An efficient conjugate direction method with orthogonalization for large-scale quadratic optimization problems

Article *in* Optimization Methods and Software · April 2007

DOI: 10.1080/10556780500532209

CITATIONS
0

READS
78

2 authors:

Edouard Boudinov
ASR
673 PUBLICATIONS **4,059** CITATIONS

SEE PROFILE

Arkadiy Isaakovich Manevich
Dniepropetrovsk National University, Ukraine
41 PUBLICATIONS **175** CITATIONS

SEE PROFILE

Optimization Methods and Software
Vol. 00, No. 00, Month 200x, 1–26

An efficient conjugate direction method with orthogonalization for large-scale quadratic optimization problems

EDOUARD R. BOUDINOV [a,*] and ARKADIY I. MANEVICH[b]

[a] *GSLA, FORTIS Bank, Amsterdam, The Netherlands.*
[b] *Department of Computational Mechanics and Strength of Structures, Dniepropetrovsk National University, Dniepropetrovsk, Ukraine.*

(Received 00 Month 200x; In final form 00 Month 200x)

A new conjugate direction method is proposed, which is based on an orthogonalization procedure and does not make use of line searches for the conjugate vector set construction. This procedure prevents the set of conjugate vectors from degeneracy and eliminates high sensitivity to computation errors pertinent to methods of conjugate directions, resulting in an efficient algorithm for solving large-scale ill-conditioned minimization problems without preconditioning. The advantages of our algorithms are illustrated by results of the extensive numerical experiments with large-scale quadratic functions.

Keywords: optimization, unconstrained minimization, conjugate directions methods, large-scale problems, orthogonalization

AMS Subject Classification: 90C53; 90C06

1 Introduction

Conjugate directions (CD) methods have, it is widely acknowledged, great merits and one principal drawback, all being a consequence of the basic procedure of generating sequential conjugate directions. The main merits, which are particularly significant at solving large-scale problems, are economy in the operation count and minimum storage requirements. The principal drawback is accumulation of errors over successive iterations. At solving optimization problems with thousands variables even very small errors (say, of order 10^{-10}) at each iteration lead to disturbance and derangement of the conjugate vector set and can result in failing the optimization procedure.

The main source of errors is the inherent link between the linear search and constructing conjugate directions in the classical conjugate gradient (CG) al-

*Corresponding author. Tel.: +31 20 527 2872; E-mail: edouard.boudinov@nl.fortisbank.com
†E-mail: aman@atlantis.dp.ua

gorithms. The basic procedure of conjugation in the classical CG-algorithm strongly depends upon accuracy of linear searches (only exact locating linear minima along *all* preceding conjugate vectors provides a correct new conjugate vector), and even inevitable roundoff errors in locating linear minima result in gradual violation of conjugacy which becomes significant in large-scale problems.

It is clear that efficient CD-algorithms for large-scale problems should construct accurate conjugate directions independently of accuracy of linear searches on each iteration. It has been understood already in 1970's, when algorithms without line searches have been proposed, first with reference to quasi-Newton methods, and then to CD-methods.

Dixon [8] proposed a modified form of the quasi-Newton family of variable metric algorithms that generates, without accurate linear searches, the same sequence of matrices and the same set of conjugate directions as if accurate linear searches had been used. The modified method had been tested on the standard non-quadratic test functions of $N = 2..10$ variables, and it turned out to be competitive with the most of the efficient unconstrained algorithms available at that time.

Nazareth [9] developed a conjugate direction algorithm generating conjugate directions and maintaining finite termination over a quadratic function without requiring that line searches be exact. Instead of usual formula for constructing successive conjugate vectors

$$d_{j+1} = -g_{j+1} + \beta_j d_j, \qquad \beta_j = \frac{(y_j, g_{j+1})}{(y_j, d_j)}, \tag{1}$$

where g_j is the gradient of the objective function at x_j and $y_j \equiv g_{j+1} - g_j$, he sought a new conjugate direction in the form

$$d_{j+1} = -y_j + \sum_{i=1}^{j} r_{i,j} d_i, \qquad r_{i,j} \in \Re \tag{2}$$

and obtained the following expression

$$d_{j+1} = -y_j + r_{i,j-1} d_{j-1} + r_{i,j} d_j, \qquad r_{i,j} = \frac{(y_i, y_j)}{(y_i, d_j)} \tag{3}$$

((x, y) denotes the scalar product of x and y). Apart from modifying formula for conjugate directions, it is necessary after $N + 1$-th iteration to perform an additional step determined by a correction term (accumulated on each

iteration)

$$C = \sum_{i=1}^{N} \epsilon_i d_i, \tag{4}$$

where ϵ_i is defined by the step length and the directional derivative on i-th iteration. Note that the Nazareth's algorithm uses the gradient difference y_j instead of the gradient itself, see Eq. (3). It is easy to check that the gradient differences y_j at any two points on a given line have identical directions (on quadratics), in distinction from the gradient g_{j+1}, and therefore formula (3) develops identical conjugate directions for exact and inexact linear searches. This algorithm was successfully examined on quadratic and non-quadratic test-functions, but also only for small numbers of variables ($N < 10$).

Another modification of the CD-method was proposed in paper [13] (see also [21]). In order to provide an accurate conjugate directions set without line searches the general procedure for constructing conjugate vectors (type of the Gram-Schmidt procedure) was used: conjugacy conditions with respect to all already found conjugate vectors were imposed to determine a new vector on the iteration. Only one step was made per iteration, but this step included, apart the movement along a new conjugate direction, displacements along all preceding conjugate vectors (these Newton-like displacements were deduced directly from the knowledge of the gradients projections at the last two points). In other words, in distinction from [9], the correction terms were accounted for on each iteration, and not only along one preceding vector (that should be done for a quadratic function in precise arithmetic), but also along all found conjugate vectors, for correction of roundoff errors and for general functions. High efficiency of the algorithm was verified in numerical experiments with quadratic and non-quadratic test-functions, but with not larger than $N = 40$ variables. Of course, using the general procedure of constructing conjugate directions deprives the CD-algorithm of its important advantage ($O(n)$ storage requirement), and this circumstance constricted the area of applicability of this technique at that time.

Since then a good deal of effort has been undertaken in the study of convergence of CD-algorithms with inexact linear search ([14], [15] and others), as applied to non-quadratic functions. But the pure mathematical aspect of the problem is here less important, in our opinion, than the computational aspect. Existing CD-algorithms, using preconditioning, are often able to solve optimization problems with thousands variables. But their theoretical property of quadratic termination holds only in precise arithmetic, and these algorithms can undergo essential difficulties at solving large-scale problems, in particular, for quadratic functions with tens and hundreds thousands of variables.

In our opinion, the key problem in improvement of CD-methods is elaboration of perfect schemes of calculations that have very high computational accuracy at all stages of the algorithm. The above mentioned algorithms [9], [13] deal with conjugate vectors sets, which are rather close to degenerative ones for ill-conditioned problems. So the rounding errors can result in degenerating these sets for large-scale minimization problems.

In this paper a new CD - method without linear searches is proposed. The main basic idea is to use, in constructing conjugate vectors, the component of the function gradient that is orthogonal to the subspace of preceding conjugate vectors (not the gradient itself). These "normal" components may be found at arbitrary points of a conjugate vector line, and their directions coincide with those of the gradients at points of linear minimum, so we obtain the same conjugate vectors set as do the CD-algorithms with exact linear searches. Moreover, the new algorithm finds the conjugate directions with much higher accuracy, since the most computations are performed on the base of orthonormalized vectors set formed by the gradients normal projections (instead of the conjugate vectors set), which cannot degenerate. Note that the normal components of gradients can be easily calculated with extremely high accuracy.

At each iteration only one step is done, which includes displacements along the last two conjugate vectors, so the movement occurs in the hyper-plane determined by these conjugate vectors. Along each conjugate vector two successive steps are made: a preliminary step on the iteration, which specified this vector, and the Newton-like step on the next iteration. Such a technique allows attaining practically exact conjugate directions (for a quadratic), without any function evaluations for linear minimization, crucially decreasing sensitivity to computation errors and ensuring considerable decrease of the number of functions and gradients evaluations and the running time.

Here we present the method that has no restart procedures implemented, so in its direct form the algorithm is applicable only to quadratic functions. Problems of this type are important in their own right, and they also arise as subproblems in various methods for unconstrained and constrained optimization.

In Section 2 we state the basic algorithm with minimal storage requirements and present its justification. A special means for additional suppression of rounding errors in large-scale minimization is discussed in Section 3. A modified (long-recurrence) algorithm, which is expected to be more powerful for ill-conditioned problems, is proposed in Section 4. In Section 5 the performance of the algorithms is demonstrated by applying them to a set of standard quadratic test functions and is compared to that of the classical conjugate gradients and other known methods.

2 Description of the method

2.1 *Basic algorithm*

Let us consider the following process over a quadratic function

$$f(x) = \frac{1}{2}(\mathbf{A}x, x) + (b, x), \qquad x, b \in \Re_N, \tag{5}$$

where $\mathbf{A}$ is an $N \times N$ symmetric positive definite matrix.

$\mathbf{k = 1}$: at a point x_1 vectors d_1, n_1, and the step are:

$$d_1 = n_1 = -\frac{g_1}{\|g_1\|}, \qquad x_2 = x_1 + \delta_1 d_1, \tag{6}$$

where $g_k = g(x_k)$ is the gradient of the objective function $f(x)$, and δ_1 is a trial step, which may be, in principle, an arbitrary one.

$\mathbf{k > 1}$: define the k-th normal vector

$$n_k^* = -g_k + \gamma_{k-1} n_{k-1}, \qquad \gamma_{k-1} = (g_k, n_{k-1}), \qquad n_k = \frac{n_k^*}{\|n_k^*\|}, \tag{7}$$

$$d_k^* = n_k + \beta_{k-1} d_{k-1}, \qquad \beta_{k-1} = -\frac{(n_k, y_{k-1})}{(d_{k-1}, y_{k-1})}, \qquad d_k = \frac{d_k^*}{\|d_k^*\|}, \tag{8}$$

$$x_{k+1} = x_k + \alpha_{k-1} d_{k-1} + \delta_k d_k, \tag{9}$$

$$\alpha_{k-1} = -\frac{(g_k, d_{k-1})}{(y_{k-1}, d_{k-1})} \delta_{k-1}, \tag{10}$$

where

$$y_{k-1} = g_k - g_{k-1}. \tag{11}$$

Strategy of updating δ_k may be different. The implemented formula is given in Section 2.4.

The algorithm stops when, for a given ϵ, the condition $\|g_k\| \leq \epsilon$ is satisfied (see also Section 2.4 for implementation details).

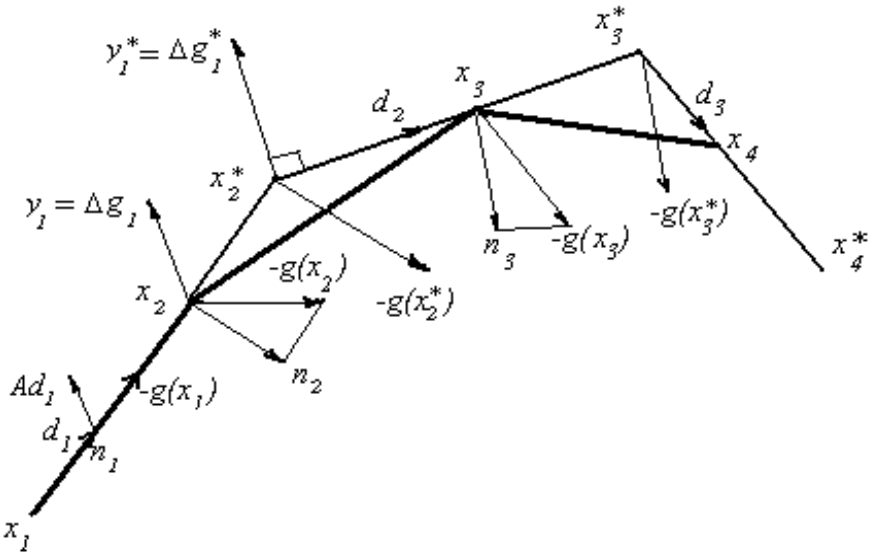

Figure 1. The trajectory of the moving point in the proposed algorithm

2.2 *Discussion*

According to (9), the motion on the k-th iteration occurs in the hyper-plane spanned by the last two conjugate vectors. The component of this step along direction d_k is a preliminary step δ_k. The component of the step along the direction d_{k-1} is the Newton-like step α_{k-1} (10) obtained from the gradient projections on d_{k-1} at the last two points x_{k-1} and x_k. The sum of two displacements along this vector (δ_{k-1} on $k-1$-th iteration and α_{k-1} on k-th iteration) provides the point x_{k+1} be the point of minimum along d_{k-1} for a quadratic.

Let us compare the proposed basic method with the known CD-algorithms that use exact linear searches, see [2], [3], [4].

The proposed algorithm does only one step on each iteration (consisting of components along vectors d_{k-1} and d_k), and determination of the step does not require special function evaluations. Instead of linear searches, two sequential steps are done along each conjugate direction.

Distinction of this method from the known CD-algorithms is illustrated in Fig. 1. In CD-algorithms the trajectory of motion is $x_1 x_2^* x_3^* x_4^*$ (the quantities marked by "*" relate to the points of line minima). In our algorithm the trajectory is $x_1 x_2 x_3 x_4$ (bold lines).

In contrast to the conjugate gradient methods with the exact line searches, which demand the line minimum along the d_k be reached on the current k-th iteration (as well as on all preceding iterations), our method proceeds from assumption that on the k-th iteration only the minima along $\{d_i\}$ $(i = 1, \ldots, k-1)$ be reached. This is a principal difference because on the k-th iteration the gradient projections on the $\{d_i\}$ $(i = 1, \ldots, k-1)$ are known at the two sequential points, but the gradient projection on d_k is known at the last point only. This creates the opportunity to perform only the Newton-like steps using these gradient projections without any function evaluations.

Note that the conjugate gradient methods with exact line searches also provide a set of orthogonal vectors $\{g_i\}$ $(i = 1, \ldots, k)$, which is identical (after normalization) to our set $\{n_i\}$ $(i = 1, \ldots, k)$ (in our algorithm the gradients $\{g_i\}$ $(i = 1, \ldots, k)$ are not orthogonal). The major difference between these two sets is how they are obtained: the gradient g_k is orthogonal to all $\{g_i\}$ $(i = 1, \ldots, k-1)$ only after the exact linear minimization on $k-1$-th step; the n_k is orthogonal to all $\{n_i\}$ $(i = 1, \ldots, k-1)$ by construction and without any line searches. It is important that the normal vectors sets never degenerates, in distinction from the conjugate directions sets of the known CD-methods. Computations based on these sets should retain high accuracy.

Comparing our algorithm with Nazareth's one, we would like to note the following two major differences. First, Nazareth's algorithm uses vectors y_i in constructing conjugate vectors (3). These vectors, similarly to normal vectors n_i in our method, are collinear at all points of the conjugate vector line, and this allows line searches be avoided. However the set y_i is gradually destroyed and can degenerate when the number of variables is large, and there is no convenient procedure for correcting this set. Second, the account for inexact movements along each conjugate direction (which does not reach the line minimum) can be made in Nazareth's algorithm by the correction term (4) only after N iterations, whereas in our algorithm it can be done on each iteration (see formula (9) and Section 4). It is obvious that the last procedure is preferable, especially for non-quadratic functions.

The orthogonalization procedure, used in the proposed method, is somewhat similar to the orthogonalization methods of computational linear algebra (matrix orthogonalization [1] and others). Of course, our algorithm and the linear algebra methods solve different problems. Our procedure does not assume that the matrix is given, only an algorithm of computation of an objective function should be supplied. The problem of linear minimization does not arises in linear algebra at all, so the idea of replacing of linear searches by orthogonalization is not applicable directly to linear algebra. But nevertheless these methods are inherently close since they prevent the accumulation of calculation errors by means of orthogonalization.

2.3 *Justification of the basic algorithm*

The following statements can be proved.

Theorem. *The above procedure (Eqs. (5)-(11)) generates for a quadratic function with symmetric positive-definite matrix* $\mathbf{A}$ *an orthonormalized vector set* $\{n_i\}$ $(i = 1, \ldots, k)$ *and an* $\mathbf{A}$*-conjugate vector set* $\{d_i\}$ $(i = 1, \ldots, k)$ *on the* k*-th iteration. The vectors* d_i *provide the same conjugate directions that would have been obtained by the CD-methods with exact line searches.*

Proof. We use the mathematical induction method. Denote by Ω_k the subspace spanned by vectors $\{g_i\}$ $(i = 1, \ldots, k)$. It follows from Eqs. (6), (7), (8) that vectors n_i and d_i $(i \leq k)$ belong to the subspace Ω_k, and vectors y_i $(i \leq k-1)$ also belong to the subspace Ω_k, accordingly to (11).

As was noted above, the point x_k is the point of minimum in the subspace Ω_{k-2}, i.e. the gradient at this point is orthogonal to all vectors of this subspace:

$$(g_k, n_i) = 0 \text{ for } i \leq k-2. \tag{12}$$

At the second iteration the vectors n_1, n_2 are orthonormalized (from Eqs. (6), (7)) and d_1, d_2 are $\mathbf{A}$-conjugate. The latter statement follows from (6),(8) taking into account that for quadratic functions with the first step (6) $y_1 = g_2 - g_1 = \mathbf{A}(x_2 - x_1) = \delta_1 \mathbf{A} d_1$.

Let after $k-1$-th step (at the point x_k) the set n_i $(i \leq k-1)$ be an orthonormalized vector set in the subspace Ω_{k-1}, and d_i $(i \leq k-1)$ be a conjugate vector set in this subspace.

Formula (7) provides a vector n_k^* (and n_k), which is orthogonal to n_{k-1}. Let us prove that n_k^* is also orthogonal to all $\{n_i\}$ $(i = 1, \ldots, k-2)$. One has

$$(n_k^*, n_i) = (-g_k + \gamma_{k-1} n_{k-1}, n_i) = -(g_k, n_i) + \gamma_{k-1}(n_{k-1}, n_i) = 0, \tag{13}$$

where $i = 1, \ldots, k-2$. The first scalar product in the right hand side (13) vanishes due to (12), the second one equals to 0 as $\{n_i\}$ $(i = 1, \ldots, k-1)$ is an orthonormalized vector set.

Consider now the vector d_k^* specified by formulae (8). Note at first that vectors y_k, defined by (11), do not coincide for $k > 2$ with vectors $\delta_k \mathbf{A} d_k$ (in distinction from the known CD-algorithms), since at each iteration the step includes displacements along the two vectors d_{k-1} and d_k (9). Multiplying (9) by $\mathbf{A}$ we have

$$y_k = \mathbf{A}(x_{k+1} - x_k) = \alpha_{k-1} \mathbf{A} d_{k-1} + \delta_k \mathbf{A} d_k. \tag{14}$$

It follows from this expression that $\mathbf{A} d_k \in \Omega_{k+1}$. Replacing here k with $k-1$

and substituting (14) into expression for β_{k-1} (8) we obtain

$$\beta_{k-1} = -\frac{(n_k, \mathbf{A}d_{k-1})}{(d_{k-1}, \mathbf{A}d_{k-1})}. \tag{15}$$

Here we took into account that $(d_{k-1}, \mathbf{A}d_{k-2} = 0)$ (as d_i $(i \leq k-1)$ is a conjugate vector set in the subspace Ω_{k-1}) and $(n_k, \mathbf{A}d_{k-2}) = 0$ (as $\mathbf{A}d_{k-2}$ belongs to the subspace Ω_{k-1}). Then it follows from formula for d_k (8) that d_k and d_{k-1} are $\mathbf{A}$-conjugate.

Let us prove that d_k^* is $\mathbf{A}$-conjugate to all $\{d_i\}$ $(i = 1, \ldots, k-2)$. We have

$$(d_k^*, \mathbf{A}d_i) = (n_k + \beta_{k-1}d_{k-1}, \mathbf{A}d_i) = (n_k, \mathbf{A}d_i) + \beta_{k-1}(d_{k-1}, \mathbf{A}d_i) = 0. \tag{16}$$

The first scalar product vanishes because all vectors $\mathbf{A}d_i$ $(i = 1, \ldots, k-2)$ belong to subspace Ω_{k-1}, which is orthogonal to n_k, the second one – because the vector set $\{d_i\}$ $(i = 1, \ldots, k-1)$ is $\mathbf{A}$-conjugate.

Let us prove now that the above algorithm provides, at the k-th step, the same conjugate directions set d_i $(i = 1, \ldots, k)$ as do CD-methods with exact line searches. Let x_k^* be the point of line minimum along d_{k-1}. It is sufficient to show that at each iteration the normal vector n_k defined by (7) is collinear to the gradient g_k^* at point x_k^* because this would mean the formula (8) for vector d_k^* is equivalent to the usual formula of CD-methods with exact line searches.

For $k = 2$ this is almost evident. Indeed, gradients at all points of the direction $d_1 = g_1/\|g_1\|$ belong to the same hyperplane Ω_2 spanned by vectors g_1 and g_2, since the difference of gradients at two point x_2^* and x_2 (arbitrary point on this line) equals to

$$g_2^* - g_2 = \mathbf{A}(x_2^* - x_2) = \lambda_1 \mathbf{A}d_1, \tag{17}$$

where λ_1 is a scalar and the vector $\mathbf{A}d_1 = (g_2 - g_1)/\delta_1$ (see (6)) belongs to Ω_2. The vector n_2 belongs to the same hyperplane Ω_2 because it is a linear combination of g_2 and n_1 (from (7)), where the vectors n_1 and d_1 are identical. As the vectors n_2 and g_2^* are orthogonal to d_1, they are collinear.

By induction, assume now that the vectors n_i and g_i^* are collinear for $i = 1, \ldots, k-1$, and let us show that n_k and g_k^* are collinear. Denote now by g_k the gradient at an arbitrary point x_k on the d_k-line. Similarly to (17) we have

$$g_k^* - g_k = \mathbf{A}(x_k^* - x_k) = \lambda_k \mathbf{A}d_{k-1}. \tag{18}$$

As $\mathbf{A}d_{k-1} \in \Omega_k$, the vector g_k^* also belongs to this subspace, as well as n_k. Both these vectors are orthogonal to the subspace Ω_{k-1}, so they are collinear.

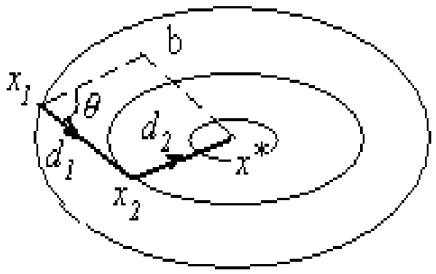

Figure 2. Relation between steps

This completes the proof.

2.4 *Details of implementation*

Two important items require special attention – (a) the choice of "trial step" δ_k (the component of the k-th step along the current conjugate vector d_k) and (b) the stopping criterion.

(a) The magnitude of δ_k may be, as was noted, arbitrary for a quadratic function (in our first implementation of the algorithm we took δ_k as a constant, and the algorithm reached the minimum for all quadratic test-functions exactly in $N + 1$ steps, not less). In order to reduce the number of steps, if it is possible, and to adapt the algorithm to non-quadratic functions, we take here the variable value of δ_k. It is chosen depending on the step along the preceding conjugate vector d_{k-1} and the angle between d_{k-1} and d_k.

Consider, for simplicity, a function of two variables. Let at point x_1 we have conjugate direction d_1 (Fig. 2), and x_2 is the point of line minimum along d_1. The step length $r_1 = \|x_2 - x_1\|$ can be expressed in terms of the directional derivative and the curvature χ_1 at point x_1 along d_1 as follows:

$$r_1 = -\frac{(g(x_1), d_1)}{\chi_1}. \tag{19}$$

It is known that the step length r_2 from point x_2 to the minimum point x^* along d_2 is equal to the distance $x_1 b$ between the point x_1 and the line minimum along the direction collinear to d_2. This distance can be found similarly to (19):

$$r_2 = -\frac{(g(x_1), d_2)}{\chi_2}, \tag{20}$$

where χ_2 is the curvature along d_2. If to assume that direction of d_1 coincides with the negative gradient at x_1 then

$$(g(x_1), d_2) = (g(x_1), d_1)\cos\theta, \tag{21}$$

where θ is the angle between d_1 and d_2. As the relationship between the curvatures χ_1 and χ_2 is unknown, we assume, in our approximate estimates, them to be equal. So we get the following approximate relationship:

$$r_2 = r_1\cos\theta. \tag{22}$$

In our algorithm $\cos\theta_k$ on k-th step can be found from (8). As vectors d_{k-1} and n_k are orthogonal and normalized, we have

$$\|d_k^*\| = \sqrt{1+\beta_{k-1}^2} \tag{23}$$

and hence

$$\cos\theta_k = \frac{(d_k^*, d_{k-1})}{\|d_k^*\|} = \frac{\beta_{k-1}}{\sqrt{1+\beta_{k-1}^2}}. \tag{24}$$

Accounting that the total step along d_{k-1} up to the line minimum is equal to $(\delta_{k-1}+\alpha_{k-1})$, we obtain the following formula

$$\delta_k = \frac{\beta_{k-1}}{\sqrt{1+\beta_{k-1}^2}}(\delta_{k-1}+\alpha_{k-1}), \tag{25}$$

(b) The stopping criterion also can be based on a simple interpolation formula. As the gradient g_{k+1}^* at the point x_{k+1}^* of line minimum along d_k is collinear (on quadratics) to the normal vector n_{k+1}^* (at point x_{k+1}) and the normal vector varies linearly along d_k, modulus of the gradient g_{k+1}^* ("expected gradient") can be estimated by the following formula

$$\|g_{k+1}^*\| = \frac{\|x_{k+1}^* - x_k\|}{\|x_{k+1} - x_k\|}\|n_{k+1}^*\| = |\frac{\delta_k+\alpha_k}{\delta_k}|\|n_{k+1}^*\|. \tag{26}$$

So, before performing the next step, we calculate the "expected gradient", and if it is less then ϵ, then the step is made only along vector d_k (no displacement along the new vector d_{k+1}), and if the real gradient at point x_{k+1}^* satisfies the condition $\|g_{k+1}^*\| \le \epsilon$, the run terminates. Such an approach excludes a

situation when the trial step takes away the search from the minimum point already found.

3 Suppressing calculation errors

The accuracy of determining the normal vector by formula (7) can be insufficient for large N in the case when the vector g_k makes a small angle to the subspace Ω_{k-1} (then Eq. (7) gives a small difference of close vectors). Accumulating very small rounding errors leads to gradual derangement of the normal and conjugate vector sets. The proposed orthogonalization procedure allows for easy correction of the obtained vector sets.

The simplest way to raise the accuracy is to perform the next iteration by formulae (7) substituting the vector $-g_k$ by the vector n_k^* ("reorthogonalization"). We obtain an improved normal vector n_k^{**}:

$$n_k^{**} = n_k^* + \nu_{k-1} n_{k-1}, \qquad \nu_{k-1} = -(n_k^*, n_{k-1}), \tag{27}$$

which, after normalization, is taken as a new vector n_k. Such an additional iteration turns out to be efficient tool to suppress the calculation errors.

4 Modified algorithm

Solving ill-conditioned large-scale problems may require higher accuracy and additional control of calculation errors. This can be achieved by orthogonalizing the vector n_k not only with respect to n_{k-1} but with respect to all preceding vectors n_i (Gram–Schmidt orthogonalization, compare with [13], [21]). Simultaneously one can add additional corrective displacements along all found vectors d_i. This way leads to a modified algorithm. Equation (7) is transformed into

$$n_k^* = -g_k + \sum_{i=1}^{k-1} \gamma_{ki} n_i, \qquad \gamma_{ki} = (g_k, n_i). \tag{28}$$

Similarly, formula (9) for the step at k-th iteration is replaced by the following one:

$$x_{k+1} = x_k + \sum_{i=1}^{k-1} \alpha_{ki} d_i + \delta_k d_k, \tag{29}$$

with

$$\alpha_{ki} = -\frac{(g_k, d_i)}{(g_k, d_i) - (g_i, d_i)}\delta_i, \quad i = 1, \ldots, k-2, \tag{30}$$

$$\alpha_{k,k-1} = -\frac{(g_k, d_{k-1})}{(y_{k-1}, d_{k-1})}\delta_{k-1}, \tag{31}$$

where δ_i is the total step along the d_i made so far. We may expect that the modified algorithm will be more powerful at solving complex ill-conditioned problems. Note that the modified algorithm in such a form has an obvious drawback — it increases the storage requirements since it is necessary to store not only the preceding vectors n_{k-1} and d_{k-1}, but all vectors n_i, d_i ($i = 1, \ldots, k-1$). In order to lower the storage requirements one can use explicit representations of the conjugate vectors via the normal vectors following from Eqs. (8) and (23). We obtain successively

$$d_1 = n_1 \tag{32}$$

$$d_2 = \frac{\beta_1}{\sqrt{1+\beta_1^2}}n_1 + \frac{1}{\sqrt{1+\beta_1^2}}n_2 \tag{33}$$

$$d_3 = \frac{\beta_1\beta_2}{\sqrt{1+\beta_1^2}\sqrt{1+\beta_2^2}}n_1 + \frac{\beta_2}{\sqrt{1+\beta_1^2}\sqrt{1+\beta_2^2}}n_2 + \frac{1}{\sqrt{1+\beta_2^2}}n_3 \tag{34}$$

$$, \ldots ,$$

$$d_i = \sum_{j=1}^{i}\eta_{ij}n_j, \qquad \eta_{ij} = \prod_{\nu=j}^{i-1}\frac{\eta_{ii}\beta_\nu}{\sqrt{1+\beta_{\nu-1}^2}}, \qquad j \neq i, \tag{35}$$

$$\eta_{ii} = \frac{1}{\sqrt{1+\beta_{i-1}^2}}, \qquad \beta_0 = 1.$$

We see that matrix $\{\eta_{ij}\}$ is expressed through one-dimensional array $\{\beta_j\}$, so instead of storing all vectors $\{d_i\}$ one may store only the array $\{\beta_j\}$.

After rewriting Eq. (28) in the form

$$g_k = \sum_{i=1}^{k}\gamma_{ki}n_i, \qquad \gamma_{kk} = -\|n_k^*\| \tag{36}$$

(remind that in precise arithmetic the coefficients γ_{ki} ($i = 1, \ldots, k-2$) are zeros for a quadratic function) one obtains all quantities needed for Eqs. (30)-(31).

In practice the direct calculation of (g_k, d_i) using (32)–(35) is replaced by recurrence formulas. First, expression (8) with account of (23) can be written as a recurrent formula for d_i:

$$d_i = \frac{1}{\sqrt{1+\beta_{i-1}^2}} \cdot (n_i + \beta_{i-1} d_{i-1}), \quad \cdots, \qquad d_1 = n_1. \tag{37}$$

Then, using (36), we obtain a recurrent formula for (g_k, d_i):

$$(g_k, d_i) = \frac{1}{\sqrt{1+\beta_{i-1}^2}} \cdot (\gamma_{ki} + \beta_{i-1}(g_k, d_{i-1})), \quad \ldots, \quad (g_k, d_1) = \gamma_{k1}. \tag{38}$$

So, in order to determine the k-step (29) in the modified algorithm, one should store only the following one-dimensional arrays (along with vectors $\{n_i\}$ $(i = 1, \ldots, k)$):

$$\{\beta_i\}, \quad (g_i, d_i), \quad \{\delta_i\} \quad (i = 1, \ldots, k-1). \tag{39}$$

Our implementation of the modified algorithm is given below.

ALGORITHM:

1. x_1, $g_1 = g(x_1)$
2. For $k = 1, 2, \ldots, N$
3. If $k = 1$ then $d_1 = n_1 = -g_1/\|g_1\|$; $x_2 = x_1 + \delta_1 d_1$
4. Else
5. $\gamma_{k,k-1} = (g_k, n_{k-1})$; $\quad n_k^* = -g_k + \gamma_{k,k-1} n_{k-1}$
6. For $i = 1, 2, \ldots, k-2$
7. $\gamma_{ki} = -(n_k^*, n_i)$; $\quad n_k^* = n_k^* + \gamma_{ki} n_i$
8. End For
9. For $i = 1, 2, \ldots, k-1$
10. If $i = 1$ then $g_{k1} = \gamma_{k1}$,
11. Else $g_{ki} = (\gamma_{ki} + \beta_{i-1} \cdot g_{k,i-1})/\sqrt{1+\beta_{i-1}^2}$
12. End If
13. End For
14. For $i = 1, 2, \ldots, k-1$
15. $\alpha_{ki} = -g_{ki}\delta_i/(g_{ki} - g_{ii})$
16. End For
17. $n_k^* = n_k^* - (n_k^*, n_{k-1}) \cdot n_{k-1}$
18. $d = n_1$; $\quad step = \alpha_{k1} d$
19. For $i = 2, 3, \ldots, k-1$

20. $d = (n_i + \beta_{i-1} \cdot d)/\sqrt{1+\beta_{i-1}^2}; \quad step = step + \alpha_{ki} \cdot d$
21. End For
22. $x_k^* = x_k + step; \quad \|g_k^*\| = \|n_k^*\| \cdot |(\delta_{k-1} + \alpha_{k,k-1})/\delta_{k-1}|$
23. For $i = 1, 2, \ldots, k-1$
24. $\delta_i = \delta_i + \alpha_{ki}$
25. End For
26. If $\|g_k^*\| < \epsilon\|g_1\|$ then
27. $g_k^* = g(x_k^*)$; If $\|g_k^*\| < \epsilon\|g_1\|$ then Quit
28. End If
29. $n_k = n_k^*/\|n_k^*\|; \quad \beta_{k-1} = \|n_k^*\|/(g_{k,k-1} - g_{k-1,k-1})$
30. $d_k = (n_k + \beta_{k-1} \cdot d)/\sqrt{1+\beta_{k-1}^2}$
31. $g_{kk} = (-\|n_k^*\| + \beta_{k-1} \cdot g_{k,k-1})/\sqrt{1+\beta_{k-1}^2}$
32. $\delta_k = \beta_{k-1}\delta_{k-1}/\sqrt{1+\beta_{k-1}^2}; \quad x_{k+1} = x_k^* + \delta_k \cdot d_k$
33. End If
34. $g_{k+1} = g(x_{k+1}); \quad$ If $\|g_{k+1}\| < \epsilon\|g_1\|$ then Quit
35. End For

Note that we have used here the *modified* Gram-Schmidt orthogonalization (see, e.g., [19]), i.e. the vector n_k^* was computed sequentially: first it was orthogonalized against n_{k-1}, and then the obtained vector was corrected step-by-step by adding other terms $\gamma_{k-1,i}\ n_i,\ i < k-1$, see steps 5-7 and 17. Such a procedure secures the highest possible accuracy.

5 Numerical experiments

The algorithms have been realized in JAVA programming language and have been tested on several quadratic test–functions with the number of variables N up to 1'000'000. Some results of the numerical experiment were presented in our reports on conferences [23], [24].

For comparison we have chosen the implementations of the following four groups of methods: the conjugate gradients methods for optimization from CG+ package [15], our JAVA implementation of the conjugate gradient method for linear algebra [17], the quasi-Newton BFGS method (Broyden, Fletcher, Goldfarb, and Shanno [5], [6], [7], [11], [22]), and the limited-memory L-BFGS-B method [16], [18], [20].

Our implementation of the CG for linear algebra was benchmarked against the MATLAB implementation of the conjugate gradient method (*pcg* function) [22], and it was established that the numbers of the iterations were

identical in these implementations. The CG algorithm for linear algebra was supplied by the function of the matrix-vector product, $(\mathbf{A}, x)$ (and did not include linear minimization along the conjugate directions).

Note that these algorithms were implemented in various programming languages (FORTRAN, MATLAB, JAVA), hence the reader should take it into account when comparing the running times (CPU) for different algorithms.

All computations have been performed on PC Pentium 2.66 GHz with RAM of 785.9 MB in double precision. The gradient was calculated analytically.

The proposed algorithms have only two parameters that should be supplied by a user: an initial step δ_1, and the termination tolerance on the function gradient value ϵ. The δ_1 value was always equal to 0.5. The ϵ value was varied in the range $10^{-10} - 10^{-25}$ depending on the accuracy needed for a particular function minimum.

The same range of the termination tolerance on the function gradient was used for the conjugate gradients methods and the limited-memory method. The number of corrections used in the limited memory matrix was varied from 17 till 500, and the best results for the method are reported. For the quasi-Newton method the termination tolerance on the function value was taken equal to 10^{-20}.

First, we present the results for the following quadratic functions of N variables:

$$F_1(x_1, \ldots, x_N) = \sum_{i=1}^{N} \frac{x_i^2}{i} + \lambda \sum_{i=1}^{N} \sum_{j=i+1}^{N} \frac{x_i\, x_j}{ij}, \tag{40}$$

In all cases the start point was $x^{(0)} = (1, \ldots, 1)$ and the minimum $F(0, \ldots, 0) = 0$. Functions (40) were used to examine the proposed algorithms at solving very large scale quadratic problems with diagonal matrices (if $\lambda = 0$), as well as with dense matrices (if $\lambda = 1$).

Table 1 presents the results obtained for the function F_1 with $\lambda = 1$. The number of variables N was varied from 1000 till 20'000. The "Basic Algorithm" relates to the algorithm described in Section 2. The "Modified Algorithm" relates to the algorithm described in Section 4. In all tables N is the number of variables; ϵ_g and ϵ_x are the accuracy in function gradient and arguments, respectively; k_g is the number of matrix-vector multiplications in the case of the CG for linear algebra and the number of gradient evaluations for other methods; k_f is the number of function evaluations; t is the running time (CPU consumption) in seconds; the column $n_{\vec{n}}^{max}$ is either the number of stored normal vectors, see Eqs. (7) and (28) in the case of the proposed algorithm, or the number of corrections used in the limited memory matrix of L-BFGS-B algorithm.

Table 1. Results for function (40) with $\lambda = 1$

N	ϵ_g	ϵ_x	iter	k_g	k_f	$n_{\vec{n}}^{max}$	t (sec)
Modified Algorithm							
4'000	$< 10^{-12}$	$< 10^{-10}$	144	145	1	143	33
10'000	$< 10^{-12}$	$< 10^{-10}$	196	197	1	195	303
20'000	$< 10^{-12}$	$< 10^{-10}$	247	248	1	246	1525
Basic Algorithm							
4'000	$< 10^{-12}$	$< 10^{-10}$	305	306	1	1	73
10'000	$< 10^{-12}$	$< 10^{-10}$	473	474	1	1	702
20'000	$< 10^{-12}$	$< 10^{-10}$	668	669	1	1	3926
Fletcher-Reeves							
4'000	$< 10^{-12}$	$< 10^{-10}$	830	1666	1666		1275
10'000	$< 10^{-12}$	$< 10^{-10}$	1121	2249	2249		10236
Polak-Ribiére-Polyak							
4'000	$< 10^{-12}$	$< 10^{-10}$	793	1587	1587		1160
10'000	$< 10^{-12}$	$< 10^{-10}$	823	1663	1663		7569
CG for Linear Algebra							
4'000	$< 10^{-12}$	$< 10^{-9}$	268	270			60
10'000	$< 10^{-12}$	$< 10^{-9}$	409	411			563
20'000	$< 10^{-12}$	$< 10^{-10}$	579	581			3180
Quasi-Newton							
4'000	$< 10^{-9}$	$< 10^{-7}$	315	316	2024		1062
Limited memory L-BFGS-B method							
4'000	$< 10^{-12}$	$< 10^{-9}$	188	205	205	500	166
10'000	$< 10^{-12}$	$< 10^{-9}$	249	258	258	500	1270
20'000	$< 10^{-12}$	$< 10^{-9}$	330	342	342	500	6660

For all values of N the minima have been obtained by both proposed algorithms in less than $N + 1$ steps with very high accuracy in function and arguments x_i, and this accuracy can be ensured for very large N. The modified version of the proposed algorithm appears to be most efficient in terms of number of iterations, functions and gradient calls as well as CPU time; the basic algorithm requires the number of iterations and CPU time approximately twice as much as the modified method, but the number of iterations remains much less than the number of variables.

The Fletcher-Reeves and Polak-Ribiere-Polyak algorithms demand the number of iterations approximately by 10 times as much as the modified algorithm. The line minimization used in the conjugate gradients algorithms does not ensure the necessary accuracy in constructing conjugate vectors and thus leads to consumption of lots of CPU through the large number of iterations.

The conjugate gradient method for linear algebra has, in terms of the number of iterations and the running time, slightly higher efficiency than our basic

algorithm, but is at a disadvantage in relation to the modified algorithm.

The quasi-Newton method requires the memory allocation proportional to N^2, this results in 'Out-of-memory' problem when N is larger than approximately 4000 (for the computer used). For $N < 4000$ the running time is close to that of the conjugate gradient algorithms.

In order to check the algorithm for even higher N values we have considered the quadratic function $F_1(x_1, \ldots, x_N)$ with $\lambda = 0$, which has a diagonal Hessian. Note that both the proposed algorithms are independent of whether Hessian is dense or diagonal. But the running time for a function with diagonal Hessian is essentially lower (the search includes at most $N + 1$ determinations of steps, every step determination takes the number of basic operations proportional to N; each iteration includes one gradient evaluation, every gradient evaluation for a quadratic function takes the number of basic operations proportional to N^2 in the case of dense Hessian, and to N in the case of diagonal Hessian).

The results for N values up to 1'000'000 are presented in Table 2. The results obtained for the two proposed algorithms are compared with those of the Fletcher-Reeves algorithm and the L-BFGS-B algorithm.

Table 2. Results for function (40) with $\lambda = 0$

N	ϵ_g	ϵ_x	iter	k_g	k_f	$n_{\vec{n}}^{max}$	t (sec)
Modified Algorithm							
20'000	$< 10^{-12}$	$< 10^{-9}$	241	242	1	240	16
50'000	$< 10^{-12}$	$< 10^{-9}$	324	325	1	324	118
100'000	$< 10^{-12}$	$< 10^{-9}$	406	407	1	405	356
Basic Algorithm							
20'000	$< 10^{-12}$	$< 10^{-9}$	652	653	1	1	8
50'000	$< 10^{-12}$	$< 10^{-9}$	1021	1022	1	1	33
100'000	$< 10^{-12}$	$< 10^{-9}$	1446	1447	1	1	86
1'000'000	$< 10^{-12}$	$< 10^{-9}$	4557	4558	1	1	2490
Fletcher-Reeves							
20'000	$< 10^{-12}$	$< 10^{-10}$	1520	3054	3054		28
50'000	$< 10^{-12}$	$< 10^{-10}$	2616	5245	5245		125
100'000	$< 10^{-12}$	$< 10^{-10}$	3786	7591	7591		361
Limited memory L-BFGS-B method							
20'000	$< 10^{-12}$	$< 10^{-9}$	335	336	336	500	114
	$< 10^{-12}$	$< 10^{-9}$	1102	1182	1182	17	33
50'000	$< 10^{-12}$	$< 10^{-8}$	431	438	438	500	553
	$< 10^{-12}$	$< 10^{-9}$	1745	1882	1882	17	163
100'000	$< 10^{-12}$	$< 10^{-9}$	2434	2625	2625	17	448

One can see that for very large numbers of variables the basic algorithm

becomes preferable. It enables us to solve the problem up to N=1'000'000 performing only about 5000 steps. The storage requirements do not permit the modified algorithm be used for N >100'000 on our computer; note its remarkable performance up to 100'000 variables. The Fletcher-Reeves algorithm demands at least twice the number of iterations in comparison with the basic algorithm. The L-BFGS-B algorithm with the number of corrections, used in the limited memory matrix, equal to 17, provides results comparable to the conjugate gradient methods. In the case of 500 corrections in the limited memory matrix it requires the number of gradient calculations less than the proposed basic algorithm, but it also requires much more CPU and cannot obtain results for the number of variables N=100'000 and higher because of both CPU consumption and storage requirements.

Next we examined the performance of the algorithms on quadratic functions with Hessian that is very close to degenerative one (ill-conditioned problems):

$$F_s(x_1, \ldots, x_N) = \sum_{i=1}^{N} \frac{x_i^2}{i^s}, \qquad s = 1, \ldots, 5 \tag{41}$$

These functions have eigenvalues sets with different rate of decrease, which is governed by the parameter s. Comparison of the results for these functions at various s obtained by the proposed methods, CG - algorithms and L-BFGS-B algorithm is presented in Table 3, for various values of N, and in Figures 3 and 4, for N=10'000.

The modified algorithm has effectively solved all problems up to s=5 with very high accuracy; the number of steps for all N much less than the number of variables. The basic algorithm as well as the CG-methods can solve problems only for $s = 1$ and $s = 2$. These algorithms have approximately the same convergence at $s = 1$. For $s = 2$ the considered CG-algorithms for unconstrained minimization require the number of gradient evaluations about ten times as much as the proposed basic algorithm. The L-BFGS-B algorithm can also solve problems only for $s = 1$ and $s = 2$, similarly to CG-methods, but it requires lots of CPU despite of the numbers of gradient evaluations required much less than those in the CG-methods. The CG algorithm for linear algebra was rather efficient in the cases $s = 1$ and $s = 2$, but its convergence was very slow for $s = 3$, and it was practically unsuitable for larger values of s.

In order to verify that our algorithms are independent of whether Hessian is dense or diagonal also in the case of ill-conditional problems, we considered the quadratic function (5) with the following matrices:

$$QF_{ND1}(x): \qquad a_{ii} = \tfrac{1}{i} \qquad a_{ij}(i \neq j) = \tfrac{1}{ij} \tag{42}$$

$$QF_{ND2}(x): \qquad a_{ii} = \tfrac{1}{i^2} \qquad a_{ij}(i < j) = \tfrac{1}{ij^2} \qquad a_{ij}(i > j) = \tfrac{1}{i^2 j} \tag{43}$$

Table 3. Results for function (41): start point $x_1 = (1, \dots, 1)$ and minimum $F(0, \dots, 0) = 0$

s	N	ϵ_g	ϵ_x	iter	k_g	k_f	$n_{\vec{n}}^{max}$	t (sec)
Modified Method								
s=1	1'000	$< 10^{-15}$	$< 10^{-13}$	105	105	1	104	0.2
	10'000	$< 10^{-15}$	$< 10^{-12}$	226	226	1	225	8.7
	20'000	$< 10^{-15}$	$< 10^{-12}$	283	283	1	282	28.2
s=2	1'000	$< 10^{-15}$	$< 10^{-10}$	202	202	1	201	0.8
	10'000	$< 10^{-15}$	$< 10^{-8}$	605	605	1	604	62
s=3	1'000	$< 10^{-20}$	$< 10^{-12}$	332	332	1	331	2.3
	10'000	$< 10^{-20}$	$< 10^{-9}$	1232	1232	1	1231	262
s=4	1'000	$< 10^{-20}$	$< 10^{-9}$	394	394	1	393	3.2
	10'000	$< 10^{-20}$	$< 10^{-6}$	1625	1625	1	1624	463
s=5	1'000	$< 10^{-25}$	$< 10^{-11}$	498	498	1	497	5.3
	10'000	$< 10^{-25}$	$< 10^{-6}$	2299	2298	1	2298	989
Basic Algorithm								
s=1	10'000	$< 10^{-12}$	$< 10^{-9}$	463	464	1	1	3
s=2	10'000	$< 10^{-12}$	$< 10^{-6}$	19413	19414	1	1	128
Fletcher-Reeves								
s=1	10'000	$< 10^{-12}$	$< 10^{-10}$	1024	2054	2054		9
s=2	10'000	$< 10^{-12}$	$< 10^{-7}$	100012	201008	201008		1129
Polak-Ribiére-Polyak								
s=1	10'000	$< 10^{-12}$	$< 10^{-10}$	764	1538	1538		6
s=2	10'000	$< 10^{-12}$	$< 10^{-6}$	110013	220521	220521		1237
Positive Polak-Ribiere-Polyak: $\beta = \max(\beta, 0)$								
s=1	10'000	$< 10^{-12}$	$< 10^{-10}$	764	1538	1538		7
s=2	10'000	$< 10^{-12}$	$< 10^{-7}$	100012	200512	200512		1113
CG for Linear Algebra								
s=1	10'000	$< 10^{-12}$	$< 10^{-9}$	448	450			1
s=2	10'000	$< 10^{-12}$	$< 10^{-6}$	8997	8999			22
s=3	10'000	$< 10^{-12}$	$< 10^{-2}$	269908	269910			756
Limited memory L-BFGS-B method								
s=1	10'000	$< 10^{-12}$	$< 10^{-9}$	255	267	267	500	36
s=2	10'000	$< 10^{-12}$	$< 10^{-5}$	6973	7495	7495	500	16738

$$QF_{ND3}(x): \qquad a_{ii} = \tfrac{1}{i^3} \qquad a_{ij}(i \neq j) = \tfrac{1}{(ij)^2} \tag{44}$$

$$QF_{ND4}(x): \qquad a_{ii} = \tfrac{1}{i^4} \qquad a_{ij}(i < j) = \tfrac{1}{(ij)^2 j} \qquad a_{ij}(i > j) = \tfrac{1}{(ij)^2 i} \tag{45}$$

$$QF_{ND5}(x): \qquad a_{ii} = \tfrac{1}{i^5} \qquad a_{ij}(i \neq j) = \tfrac{1}{(ij)^3}. \tag{46}$$

One can easily see that the rates of decrease of eigenvalues for these matrices are close to those of functions (41) with corresponding values of s. This explains the results for the number of iterations presented in Table 4. The results of

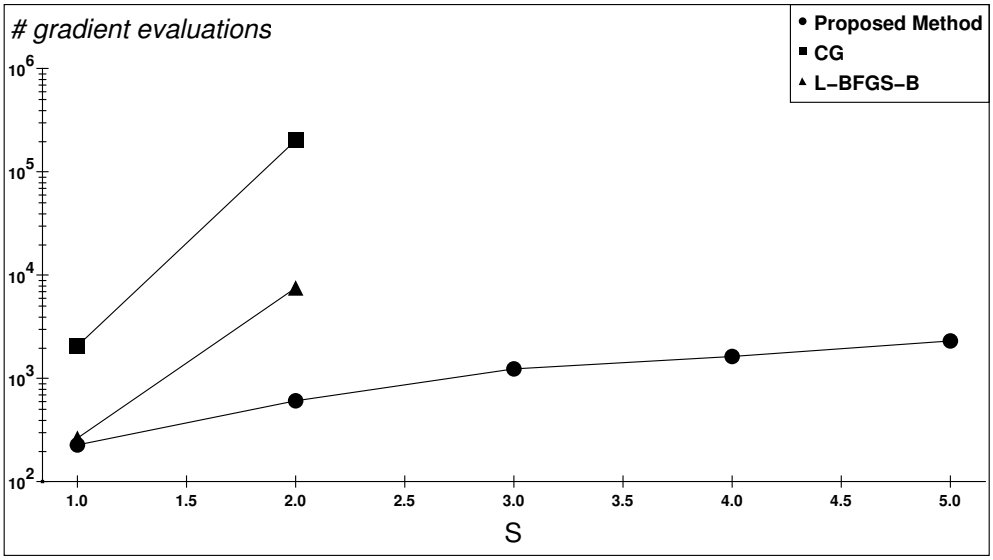

Figure 3. Comparison of efficiency of the proposed algorithm and other methods for function (41) with N=10'000.

the CG-methods and L-BFGS-B algorithm for $s = 1$ and $s = 2$ are not shown here, because they are similar to those for functions Eq. (41). Instead we show the results for Eq. (44) ("twin-brother" of function (41) with $s = 3$) with N=1'000. The running time for these methods is incomparable with that of the proposed algorithm (larger at least by three orders). The CG for linear algebra also compares unfavourably with the modified algorithm.

Finally, Table 5 and Figure 5 present the results for quadratic function with Hilbert matrix for various N values:

$$a_{ij} = \frac{1}{i + j - 1}. \tag{47}$$

The modified algorithm was able to solve the problems with the number of variables up to 500'000: the gradient was lowered by a factor of 10^{13}, the accuracy in arguments is better than 10^{-3}, and just a few gradient calculations were needed. The conjugate gradient methods demonstrate very unstable results dependent on success of the linear searches. The limited memory L-BFGS-B method gives stable results but requires at least five times more gradient evaluations than the proposed algorithm. The CG for linear algebra has effectively solved the problems with relatively small N, but its comparative efficiency decreases with the increase of N.

It seems that the limited memory L-BFGS-B method is the most reliable

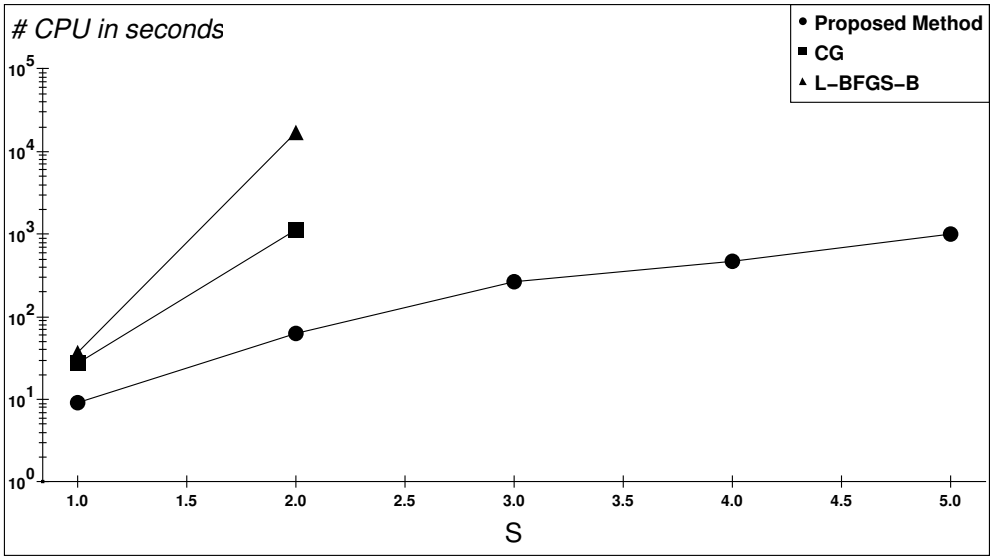

Figure 4. Comparison of running time in different algorithms for function (41) with N=10'000.

Table 4. Results for functions Eqs. (42)-(46): start point $x_1 = (1, \ldots, 1)$ and minimum $F(0, \ldots, 0) = 0$

Function	N	ϵ_g	ϵ_x	iter	k_g	k_f	$n_{\vec{n}}^{max}$	t (sec)
Modified Method								
QF_{ND1}	1'000	$< 10^{-15}$	$< 10^{-13}$	106	106	1	105	1.7
	10'000	$< 10^{-15}$	$< 10^{-13}$	228	228	1	227	328
QF_{ND2}	1'000	$< 10^{-15}$	$< 10^{-10}$	204	204	1	203	6.5
	10'000	$< 10^{-15}$	$< 10^{-9}$	611	611	1	610	1753
QF_{ND3}	1'000	$< 10^{-20}$	$< 10^{-12}$	335	335	1	334	11.6
	10'000	$< 10^{-20}$	$< 10^{-10}$	1245	1245	1	1244	3701
QF_{ND4}	1'000	$< 10^{-20}$	$< 10^{-9}$	397	397	1	396	192
QF_{ND5}	1'000	$< 10^{-25}$	$< 10^{-11}$	501	501	1	500	394
Fletcher-Reeves								
QF_{ND3}	1'000	$< 10^{-15}$	$< 10^{-7}$	241256	514051	514051		39261
Positive Polak-Ribiére-Polyak: $\beta = \max(\beta, 0)$								
QF_{ND3}	1'000	$< 10^{-15}$	$< 10^{-8}$	78095	184888	184888		14112
CG for Linear Algebra								
QF_{ND3}	1'000	$< 10^{-15}$	$< 10^{-7}$	10606	10608			147
Limited memory L-BFGS-B method								
QF_{ND3}	1'000	$< 10^{-15}$	$< 10^{-7}$	6776	7575	7575	500	14773

Table 5. Results for functions Eq. (47): start point $x_1 = (1, \ldots, 1)$ and minimum $F(0, \ldots, 0) = 0$.

N	g/g_1	g	ϵ_x	iter	k_g	k_f	$n_{\vec{n}}^{max}$	t (sec)
Modified Method								
100	$< 10^{-11}$	$< 10^{-10}$	$< 10^{-3}$	13	13	1	12	0.031
1'000	$< 10^{-13}$	$< 10^{-11}$	$< 10^{-3}$	19	19	1	18	0.328
10'000	$< 10^{-13}$	$< 10^{-11}$	$< 10^{-3}$	24	24	1	23	35
20'000	$< 10^{-13}$	$< 10^{-10}$	$< 10^{-3}$	25	25	1	24	146
100'000	$< 10^{-13}$	$< 10^{-10}$	$< 10^{-3}$	28	28	1	27	4137
200'000	$< 10^{-13}$	$< 10^{-10}$	$< 10^{-3}$	29	29	1	28	17468
500'000	$< 10^{-13}$	$< 10^{-10}$	$< 10^{-3}$	30	30	1	29	107780
Positive Polak-Ribiére-Polyak: $\beta = \max(\beta, 0)$								
100		$< 10^{-10}$	$< 10^{-3}$	27	73	73		0.031
1'000		$< 10^{-10}$	$< 10^{-3}$	503	1091	1091		50.5
Fletcher-Reeves								
100		$< 10^{-10}$	$< 10^{-3}$	19	51	51		0.016
1'000		$< 10^{-10}$	$< 10^{-3}$	13038	26719	26719		1234
CG for Linear Algebra								
100	$< 10^{-11}$	$< 10^{-10}$	$< 10^{-3}$	23	25			0.016
1'000	$< 10^{-13}$	$< 10^{-10}$	$< 10^{-3}$	70	72			1.031
10'000	$< 10^{-13}$	$< 10^{-10}$	$< 10^{-3}$	127	129			177
20'000	$< 10^{-13}$	$< 10^{-10}$	$< 10^{-3}$	181	183			1020
Limited memory L-BFGS-B method								
100		$< 10^{-10}$	$< 10^{-3}$	73	78	78	500	0.19
1'000		$< 10^{-11}$	$< 10^{-3}$	138	143	143	500	8.5
10'000		$< 10^{-11}$	$< 10^{-3}$	158	166	166	500	1189

state-of-the-art quasi-Newton method among other known minimization methods that gives stable results for quite large and complex problems. But it cannot control the rounding errors as good as does the proposed algorithm based on the orthogonalization procedure. In addition, it is still too complicated comparing with our algorithm.

6 Conclusions

The use of the proposed orthogonalization procedure for constructing conjugate directions leads to a very efficient algorithm, which overcomes the principal drawback of CD-method – high sensitivity to calculation errors. The algorithm does not include linear searches, replacing them by two sequential steps along each conjugate direction (the preliminary step and the Newton-like step). Such a procedure ensures a significant reduction in the number of gradient evaluations and the running time.

Two variants of the method are investigated. The basic algorithm uses the

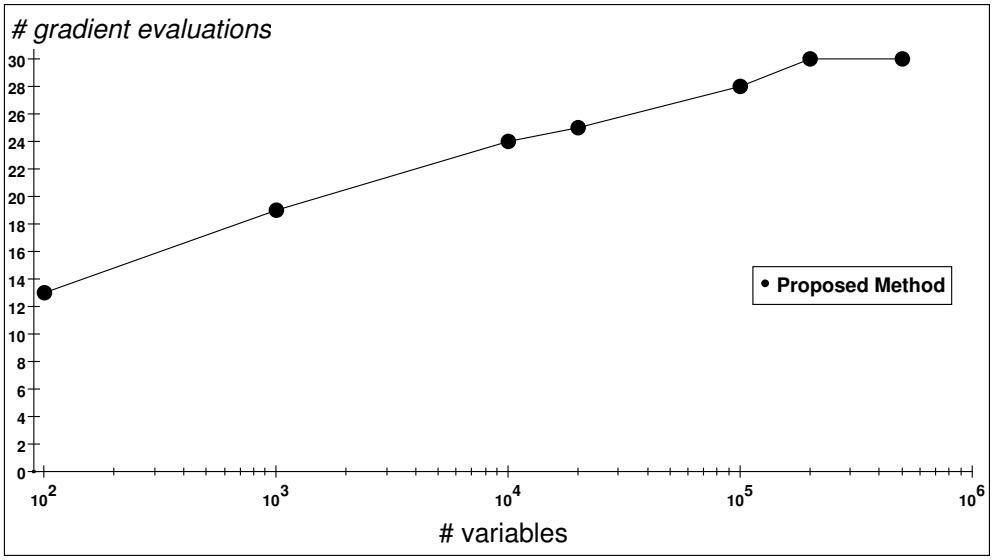

Figure 5. Number of gradient evaluations vs the number of variables for Hilbert functions in the proposed algorithm

preceding normal vector only and requires the storage of the conjugate vector and current normal vector. The modified algorithm employs the entire set of normal vectors in the orthogonalization procedure.

The proposed algorithms have been tested on various quadratic test–functions of very large number of variables N, up to 1'000'000. In all the problems considered the algorithms have successfully found the minima with very high accuracy, and the number of gradient evaluations required by the modified algorithm was considerably less than the number of variables. Comparison with the CG–algorithms, quasi-Newton and limited memory methods has shown the evident advantages of the proposed method in the whole range of N considered. Of course, we are conscious that Fletcher-Reeves, Polak-Ribiére-Polyak and L-BFGS-B methods were intended for general unconstrained minimization problems, so such a comparison within the framework of the quadratic minimization might be criticized. But this may relate only to the number of gradient evaluations and running time, but not to failing or success in solving ill-conditioned problems.

As for comparison of the two variants of the method, the basic algorithm turns out to be preferable at solving very large dimension problems, but it is less efficient (in comparison with the modified algorithm) at minimization of functions with sharply changing eigenvalues. The modified algorithm is found to be the best at solving ill-conditioned minimization problems.

Since our goal is to demonstrate the advantages of the proposed orthogo-

nalization procedure over the procedure of linear minimization in constructing conjugate vectors, we do not consider here modifications of the proposed algorithms, which would make them applicable to general non-quadratic functions. The comparison can correctly be done only on quadratic functions, because in the case of non-quadratic functions the very notion "conjugate directions" becomes fuzzy and approximate. We are convinced that the advantage of every CD-algorithm, which is claimed to be efficient, should be demonstrated, first of all, on quadratic functions of very many variables. The traditional way of testing algorithms at once on quadratic and non-quadratic functions can lead to confusion and mixing different problems. It is well known that in the case of non-quadratic functions and far from the minimum the success of any method depends on a restart procedure [10]. Apparently, no considerable advantages from the precise reconstruction of conjugate vectors can be expected at the first stage of the search (although some new possibilities appear due to additional flexibility provided by the new step composition). As for the second stage, near the optimum, the situation will be similar to that of quadratic functions.

A restart procedure for our algorithm could be implemented without any specific difficulties. In our paper [25] restarts have been incorporated in the algorithm to solve optimization problems with bound constraints. In particular, we considered the case when the first conjugate direction after the restart differs from the minus-gradient.

Large-scale linear problems arise often as a sub-problem in the nonlinear programming, for example, in the well-known truncated-Newton algorithm [12]. Conjugate direction methods are frequently used for solving such sub-problems, and our algorithm could be an efficient tool in these approaches.

We did not include the preconditioning technique in our considerations because this would divert us from the principal task - to show that the CG-algorithm can be improved so that the preconditioning becomes often redundant even in very ill-conditioned minimization problems.

References

[1] K. Lanczos, *Applied Analysis*, Prentice Hall, Inc., 1956.

[2] R. Fletcher and C. Reeves, *Function Minimization by Conjugate Gradients*, Comput. J. 7 (1964), pp. 149-154.

[3] E. Polak and G. Ribiére, *Note sur la convergence des méthodes de directions conjugées*, Rev. Francaise Informat. Recherche Opérationelle 16 (1969), pp. 35-43.

[4] B.T. Polyak, *The Conjugate gradient method in extreme problems*, USSR Comput. Math. And Math. Phys. 9 (1969), pp. 94-112 (in Russian).

[5] C.G. Broyden, *The Convergence of a Class of Double-rank Minimization Algorithms*, J. Inst. Maths. Applics. 6 (1970), pp. 76–90.

[6] D. Goldfarb, *A Family of Variable Metric Updates Derived by Variational Means*, Mathematics of Computing 24 (1970), pp. 23–26.

[7] D.F. Shanno, *Conditioning of Quasi-Newton Methods for Function Minimization*, Mathematics of Computing 24 (1970), pp. 647–656.

[8] L.C.W. Dixon, *Conjugate Directions without Linear Searches*, J. Inst. Maths. Applic. Vol. 11 (1973), pp. 317–328.
[9] L. Nazareth, *A Conjugate Direction Algorithm without Line Searches*, Journal of Optimization Theory and Applications 23 (1977), pp. 373–387.
[10] M.J.D. Powell, Restart procedures for the conjugate gradient method , Mathematical Programming, Vol. 12 (1977), pp. 241-254.
[11] R. Fletcher, *Practical Methods of Optimization*, Vol. 1. J. Wiley, 1980.
[12] R.S. Dembo and T. Steihaug, *Truncated-Newton algorithms for large-scale unconstrained optimization*, Mathematical Programming, No. 26 (1983), pp. 190–212.
[13] A.I. Manevich and P.I. Polyanchikov, *Single-step method of conjugate directions*, Izvestija of USSR Academy of Sciences, Technical Cybernetics, No. 6 (1984), pp. 41–47 (in Russian).
[14] M. Al-Baali, *Descent Property and Global Convergence of the Fletcher-Reeves Method with Inexact Line Search*, IMA J. of Numer. Analysis, vol. 5 (1985), pp. 121–124.
[15] J.C. Gilbert and J. Nocedal , *Global Convergence Properties of Conjugate Gradient Methods for Optimization*, SIAM Journal on Optimization 2 (1992), pp. 21–42.
[16] C. Zhu, R.H. Byrd, P. Lu and J. Nocedal, *L-BFGS-B: FORTRAN Subroutines for Large Scale Bound Constrained Optimization*, Tech. Report, NAM-11, EECS Department, Northwestern University, 1994.
[17] R. Barrett, M. Berry, T. F. Chan, et al., *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*, SIAM, Philadelphia, 1994.
[18] R.H. Byrd, P. Lu, J. Nocedal and C. Zhu, *A limited memory algorithm for bound constrained optimization*, SIAM J. Scientific Computing 16 (1995), no. 5, pp. 1190–1208.
[19] N.J. Higham, *Accuracy and Stability of Numerical Algorithms*, SIAM, Philadelphia, PA (1996).
[20] C. Zhu, R.H. Byrd, P. Lu and J. Nocedal, *Algorithm 778: L-BFGS-B: Fortran subroutines for large-scale bound-constrained optimization*, ACM Trans. Math. Software, 23 (1997), pp. 550–560.
[21] A.I. Manevich and E. Boudinov, *An efficient conjugate directions method without linear minimization*, Nuclear Instruments and Methods in Physics Research, **A** 455 (2000), pp. 698–705.
[22] *Optimization Toolbox for Use with MATLAB. User's Guide. Version 2*, The MathWorks Inc., 2000.
[23] A. Manevich and E. Boudinov *A Conjugate Directions Method with Orthogonalization For Large-Scale Problems*, In: Proceedings of the European Congress on Computational Methods in Applied Sciences and Engineering ECCOMAS04, Jyväskylä, 24–28 July 2004, Editors: P. Neittaanmäki, T. Rossi, K. Majava, and O. Pironneau, 2004.
[24] E. Boudinov and A. Manevich, *An Efficient Conjugate Directions Method Without Linear Searches*, In Springer Series: Operations Research Proceedings, Conference 2004, Editors: Fleuren, Hein; den Hertog, Dick; Kort, Peter; **XIV**, pp. 327-334, 2005.
[25] E. Boudinov and A. Manevich, *An Extension of the Conjugate Directions Method With Orthogonalization to Large-Scale Problems With Bound Constraints*, PAMM, 2005 (the paper is accepted for publication).