the local search
- replace the fitness value of the original individual by the fitness of the local minimum, but leave the position unchanged.

Usually, the EA runs independent of these steps and therefore these couplings can be seen as transformation of the underlying fitness landscape. This transformation removes the multi modality from the landscape and therefore, the applied EA will perform much better, as can be seen later in this work within the double funnel benchmark, when compare double sphere with double rastrigin.

1.2 Quasi-Newton Methods

In the mid 1950s, W.C. Davidon, a physicist working at Argonne National Laboratory, was using the coordinate descent method to perform a long optimization calculation. At that time computers were not very stable, and to Davidon’s frustration, the computer system would always crash before the calculation was finished. So Davidon decided to find a way of accelerating the iteration. The algorithm he developed, the first quasi-Newton algorithm, turned out to be one of the most revolutionary ideas in nonlinear optimization. It was soon demonstrated by Fletcher and Powell that the new algorithm was much faster and more reliable than the other existing methods, and this dramatic advance transformed nonlinear optimization overnight. During the following twenty years, numerous variants were proposed and hundreds of papers were devoted to their study. An interesting historical irony is that Davidon’s paper was not accepted for publication; it remained as a technical report for more than thirty years until it appeared in the first issue of the SIAM Journal on Optimization in 1991 [9].

Quasi-Newton methods, like steepest descent, require only the gradient of the objective function to be supplied at each iterate. By measuring the changes in gradients, they construct a model of the objective function that is good enough to produce superlinear convergence. The improvement over steepest descent is dramatic, especially on difficult problems. Moreover, since second derivatives are not required, quasi-Newton methods are sometimes more efficient than Newton’s method. Today, optimization software libraries contain a variety of quasi-Newton algorithms for solving unconstrained, constrained, and large-scale optimization problems.

1.2.1 The BFGS Method

The most popular quasi-Newton algorithm is the BFGS method, named for its discoverers Broyden, Fletcher, Goldfarb, and Shanno. In this section we derive this algorithm (and its close relative, the DFP algorithm [1]). We begin the derivation by forming the following quadratic model of the objective function at the current iterate \(x_k\):

\[
m_k(p) = f_k + \nabla f_k^T p + \frac{1}{2} p^T B_k p.
\]

Here \(B_k\) is an \(n \times n\) symmetric positive definite matrix that will be revised or updated at every iteration. Note that the value and gradient of this model at \(p = 0\) match \(f_k\) and \(\nabla f_k\), respectively. The minimizer \(p_k\) of this convex quadratic model, which we can write explicitly as

\[
p_k = -B_k^{-1} \nabla f_k
\]

is used as the search direction, and the new iterate is

\[
x_{k+1} = x_k + a_k p_k,
\]

where the step length \(a_k\) is chosen to satisfy the Wolfe conditions [3]. This iteration is quite similar to the line search Newton method; the key difference is that the approximate Hessian \(B_k\) is used in place of the true
Hessian. Instead of computing $B_k$ afresh at every iteration, Davidon proposed to update it in a simple manner to account for the curvature measured during the most recent step. Suppose that we have generated a new iterate $x_k + 1$ and wish to construct a new quadratic model, of the form

$$m_k+1(p) = f_{k+1} + \nabla f_{k+1}^T p + \frac{1}{2} p^T B_{k+1} p. \tag{4}$$

What requirements should we impose on $B_{k+1}$, based on the knowledge we have gained during the latest step? One reasonable requirement is that the gradient of $m_{k+1}$ should match the gradient of the objective function $f$ at the latest two iterates $x_k$ and $x_{k+1}$. Since $\nabla m_{k+1}(p)$ is precisely $\nabla f_{k+1}$, the second of these conditions is satisfied automatically. The first condition can be written mathematically as

$$\nabla m_{k+1}(-a_k p_k) = \nabla f_{k+1} - a_k B_{k+1} p_k = \nabla f_k \tag{5}$$

By rearranging, we obtain

$$B_{k+1} a_k p_k = \nabla f_{k+1} - \nabla f_k \tag{6}$$

To simplify the notation it is useful to define the vectors

$$s_k = x_{k+1} - x_k, y_k = \nabla f_{k+1} - \nabla f_k, \tag{7}$$

so that (6) becomes

$$B_{k+1} s_k = y_k \tag{8}$$

We refer to this formula as the secant equation.

Given the displacement $s_k$ and the change of gradients $y_k$, the secant equation requires that the symmetric positive definite matrix $B_{k+1}$ map $s_k$ into $y_k$. This will be possible only if $s_k$ and $y_k$ satisfy the curvature condition

$$s_k^T y_k > 0, \tag{9}$$

as is easily seen by premultiplying (8) by $s_k^T$. When $f$ is strongly convex, the inequality (9) will be satisfied for any two points $x_k$ and $x_{k+1}$ (see the exercises). However, this condition will not always hold for nonconvex functions, and in this case we need to enforce (9) explicitly, by imposing restrictions on the line search procedure that chooses $a$. In fact, the condition (9) is guaranteed to hold if we impose the Wolfe [3] or strong Wolfe conditions on the line search. To verify this claim, we note from (7) that $\nabla f_{k+1}^T s_k > c_2 \nabla f_k^T s_k$, and therefore

$$y_k^T s_k > (c_2 - 1) a_k \nabla f_k^T p_k. \tag{10}$$

Since $c_2 < 1$ and since $p_k$ is a descent direction, the term on the right will be positive, and the curvature condition (9) holds.

When the curvature condition is satisfied, the secant equation (8) always has a solution $B_{k+1}$. In fact, it admits an infinite number of solutions, since there are $n(n + 1)/2$ degrees of freedom in a symmetric matrix, and the secant equation represents only $n$ conditions. The requirement of positive definiteness imposes $n$ additional inequalities all principal minors must be positive but these conditions do not absorb the remaining degrees of freedom.

To determine $B_{k+1}$ uniquely, then, we impose the additional condition that among all symmetric matrices satisfying the secant equation, $B_{k+1}$ is, in some sense, closest to the current matrix $B_k$. In other words, we solve the problem

$$\min_B \|B - B_k\| \tag{11}$$

subject to $B = B^T, B s_k = y_k. \tag{12}$

where $s_k$ and $y_k$ satisfy (9) and $B_k$ is symmetric and positive definite. Many matrix norms can be used in (11), and each norm gives rise to a different quasi-Newton method. A norm that allows easy solution of the minimization problem (11), and that gives rise to a scale-invariant optimization method, is the weighted Frobenius norm
\[ \|A\|_W \equiv \left\| W^{1/2} A W^{1/2} \right\|_F, \]  

(13)

where \( \| \cdot \|_F \) is defined by \( \|C\|_F^2 = \sum_{i=1}^n \sum_{j=1}^n c_{ij}^2 \). The weight \( W \) can be chosen as any matrix satisfying the relation \( W y_k = s_k \). For concreteness, the reader can assume that \( W = \tilde{G}_k^{-1} \) where \( \tilde{G}_k \) is the average Hessian defined by

\[ \tilde{G}_k = \left[ \int_0^1 \nabla^2 f(x_k + \tau a_k p_k) d\tau \right]. \]  

(14)

The property

\[ y_k = \tilde{G}_k a_k p_k = \tilde{G}_k s_k \]  

(15)

follows from Taylor’s theorem. With this choice of weighting matrix \( W \), the norm (13) is adimensional, which is a desirable property, since we do not wish the solution of (11) to depend on the units of the problem.

With this weighting matrix and this norm, the unique solution of (11) is

\[ (DFP) \quad B_{k+1} = (I - \gamma_k y_k s_k^T) B_k (I - \gamma_k s_k y_k^T) + \gamma_k y_k y_k^T, \]  

(16)

with

\[ \gamma_k = \frac{1}{y_k^T s_k}. \]  

(17)

This formula is called the DFP updating formula, since it is the one originally proposed by Davidon in 1959 [1], and subsequently studied, implemented, and popularized by Fletcher and Powell [1].

The inverse of \( B_k \), which we denote by

\[ H_k = B_k^{-1}, \]  

(18)

is useful in the implementation of the method, since it allows the search direction (2) to be calculated by means of a simple matrix vector multiplication. Using the Sherman-Morrison-Woodbury formula [2], we can derive the following expression for the update of the inverse Hessian approximation \( H_k \) that corresponds to the DFP update of \( B_k \) in (23):

\[ (DFP) \quad H_{k+1} = H_k - \frac{H_k y_k y_k^T H_k}{y_k^T H_k y_k} + \frac{s_k s_k^T}{y_k^T s_k}. \]  

(19)

Note that the last two terms in the right-hand-side of (19) are rank-one matrices, so that \( H_k \) undergoes a rank-two modification. It is easy to see that (16) is also a rank-two modification of \( B_k \). This is the fundamental idea of quasi-Newton updating: Instead of recomputing the iteration matrices from scratch at every iteration, we apply a simple modification that combines the most recently observed information about the objective function with the existing knowledge embedded in our current Hessian approximation.

The DFP updating formula is quite effective, but it was soon superseded by the BFGS formula, which is presently considered to be the most effective of all quasi-Newton updating formulae. BFGS updating can be derived by making a simple change in the argument that led to (16). Instead of imposing conditions on the Hessian approximations \( B_k \), we impose similar conditions on their inverses \( H_k \). The updated approximation \( H_{k+1} \) must be symmetric and positive definite, and must satisfy the secant equation (8), now written as

\[ H_{k+1} y_k = s_k. \]  

(20)

The condition of closeness to \( H_k \) is now specified by the following analogue of (8.9):

\[ \min_H \| H - H_k \| \]  

subject to \( H = H^T, H y_k = s_k \).  

(21)

(22)

The norm is again the weighted Frobenius norm described above, where the weight matrix \( W \) is now any matrix satisfying \( W s_k = y_k \). (For concreteness, we assume again that \( W \) is given by the average Hessian \( \tilde{G}_k \) defined in (14).) The unique solution \( H_{k+1} \) to (21) is given by

4
(BFGS) \[ H_{k+1} = (I - \rho_k s_k y_k^T)H_k (I - \rho_k y_k s_k^T) + \rho_k s_k s_k^T, \] (23)

where

\[ \rho_k = \frac{1}{y_k^T s_k} \] (24)

Just one issue has to be resolved before we can define a complete BFGS algorithm: How should we choose the initial approximation \( H_0 \)? Unfortunately, there is no magic formula that works well in all cases. We can use specific information about the problem, for instance by setting it to the inverse of an approximate Hessian calculated by finite differences at \( x_0 \). Otherwise, we can simply set it to be the identity matrix, or a multiple of the identity matrix, where the multiple is chosen to reflect the scaling of the variables.

Algorithm 1 BFGS Method

Given starting point \( x_0 \), convergence tolerance \( \epsilon > 0 \),
inverse Hessian approximation \( H_0 \);
\[ k \leftarrow 0; \]
while \( \|\nabla f_k\| > \epsilon \) do

Compute search direction
\[ p_k = -H_k \nabla f_k \]
Set \( x_{k+1} = x_k + a_k p_k \) where \( a_k \) is computed from a line search procedure to satisfy the Wolf conditions;
Define \( s_k = x_{k+1} - x_k \) and \( y_k = \nabla f_{k+1} - \nabla f_k \);
Compute \( H_{k+1} \) by means of (23);
\[ k \leftarrow k + 1 \]
end while

Each iteration can be performed at a cost of \( O(n^2) \) arithmetic operations (plus the cost of function and gradient evaluations); there are no \( O(n^3) \) operations such as linear system solves or matrix-matrix operations. The algorithm is robust, and its rate of convergence is superlinear, which is fast enough for most practical purposes. Even though Newton’s method converges more rapidly (that is, quadratically), its cost per iteration is higher because it requires the solution of a linear system. A more important advantage for BFGS is, of course, that it does not require calculation of second derivatives.

We can derive a version of the BFGS algorithm that works with the Hessian approximation \( B_k \) rather than \( H_k \). The update formula for \( B_k \) is obtained by simply applying the Sherman-Morrison-Woodbury formula to (23) to obtain

\[ (BFGS) \quad B_{k+1} = B_k - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k} + \frac{y_k y_k^T}{y_k^T s_k}. \] (25)

A naïve implementation of this variant is not efficient for unconstrained minimization, because it requires the system \( B_k p_k = -\nabla f_k \) to be solved for the step \( p_k \), thereby increasing the cost of the step computation to \( O(n^3) \). However, that less expensive implementations of this variant are possible by updating Cholesky factors of \( B_k \).

1.3 Limited memory quasi-Newton methods

Instead of the BFGS algorithm introduced above, we use a limited memory quasi-Netwon algorithm for solving large nonlinear optimization problems with simple bounds on the variables. We write this problem as

\[ \min f(x) \] (26)

subject to \( l \leq x \leq u \), (27)

where \( f : \mathbb{R}^n \to \mathbb{R} \) is a nonlinear function whose gradient \( g \) is available, the vectors \( l \) and \( u \) represent lower and upper bounds on the variables, and the number of variables \( n \) is assumed to be large. Not all the variables need to have bounds; in fact the algorithm is also appropriate and efficient for solving unconstrained problems.
The algorithm does not require second derivatives or knowledge of the structure of the objective function, and can therefore be applied when the Hessian matrix is not practical to compute. The algorithm is described in detail in [17] and proceeds roughly as follows. At each iteration a limited memory BFGS approximation to the Hessian is updated. This limited memory matrix is used to define a quadratic model of the objective function \( f \). A search direction is then computed using a two-stage approach: first, the gradient projection method [11, 5, 15, 4] is used to identify a set of active variables, i.e. variables that will be held at their bounds; then the quadratic model is approximately minimized with respect to the free variables. The search direction is defined to be the vector leading from the current iterate to this approximate minimizer. Finally a line search is performed along the search direction using the subroutine described in [14]. A novel feature of the algorithm is that the limited memory BFGS matrices are represented in a compact form [16] that is efficient for bound constrained problems.

The user can control the amount of storage required by L-BFGS-B by selecting a parameter \( m \) that determines the number of BFGS corrections saved. The algorithm requires roughly \((12 + 2m)n\) storage locations, and since small values of \( m \) (say \( 3 \leq m \leq 20 \)) are recommended, it can be used to solve very large problems. The computational cost of one iteration of the algorithm is modest, ranging from \( 4mn + n \) multiplications when no bounds are active, to approximately \( mn^2 \) multiplications when all variables are at their bounds.

If no bounds are active at the solution, it is appropriate to stop the iteration when the norm of the gradient \( g \) is sufficiently small. The corresponding quantity for the case when some bounds are active is the norm of the projected gradient, which we denote by \( ||\text{proj } g|| \), and which is defined, for example, in [17]. Both the output of L-BFGS-B and its documentation, make reference to the projected gradient.

L-BFGS-B is an extension of the limited memory algorithm (L-BFGS) for unconstrained optimization described in [12] and implemented as Harwell routine VA15. The main improvement is the ability of L-BFGS-B to deal with bounds on the variables. Even though this requirement makes the new algorithm far more complex than its predecessor, the two codes perform similarly on unconstrained problems. Therefore L-BFGS-B could be considered to supersede L-BFGS - except for one fact that can be important in some applications: L-BFGS-B requires 8 more n-vectors of storage.

L-BFGS-B is, at present, the only limited memory quasi-Newton algorithm capable of handling bounds on the variables; other published codes [7, 6, 10, 12] are only able to solve unconstrained problems. We note also that the nonlinear conjugate gradient method, which is used for solving many large unconstrained problems, has not been adequately extended to handle bounds on the variables, and L-BFGS-B can be used in its place.

The advantages of L-BFGS-B are: (i) the code is easy to use, and the user need not supply information about the Hessian matrix or about the structure of the objective function; (ii) the storage requirements of the algorithm are modest and can be controlled by the user; (iii) the cost of the iteration is low, and is independent of the properties of the objective function. Due to this, L-BFGS-B is recommended for solving large problems in which the Hessian matrix is not sparse or is difficult to compute.

However L-BFGS-B suffers from the following drawbacks: (i) it is not rapidly convergent, and on difficult problems can take a large number of function evaluations to converge; (ii) on highly ill-conditioned problems the algorithm may fail to obtain high accuracy in the solution; (iii) the algorithm cannot make use of knowledge about the structure of the problem to accelerate convergence.

2 Integration into CMA-ES

The user of the pCMAlib [8] has to set the BFGS use flag to true and optionally can also choose which hybridization approach he would like to use, with Baldwinian evolution being the default choice. For CMA-ES this modification of the sampled points stays invisible and therefore nothing has to be changed within the CMA-ES code. For the interpretation of the results, one has to be careful if Lamarkian evolution is chosen, because the reported global best \( x \) is only capable to reach the global best \( f \) via local search, but would not yield this fitness value itself when evaluated.

Within the scope of this work, we improved the existing implementation of the local BFGS optimizer by implementing the L-BFGS-B limited-memory quasi-Newton code version 2.1 by Ciyou Zhu, Richard Byrd and Jorge Nocedal [18] in the pCMAlib [8]. This implementation reveals several advantages, as seen above, compared to the Liu & Nocedal [13] code of 1989, and could reduce the amount of BFGS evaluation dramatically.
2.1 Reducing BFGS evaluations

Further, we were interested in reducing the BFGS evaluations in the hybrid CMA-ES scheme. This is due to the fact, that our first preliminary results with the CMA-BFGS showed some promising results, outperforming CMA-ES in terms of Lennard Jones Cluster sizes each algorithm was able to solve. It could also be seen that this first prototype of the CMA-BFGS hybrid is not at all competitive with state of the art optimizers for the Lennard Jones cluster problem. The prototype used magnitudes more local optimizations to find any global optimum. The amount of function evaluations used by the prototype where hugely dominated by the local optimization, while on the other hand it seemed like only little information gained through a local optimization would be incorporated back into the CMA part. Therefore we decided to limit the local optimizer. The justification to limit the local optimizer is twofold. On one hand, the local BFGS minimization would run off to local minima which are way out of the covariance of our sampling distribution and hence perturb the global CMA-ES strategy.

On the other hand, BFGS is iterating until $\|proj g\| < pgtol$ where pgtol is controlled by the user but accounts sometimes for unnecessary evaluations when no real progress is made, i.e. the local minimizer is not rapidly convergent and taking a large number of function evaluations to converge. Thus, the local scheme does not contribute enough to the global solution and hence should be limited to a minimum progress in order to avoid in vain iterations.

![Figure 2: An example of a local BFGS minimization leaving the 99% error ellipsoid on the Rastrigin landscape. The black crosses are samples drawn by CMA. The red stars which are connected by the blue lines show the path taken by the local BFGS minimization.](image)
2.1.1 Covariance matrices and error ellipsoid

Let $X$ be a $n$-dimensional Gaussian random vector, with

$$X \sim \mathcal{N}(m_X, \Sigma_X) \tag{28}$$

and consider a constant, $K_1 \in R$. The locus for which the pdf $f(x)$ is greater or equal a specified constant $K_1$, i.e.,

$$\{x : \frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}}exp \left[ -\frac{1}{2} (x - m_x)^T \Sigma^{-1}_X (x - m_X) \right] \geq K_1 \} \tag{29}$$

which is equivalent to

$$\{x : (x - m_X)^T \Sigma^{-1}_X (x - m_X) \geq K \} \tag{30}$$

with $K = -2\ln((2\pi)^{n/2}K |\Sigma|^{1/2})$ is an $n$-dimensional ellipsoid centered at the mean $m_X$ and whose axis are only aligned with the cartesian frame if the covariance matrix $\Sigma$ is diagonal. The ellipsoid defined by 30 is the region of minimum volume that contains a given probability mass under the Gaussian assumption. When in 30 rather than having an inequality there is an equality, 30, i.e.,

$$\{x : (x - m_X)^T \Sigma^{-1}_X (x - m_X) = K \} \tag{31}$$

this locus may be interpreted as the contours of equal probability.

**Definition 1. Mahalanobis distance** The scalar quantity

$$[x - m_X]^T \Sigma^{-1}_X [x - m_X] = K \tag{32}$$

is known as the Mahalanobis distance of the vector $x$ to the mean $m_X$.

The Mahalanobis distance, is a normalized distance where normalization is achieved through the covariance matrix. The surfaces on which $K$ is constant are ellipsoids that are centered about the mean $m_X$, and whose semi-axis are $\sqrt{K}$ times the eigenvalues of $\Sigma_X$. In the special case where the random variables that are the components of $X$ are uncorrelated and with the same variance, i.e., the covariance matrix $\Sigma$ is a diagonal matrix with all its diagonal elements equal, these surfaces are spheres, and the Mahalanobis distance becomes equivalent to the Euclidean distance.

![Figure 3: Contours of equal Mahalanobis and Euclidean distance around $(m_X, m_Y)$ for a second order Gaussian random vector](image-url)
Figure 3 represents the contours of equal Mahalanobis and Euclidean distance around \((m_X, m_Y)\) for a second order Gaussian random vector. In other words, any point \((x, y)\) in the ellipse is at the same Mahalanobis distance to the center of the ellipses. Also, any point \((x, y)\) in the circumference is at the same Euclidean distance to the center. This plot enhances the fact that the Mahalanobis distance is weighted by the covariance matrix.

For decision making purposes (e.g., the field-of-view, a validation gate and the limitation of the BFGS optimizer in our scheme as we shall see later on), and given \(m_X\) and \(\Sigma_X\), it is necessary to determine the probability that a given vector will lie within, say, the 90\% confidence ellipse or ellipsoid given by 3. For a given \(K\), the relationship between \(K\) and the probability of lying within the ellipsoid specified by \(K\) is, [3],

\[
n = 1; \Pr\{x \text{ inside the ellipsoid}\} = -\frac{1}{\sqrt{2\pi}} + 2erf(\sqrt{K})
\]

\[
n = 2; \Pr\{x \text{ inside the ellipsoid}\} = 1 - e^{-K/2}
\]

\[
n = 3; \Pr\{x \text{ inside the ellipsoid}\} = -\frac{1}{\sqrt{2\pi}} + 2erf(\sqrt{K}) - \sqrt{\frac{2}{\pi}}\sqrt{K}e^{-K/2}
\]

where \(n\) is the dimension of the random vector. Numeric values of the above expression for \(n = 2\) are presented in the following table

<table>
<thead>
<tr>
<th>Probability</th>
<th>K</th>
</tr>
</thead>
<tbody>
<tr>
<td>50%</td>
<td>1.386</td>
</tr>
<tr>
<td>60%</td>
<td>1.832</td>
</tr>
<tr>
<td>70%</td>
<td>2.408</td>
</tr>
<tr>
<td>80%</td>
<td>3.219</td>
</tr>
<tr>
<td>90%</td>
<td>4.605</td>
</tr>
</tbody>
</table>

For a given \(K\) the ellipsoid axis are fixed. The probability that a given value of the random vector \(X\) lies within the ellipsoid centered in the mean value, increases with the increase of \(K\). This problem can be stated the other way around. In the case where we specify a fixed probability value, the question is the value of \(K\) that yields an ellipsoid satisfying that probability. To answer the question the statistics of \(K\) has to be analyzed. The scalar random variable 32 has a known random distribution, as stated in the following result.

**Result 1.** Given the \(n\)-dimensional Gaussian random vector \(X\), with mean \(m_X\) and covariance matrix \(\Sigma_X\), the scalar random variable \(K\) defined by the quadratic form

\[
[x - m_X]^T\Sigma^{-1}_X[x - m_X] = K
\]

(34)

has a chi-square distribution with \(n\) degrees of freedom.

The pdf of \(K\) in 34, i.e., the chi-square density with \(n\) degrees of freedom is, (see, p.e., [1])

\[
f(k) = \frac{1}{2^{n/2}\Gamma(n/2)}k^{n/2-1/2}exp\left(-\frac{k}{2}\right)
\]

where the gamma function satisfies,

\[
\Gamma\left(\frac{1}{2}\right) = \sqrt{\pi}, \Gamma(1) = 1 \quad \Gamma(n + 1) = \Gamma(n).
\]

The probability that the scalar random variable, \(K\) in 34 is less or equal a given constant, \(\chi^2_p\)

\[
Pr\{K \leq \chi^2_p\} = Pr\{[x - m_X]^T\Sigma^{-1}_X[x - m_X] \leq \chi^2_p\} = p
\]

is given in the following table where \(n\) is the number of degrees of freedom and the sub-indice \(p\) in \(\chi^2_p\) represents the corresponding probability under evaluation.
<table>
<thead>
<tr>
<th>n</th>
<th>$\chi^2_{0.995}$</th>
<th>$\chi^2_{0.99}$</th>
<th>$\chi^2_{0.975}$</th>
<th>$\chi^2_{0.95}$</th>
<th>$\chi^2_{0.90}$</th>
<th>$\chi^2_{0.75}$</th>
<th>$\chi^2_{0.5}$</th>
<th>$\chi^2_{0.25}$</th>
<th>$\chi^2_{0.10}$</th>
<th>$\chi^2_{0.05}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>7.88</td>
<td>6.63</td>
<td>5.02</td>
<td>3.84</td>
<td>2.71</td>
<td>1.32</td>
<td>0.455</td>
<td>0.102</td>
<td>0.0158</td>
<td>0.0039</td>
</tr>
<tr>
<td>2</td>
<td>10.6</td>
<td>9.21</td>
<td>7.38</td>
<td>5.99</td>
<td>4.61</td>
<td>2.77</td>
<td>1.39</td>
<td>0.575</td>
<td>0.211</td>
<td>0.101</td>
</tr>
<tr>
<td>3</td>
<td>12.8</td>
<td>11.3</td>
<td>9.35</td>
<td>7.81</td>
<td>6.25</td>
<td>4.11</td>
<td>2.37</td>
<td>1.21</td>
<td>0.584</td>
<td>0.352</td>
</tr>
<tr>
<td>4</td>
<td>14.9</td>
<td>13.3</td>
<td>11.1</td>
<td>9.49</td>
<td>7.78</td>
<td>5.39</td>
<td>3.36</td>
<td>1.92</td>
<td>1.06</td>
<td>0.711</td>
</tr>
</tbody>
</table>

From this table we can conclude, for example, that for a third-order Gaussian random vector, $n = 3$,

$$Pr\{K \leq 6.25\} = Pr\{[x - m_X]^T \Sigma^{-1}[x - m_X] \leq 6.25\} = 0.9$$

### 2.1.2 Covariance restriction on BFGS

In the CMA-ES/BFGS scheme, we want to limit the quasi-Newton algorithm to its boundaries of the local optimizer and hence restrict the solution space BFGS is allowed to travel. The population of the CMA-ES can be seen as $n$-dimensional Gaussian random vector with $X \sim \mathcal{N}(m_X, A)$, where $A = \sigma^2 C$ is the corrected Covariance Matrix of the evolutionary algorithm and $m_X$ the mean of the population. Thus, the error ellipsoid for the given population can be calculated according to

$$\{[x - m_X]^T A^{-1}[x - m_X] \geq K\} \quad (35)$$

Since $K$ has a chi-square distribution with $n$ degrees of freedom we can deduce the value $K$ by inverse chi$^2$ calculations for a given probability $P$. For our calculations we arbitrary chose $P = 95\%$ and calculated $K$ according to the problem size, dimension $n$ respectively. The restriction can also been seen as an upper limit of the Mahalanobis distance the BFGS is allowed to move in the search space. Therefore, for each iteration-step the local minimizer executes, one has to check if the abort criteria is met. According to

$$\{[x_{\text{current}} - m_X]^T A^{-1}[x_{\text{current}} - m_X] \geq K\}$$

where $K(n, P)$ depends on the dimension size $n$ and probability $P$ and $x_{\text{current}}$ is the current solution. One calculates the Mahalanobis distance and exits the BFGS algorithm if $\text{distance}_{\text{Mahalanobis}} \geq K$. For the entire CMA-ES/BFGS algorithm, $K$ is only calculated once by the subroutine $\text{CHSPPF}$ provided by James Filiben.

### Algorithm 2 BFGS with error ellipsoid exit

Given starting point $x_0$, convergence tolerance $\epsilon > 0$,
inverse Hessian approximation $H_0$; $k \leftarrow 0$;

**while** $||\nabla f_k|| > \epsilon$ **do**

  Compute search direction
  $p_k = -H_k \nabla f_k$

  Set $x_{k+1} = x_k + a_k p_k$ where $a_k$ is computed from a line search procedure to satisfy the Wolf conditions;
  if $[x_k - m_X]^T A^{-1} [x_k - m_X] \geq K$ **then**

    **exit**

  **end if**

  Define $s_k = x_{k+1} - x_k$ and $y_k = \nabla f_{k+1} - \nabla f_k$;

  Compute $H_{k+1}$ by means of (23);

  $k \leftarrow k + 1$

**end while**
3 Results

To compare the performance of the limited and unlimited hybrid we compared multiple batches of optimization, each batch with different settings and each batch optimized with both versions of the algorithm. Unfortunately no significant differences between the two versions of the algorithm could be seen. We show the setup and results of one of those comparison pairs in the following sections.

3.0.3 Experimental Setup

For this particular case, a setup which is limited on the time each optimization can run was selected, with a limit of thirty minutes runtime each. A Lennard Jones cluster with 38 atoms was used as the underlying problem. With exception of those parameters listed below, standard CMA settings were used.

```matlab
myopts.StopTimeUse = 'true';
myopts.StopTimeMM = '30';
myopts.dimensions = '108';
myopts.use_init_bounds = 'true';
myopts.init_LBounds = '3';
myopts.init_uBounds = '-3';
myopts.StopFitness = '-173.928';
myopts.use_LJC = 'true';
myopts.abs_sigma = '0.5';
myopts.MaxIncFac = '1';
myopts.RecombinationWeights = '3';
myopts.IncPopSize = '1';
myopts.BFGS_fctr = '1.0';
myopts.accuracy = '0.001';
myopts.restart_cma = 'true';
myopts.BFGS_position = '2';
myopts.BFGS_pgtol = '0.0001';
myopts.restart_type = '0';
```

3.0.4 Results without Covariance restriction

<table>
<thead>
<tr>
<th>bestever_f</th>
<th>countEval</th>
<th>bestever_evals</th>
<th>bestever_f</th>
<th>countEval</th>
<th>bestever_evals</th>
</tr>
</thead>
<tbody>
<tr>
<td>-170.254</td>
<td>4669454</td>
<td>2587375</td>
<td>-169.8038</td>
<td>4832012</td>
<td>1878875</td>
</tr>
<tr>
<td>-169.6297</td>
<td>4551194</td>
<td>1754963</td>
<td>-169.58</td>
<td>4873520</td>
<td>1004061</td>
</tr>
<tr>
<td>-169.3007</td>
<td>4017944</td>
<td>1978061</td>
<td>-169.038</td>
<td>4018142</td>
<td>1962101</td>
</tr>
<tr>
<td>-168.8996</td>
<td>4821032</td>
<td>425485</td>
<td>-168.6786</td>
<td>4018142</td>
<td>1962101</td>
</tr>
<tr>
<td>-168.2801</td>
<td>5114504</td>
<td>4978980</td>
<td>-168.2788</td>
<td>4064780</td>
<td>4064780</td>
</tr>
<tr>
<td>-168.1825</td>
<td>4024082</td>
<td>4024041</td>
<td>-168.1449</td>
<td>4460168</td>
<td>1580507</td>
</tr>
<tr>
<td>-168.0146</td>
<td>3911708</td>
<td>3661838</td>
<td>-167.9735</td>
<td>3994040</td>
<td>3994010</td>
</tr>
<tr>
<td>-167.97</td>
<td>4047428</td>
<td>2053577</td>
<td>-167.9282</td>
<td>5303090</td>
<td>5044778</td>
</tr>
<tr>
<td>-167.9262</td>
<td>4669598</td>
<td>3657309</td>
<td>-167.8183</td>
<td>4068074</td>
<td>933233</td>
</tr>
<tr>
<td>-167.6043</td>
<td>3814778</td>
<td>2508544</td>
<td>-167.5673</td>
<td>4047176</td>
<td>1230840</td>
</tr>
<tr>
<td>-167.561</td>
<td>5092616</td>
<td>2026307</td>
<td>-167.4858</td>
<td>4018772</td>
<td>1515615</td>
</tr>
<tr>
<td>-167.3695</td>
<td>4705220</td>
<td>1240518</td>
<td>-167.1955</td>
<td>3941048</td>
<td>2213152</td>
</tr>
<tr>
<td>-167.1326</td>
<td>4035872</td>
<td>2163707</td>
<td>-167.1185</td>
<td>4730906</td>
<td>1447271</td>
</tr>
<tr>
<td>-167.1127</td>
<td>4027952</td>
<td>932950</td>
<td>-167.0678</td>
<td>4904300</td>
<td>1568755</td>
</tr>
<tr>
<td>-166.8755</td>
<td>4885040</td>
<td>523148</td>
<td>-166.7341</td>
<td>4635812</td>
<td>3629416</td>
</tr>
<tr>
<td>-166.7299</td>
<td>5214044</td>
<td>2101093</td>
<td>-166.6623</td>
<td>4633706</td>
<td>2036731</td>
</tr>
</tbody>
</table>
Table 1: Results without Covariance restriction (continued)

<table>
<thead>
<tr>
<th>bestever_f</th>
<th>countEval</th>
<th>bestever_evals</th>
<th>bestever_f</th>
<th>countEval</th>
<th>bestever_evals</th>
</tr>
</thead>
<tbody>
<tr>
<td>-166.6119</td>
<td>4703780</td>
<td>1964528</td>
<td>-166.5009</td>
<td>4034180</td>
<td>2676851</td>
</tr>
<tr>
<td>-166.4783</td>
<td>4844322</td>
<td>2360893</td>
<td>-166.1692</td>
<td>4807028</td>
<td>3075652</td>
</tr>
<tr>
<td>-166.1604</td>
<td>4010186</td>
<td>3497272</td>
<td>-166.081</td>
<td>4765688</td>
<td>2169551</td>
</tr>
<tr>
<td>-166.0708</td>
<td>4880738</td>
<td>1043992</td>
<td>-166.0156</td>
<td>4716380</td>
<td>3078773</td>
</tr>
<tr>
<td>-165.9381</td>
<td>3956204</td>
<td>2785340</td>
<td>-165.7777</td>
<td>4867112</td>
<td>1771231</td>
</tr>
<tr>
<td>-165.5308</td>
<td>3999296</td>
<td>2421558</td>
<td>-165.4804</td>
<td>4010186</td>
<td>3497272</td>
</tr>
<tr>
<td>-165.4544</td>
<td>4820330</td>
<td>2404813</td>
<td>-165.2993</td>
<td>3993968</td>
<td>834126</td>
</tr>
<tr>
<td>-165.2034</td>
<td>4795706</td>
<td>3831575</td>
<td>-164.7975</td>
<td>4740734</td>
<td>562758</td>
</tr>
<tr>
<td>-164.7822</td>
<td>4002554</td>
<td>4002541</td>
<td>-164.6023</td>
<td>3947690</td>
<td>3413402</td>
</tr>
</tbody>
</table>

3.0.5 Results with Covariance restriction

<table>
<thead>
<tr>
<th>bestever_f</th>
<th>countEval</th>
<th>bestever_evals</th>
<th>bestever_f</th>
<th>countEval</th>
<th>bestever_evals</th>
</tr>
</thead>
<tbody>
<tr>
<td>-169.3177</td>
<td>3944774</td>
<td>3293420</td>
<td>-169.1939</td>
<td>3994166</td>
<td>1596315</td>
</tr>
<tr>
<td>-169.1186</td>
<td>3838996</td>
<td>2611279</td>
<td>-168.5646</td>
<td>3987794</td>
<td>3890328</td>
</tr>
<tr>
<td>-168.8362</td>
<td>4006406</td>
<td>1249624</td>
<td>-168.3928</td>
<td>4017644</td>
<td>4030224</td>
</tr>
<tr>
<td>-168.5126</td>
<td>4028924</td>
<td>1730633</td>
<td>-168.2503</td>
<td>4009484</td>
<td>342430</td>
</tr>
<tr>
<td>-168.376</td>
<td>4013246</td>
<td>3931913</td>
<td>-167.9204</td>
<td>3836576</td>
<td>3380499</td>
</tr>
<tr>
<td>-167.9548</td>
<td>4044314</td>
<td>407422</td>
<td>-167.7983</td>
<td>403378</td>
<td>404325</td>
</tr>
<tr>
<td>-167.8132</td>
<td>4040696</td>
<td>972109</td>
<td>-167.7425</td>
<td>4031318</td>
<td>821475</td>
</tr>
<tr>
<td>-167.7693</td>
<td>4031606</td>
<td>3868734</td>
<td>-167.5536</td>
<td>4015388</td>
<td>2189215</td>
</tr>
<tr>
<td>-167.6625</td>
<td>4034666</td>
<td>1591187</td>
<td>-167.4242</td>
<td>3915488</td>
<td>384254</td>
</tr>
<tr>
<td>-167.4273</td>
<td>4030814</td>
<td>1679561</td>
<td>-167.3478</td>
<td>3899648</td>
<td>3794530</td>
</tr>
<tr>
<td>-167.3783</td>
<td>4020320</td>
<td>2994101</td>
<td>-167.2641</td>
<td>3855926</td>
<td>2503615</td>
</tr>
<tr>
<td>-166.9971</td>
<td>3963692</td>
<td>873825</td>
<td>-166.9654</td>
<td>4048346</td>
<td>3183809</td>
</tr>
<tr>
<td>-166.8679</td>
<td>3946808</td>
<td>2944949</td>
<td>-166.8456</td>
<td>405142</td>
<td>320474</td>
</tr>
<tr>
<td>-166.7224</td>
<td>4028582</td>
<td>2475545</td>
<td>-166.7032</td>
<td>3866364</td>
<td>2626291</td>
</tr>
<tr>
<td>-166.5057</td>
<td>3876518</td>
<td>714101</td>
<td>-166.65</td>
<td>4031210</td>
<td>2659435</td>
</tr>
<tr>
<td>-166.5</td>
<td>4027232</td>
<td>2659435</td>
<td>-166.4067</td>
<td>3972242</td>
<td>1472964</td>
</tr>
<tr>
<td>-166.3521</td>
<td>3907676</td>
<td>2344297</td>
<td>-166.1418</td>
<td>3853280</td>
<td>3187266</td>
</tr>
<tr>
<td>-166.1371</td>
<td>4041416</td>
<td>496082</td>
<td>-166.0297</td>
<td>4123478</td>
<td>3587416</td>
</tr>
<tr>
<td>-165.8401</td>
<td>3953486</td>
<td>1473849</td>
<td>-165.6313</td>
<td>3938636</td>
<td>2520377</td>
</tr>
<tr>
<td>-165.5271</td>
<td>4065410</td>
<td>3889996</td>
<td>-165.5271</td>
<td>4061792</td>
<td>3898996</td>
</tr>
<tr>
<td>-165.4541</td>
<td>3987596</td>
<td>332460</td>
<td>-165.285</td>
<td>4028042</td>
<td>1870007</td>
</tr>
<tr>
<td>-165.1094</td>
<td>4055222</td>
<td>1252200</td>
<td>-164.0632</td>
<td>4049642</td>
<td>1649274</td>
</tr>
<tr>
<td>-164.7454</td>
<td>4031930</td>
<td>1120501</td>
<td>-164.6912</td>
<td>3825938</td>
<td>1879727</td>
</tr>
<tr>
<td>-164.4536</td>
<td>3994580</td>
<td>2513494</td>
<td>-163.9755</td>
<td>3848870</td>
<td>1044042</td>
</tr>
</tbody>
</table>

3.0.6 Conclusion

We did not see any significant improvement due to the Covariance restriction. The slight better behaviour of the results presented above are within the range of random variation. Over many batches of runs we could not see a clear trend indicating that one of the two methods yielded better results. While monitoring runs, we could see that the Covariance restriction only affects a very minor number of runs and that it is not necessarily always beneficial. Additional restrictions will be necessary to limit the huge number of BFGS evaluations during an optimization.
References


