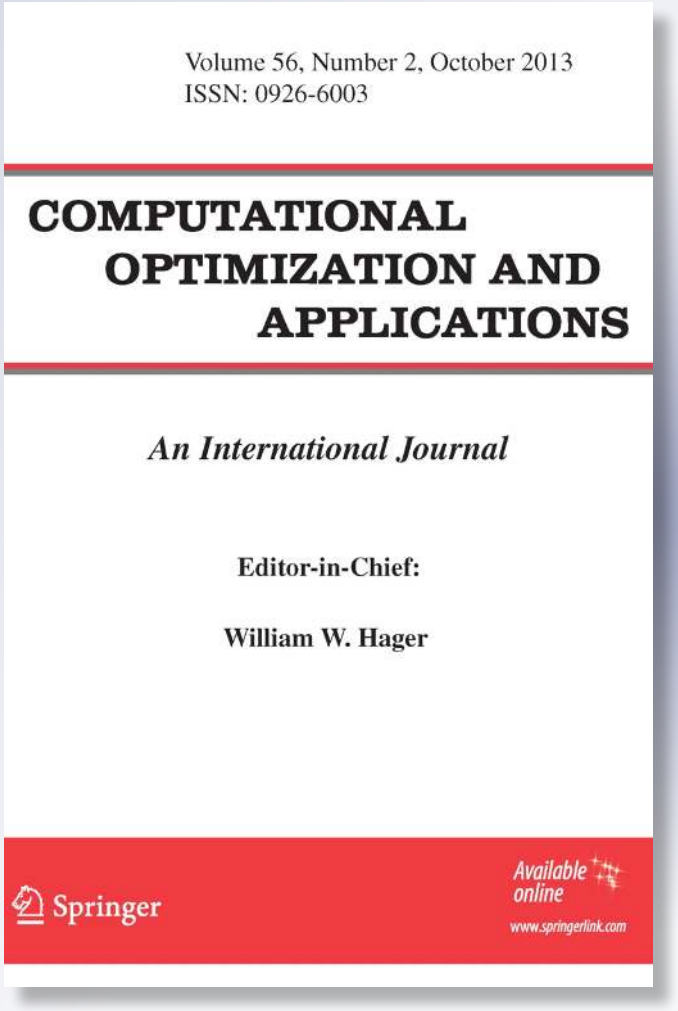
Preconditioning Newton–Krylov methods in nonconvex large scale optimization

Giovanni Fasano & Massimo Roma

Computational Optimization and Applications
An International Journal

ISSN 0926-6003
Volume 56
Number 2

Comput Optim Appl (2013) 56:253-290
DOI 10.1007/s10589-013-9563-6



Springer

Comput Optim Appl (2013) 56:253–290
DOI 10.1007/s10589-013-9563-6

Preconditioning Newton–Krylov methods in nonconvex large scale optimization

Giovanni Fasano · Massimo Roma

Received: 3 December 2009 / Published online: 7 May 2013

Abstract We consider an iterative preconditioning technique for non-convex large scale optimization. First, we refer to the solution of large scale indefinite linear systems by using a Krylov subspace method, and describe the iterative construction of a preconditioner which does not involve matrices products or matrices storage. The set of directions generated by the Krylov subspace method is used, as by product, to provide an approximate inverse preconditioner. Then, we experience our preconditioner within Truncated Newton schemes for large scale unconstrained optimization, where we generalize the truncation rule by Nash–Sofer (Oper. Res. Lett. 9:219–221, 1990) to the indefinite case, too. We use a Krylov subspace method to both approximately solve the Newton equation and to construct the preconditioner to be used at the current outer iteration. An extensive numerical experience shows that the proposed preconditioning strategy, compared with the unpreconditioned strategy and PREQN (Morales and Nocedal in SIAM J. Optim. 10:1079–1096, 2000), may lead to a reduction of the overall inner iterations. Finally, we show that our proposal has some similarities with the Limited Memory Preconditioners (Gratton et al. in SIAM J. Optim. 21:912–935, 2011).

Keywords Large scale optimization · Nonconvex problems · Preconditioning · Krylov subspace methods · Newton–Krylov methods

G. Fasano
Dipartimento di Management, Università Ca' Foscari Venezia, San Giobbe, Cannareggio, 873, 30121 Venezia, Italy
e-mail: fasano@unive.it

M. Roma (✉)
Dipartimento di Ingegneria Informatica, Automatica e Gestionale "A. Ruberti", SAPIENZA, Università di Roma, via Ariosto, 25, 00185 Roma, Italy
e-mail: roma@dis.uniroma1.it

1 Introduction

In this paper we study a preconditioning strategy to be used for the efficient solution of large scale linear systems. As well known, there are many and different contexts of nonlinear optimization in which the iterative solution of sequences of linear systems is required. Some examples are Truncated Newton methods in unconstrained optimization, equality and/or inequality constrained problems, KKT systems, interior point methods, PDE constrained optimization and many others.

Krylov subspace methods are usually used for iteratively solving large scale linear systems, by means of "matrix-free" implementations. Examples of these methods are the Conjugate Gradient (CG) for symmetric positive definite systems, the Lanczos process (SYMMLQ and MINRES) for symmetric indefinite systems, GMRES, Bi-CGSTAB, QMR methods for unsymmetric systems.

In this work we consider symmetric indefinite linear systems, and we focus on preconditioners based on using Krylov subspace methods. In particular, our aim is to iteratively construct a preconditioner by using a decomposition of the system matrix (see, e.g. [10, 13, 27]), obtained as by product of the Krylov subspace method. Firstly, we state the least requirements that a Krylov subspace method must satisfy to be suited to this aim. Then, we define our preconditioner with reference to a general Krylov subspace method and prove theoretical properties for such a preconditioner. Finally, we focus on the CG method and the Krylov planar method FLR (see [7, 10]), which is an extension of the CG method, in order to build our preconditioner. We consider these algorithms since they both generate the conjugate directions we need; moreover, algorithm FLR may also prevent from the pivot breakdowns of the CG when the system matrix is not positive definite. Possible alternatives may be considered, suitably using the information provided by MINRES, in order to build our preconditioner. Nevertheless, within optimization frameworks, since the CG is computationally cheap, it is often used also in case of an indefinite system matrix, in place of SYMMLQ or MINRES. In this case, whenever the CG even gets stuck, standard procedures are usually adopted to recover the optimization iteration (see, e.g., [3, 4] and the implementation details in [18]). Finally, consider that the optimization framework where we embed our preconditioner aims at iteratively detecting minimizers of the objective function. This implies that, as described in the next paragraphs, we are going to solve eventually positive definite linear systems, where the CG is always well-posed.

We apply the latter CG-based methods in order to generate an approximation of the inverse of the system matrix; then, the latter is used for building a preconditioner. We show that the construction of such a preconditioner is obtained by storing a few vectors, without computing matrices products or matrices inverses. Actually, we assume that the entries of the system matrix are not known and the only information of the system matrix is gained by means of a routine, which provides the product of the matrix times a vector. As well known, this routine is usually assumed available 'for free', since it is required by any implementation of a Krylov subspace method.

We use our preconditioner within a nonlinear optimization framework, namely in solving nonconvex large scale unconstrained problems. In particular, we focus on the so called *Newton–Krylov methods*, also known as Truncated Newton methods (see

[23] for a survey). The latter methods are extensions of the Newton method, in which a Krylov subspace method is usually used for approximately computing the search direction. We strongly remark that in this context, all we need in order to compute our preconditioner is already available, as by product of the inner iterations performed for computing the search direction.

Notwithstanding Truncated Newton methods have been widely studied and extensively tested, it is largely recognized (see, e.g. [25]) that two key aspects for the overall efficiency of these methods can be still considered worthwhile to be investigated. The first one regards the definition and the use of a preconditioner which enables to accelerate the convergence. In fact, preconditioning is still considered an essential tool for achieving a good efficiency and robustness, especially in the large scale setting. The second one concerns the way to tackle nonconvex problems and hence to handle indefinite Newton's equations.

In this paper we try to address both the latter aspects at once, in the large scale setting. As regards the first issue, numerical results show that the use of our preconditioner often leads to a reduction of the overall number of inner iterations, in most of the test problems considered. As regards the possibility to handle nonconvex problems, unlike the CG, the planar method FRL can avoid pivot breakdowns in the indefinite case. Moreover, drawing our inspiration from [17], in computing the search direction using the preconditioned CG, we avoid the untimely termination of the inner iterations when negative curvatures are detected.

The preconditioner detailed in this paper has also some similarities with the LM Preconditioners, proposed in [16] by Gratton et al., as commented in the next section (Remark 2.1).

Finally, in the optimization framework we adopt, we propose an extension of the truncation rule by Nash–Sofer [24] to the indefinite case.

The paper is organized as follows: in Sect. 2, we consider the problem of constructing a preconditioner for an indefinite linear system, by using a general Krylov subspace method. In Sects. 3 and 4, we introduce our preconditioned CG. In Sects. 5 and 6, we address the Truncated Newton methods for unconstrained optimization and study our new preconditioned Truncated Newton method. Finally, in Sect. 7 we report the results of an extensive numerical testing, on all the large scale unconstrained test problems in CUTEr collection [15]. We show that the approach we propose in this paper is reliable and often enables a significant reduction of the number of inner iterations, in solving both convex and nonconvex problems. A section of possible alternatives and a section of conclusions complete the paper.

As regards the notations, for a $n \times n$ real matrix M we denote with $\Lambda(M)$ the spectrum of M; I_k is the identity matrix of order k and e_h is the h-th unit vector. For sake of clarity, if the context requires, we use $0_{m\times n}$ to denote a $m \times n$ null matrix. Finally, $\|\cdot\|$ indicates the Euclidean norm and for the set B the quantity $|B|$ represents its cardinality.

2 Iterative matrix decomposition and the new preconditioner

In this section we consider the solution of a large symmetric linear system, by using a Krylov subspace method. We describe the conditions which should be satisfied by

the Krylov subspace method, in order to iteratively obtaining a decomposition of the system matrix, for constructing the preconditioner we propose. To this aim, consider the *indefinite* linear system

$$Ax = b, \tag{2.1}$$

where $A \in \mathbb{R}^{n\times n}$ is *symmetric* and *nonsingular*, n is *large* and $b \in \mathbb{R}^n$, and consider any Krylov subspace method for the solution of such a system (see, e.g. [13]). The Lanczos process (SYMMLQ and MINRES) and the CG method are among the most popular. They are equivalent as long as the matrix A is positive definite, whereas the CG, though cheaper, may not cope with the indefinite case.

Our aim is to focus on Krylov subspace methods, in order to iteratively construct a preconditioner to be used in the solution of the system (2.1). The next Assumption 2.1 summarizes the requirements for the Krylov subspace methods we adopt. We suppose to apply any iterative Krylov subspace method and we assume that a finite number of steps, say $h \ll n$, is performed.

Assumption 2.1 *Let us consider the linear system* (2.1). *At the step* $h \geq 1$ *of the Krylov method the matrices* $R_h \in \mathbb{R}^{n\times h}$, $T_h \in \mathbb{R}^{h\times h}$, $L_h \in \mathbb{R}^{h\times h}$ *and* $B_h \in \mathbb{R}^{h\times h}$ *are generated, such that*

$$AR_h = R_h T_h + \rho_{h+1} u_{h+1} e_h^T, \tag{2.2}$$

$$T_h = L_h B_h L_h^T, \tag{2.3}$$

where

$R_h = (u_1 \cdots u_h)$, $u_i^T u_j = 0$, $\|u_i\| = 1$, $1 \leq i \neq j \leq h+1$,
T_h *is symmetric tridiagonal, irreducible and nonsingular,*
L_h *is lower triangular with a "simple pattern" for its entries,*
B_h *is a nonsingular block diagonal matrix, with* 1×1 *or* 2×2 *diagonal blocks.*

On the basis of the latter assumption, we can now define our preconditioner and show its properties. Let $Z \in \mathbb{R}^{n\times n}$ be a symmetric nonsingular matrix such that $Z = WDW^T$, where $W \in \mathbb{R}^{n\times n}$ is nonsingular and $D \in \mathbb{R}^{n\times n}$ is block diagonal with a "simple pattern", namely block diagonal with 1×1 or 2×2 blocks. Then, we define

$$|Z| = W|D|W^T$$

where $|D|$ is positive definite block diagonal with 1×1 or 2×2 blocks. In particular, whenever D is diagonal also $|D|$ is diagonal, with diagonal entries corresponding to the absolute value of the diagonal entries of D. When D includes a 2×2 diagonal block, then the latter is decomposed using the eigenvalues–eigenvectors factorization, replacing the two eigenvalues by the corresponding absolute values, see Sect. 2.2 of [11]. Now we can introduce the following matrix

$$M_h = \left(I_n - R_h R_h^T\right) + R_h |T_h| R_h^T, \tag{2.4}$$

where R_h and T_h satisfy relation (2.2) and $|T_h| = L_h|B_h|L_h^T$. Observe that $(I_n - R_h R_h^T)$ is a projector onto the subspace orthogonal to the range of the matrix R_h and $R_h|T_h|R_h^T w = 0$ for any w orthogonal to the range of R_h.

Theorem 2.1 *Consider any Krylov subspace method to solve* (2.1)*; suppose it performs $h \le n$ iterations and that Assumption* 2.1 *holds. Then we have*

(a) *M_h is symmetric and nonsingular*;
(b) *the inverse M_h^{-1} exists and is given by*

$$M_h^{-1} = \left(I_n - R_h R_h^T\right) + R_h|T_h|^{-1}R_h^T. \tag{2.5}$$

Moreover, if $h = n$, then $M_n^{-1} = R_n|T_n|^{-1}R_n^T = |A^{-1}|$;
(c) *M_h is positive definite and its spectrum $\Lambda(M_h)$ is given by*

$$\Lambda(M_h) = \Lambda\left(|T_h|\right) \cup \Lambda(I_{n-h}), \quad h = 1, \ldots, n-1;$$

(d1) *$\Lambda(M_h^{-1}A) \equiv \Lambda(AM_h^{-1})$ and contains at least $h-2$ eigenvalues in the set $\{-1, +1\}$, $h = 2, \ldots, n-1$;*
(d2) *if A is positive definite, then $\Lambda(M_h^{-1}A)$ contains at least $h-1$ eigenvalues equal to $+1$, $h = 1, \ldots, n-1$;*
(e) *if $h = n$, then $\Lambda(M_n) = \Lambda(|T_n|)$ and $\Lambda(M_n^{-1}A) = \Lambda(AM_n^{-1}) \subseteq \{-1, +1\}$.*

Proof As regards (a), the symmetry trivially follows from the symmetry of T_h. Moreover, the matrix $R_{n,h}$ exists such that $R_{n,h}^T R_{n,h} = I_{n-h}$ and the columns of the matrix $[R_h \mid R_{n,h}]$ are an orthogonal basis of $\mathbb{R}^n$. Thus, for any $v \in \mathbb{R}^n$ we can write $v = R_h v_1 + R_{n,h} v_2$, with $v_1 \in \mathbb{R}^h$ and $v_2 \in \mathbb{R}^{n-h}$. Now, to prove M_h is invertible, we show that $M_h v = 0$ implies $v = 0$. In fact,

$$M_h v = R_{n,h} v_2 + R_h|T_h|v_1 = [R_h|T_h| \mid R_{n,h}] \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = 0,$$

if and only if $(v_1^T \; v_2^T)^T = 0$, since from Assumption 2.1 the matrix $|T_h|$ is nonsingular.

As regards (b), recalling that $R_h^T R_h = I_h$ and $|T_h|$ is nonsingular from Assumption 2.1, a direct computation yields the result.

As concerns item (c), since $I_n - R_h R_h^T = R_{n,h} R_{n,h}^T$, we can write (2.4) as

$$M_h = [R_h \mid u_{h+1} \mid R_{n,h+1}] \left[\begin{array}{cc|c} |T_h| & 0 & \\ 0 & 1 & 0 \\ \hline 0 & & I_{n-(h+1)} \end{array}\right] \left[\begin{array}{c} R_h^T \\ \hline u_{h+1}^T \\ \hline R_{n,h+1}^T \end{array}\right], \quad h = 1, \ldots, n-1 \tag{2.6}$$

which gives the result, since T_h is irreducible and thus $|T_h|$ is positive definite.

Item (d1) may be proved by considering that we have $\Lambda(M_h^{-1}A) \equiv \Lambda(M_h^{-1/2}A \times M_h^{-1/2}) \equiv \Lambda(AM_h^{-1})$. Moreover, by (2.6), we have

$$M_h^{-1/2} = [R_h \mid u_{h+1} \mid R_{n,h+1}] \left[\begin{array}{c|c|c} |T_h|^{-1/2} & 0 & \\ \hline 0 & 1 & 0 \\ \hline 0 & & I_{n-(h+1)} \end{array}\right] \left[\begin{array}{c} R_h^T \\ \hline u_{h+1}^T \\ \hline R_{n,h+1}^T \end{array}\right],$$

and hence

$$\begin{aligned} M_h^{-1/2} A M_h^{-1/2} &= [R_h \mid u_{h+1} \mid R_{n,h+1}] \left[\begin{array}{c|c|c} |T_h|^{-1/2} & 0 & \\ \hline 0 & 1 & 0 \\ \hline 0 & & I_{n-(h+1)} \end{array}\right] \\ &\cdot \left[\begin{array}{c|c|c} T_h & \rho_{h+1} e_h & 0 \\ \hline \rho_{h+1} e_h^T & u_{h+1}^T A u_{h+1} & u_{h+1}^T A R_{n,h+1} \\ \hline 0 & R_{n,h+1}^T A u_{h+1} & R_{n,h+1}^T A R_{n,h+1} \end{array}\right] \\ &\cdot \left[\begin{array}{c|c|c} |T_h|^{-1/2} & 0 & \\ \hline 0 & 1 & 0 \\ \hline 0 & & I_{n-(h+1)} \end{array}\right] \left[\begin{array}{c} R_h^T \\ \hline u_{h+1}^T \\ \hline R_{n,h+1}^T \end{array}\right] \\ &= [R_h \mid u_{h+1} \mid R_{n,h+1}] \left[\begin{array}{c|c|c} |T_h|^{-1/2} T_h |T_h|^{-1/2} & \rho_{h+1} |T_h|^{-1/2} e_h & 0 \\ \hline \rho_{h+1} e_h^T |T_h|^{-1/2} & u_{h+1}^T A u_{h+1} & u_{h+1}^T A R_{n,h+1} \\ \hline 0 & R_{n,h+1}^T A u_{h+1} & R_{n,h+1}^T A R_{n,h+1} \end{array}\right] \left[\begin{array}{c} R_h^T \\ \hline u_{h+1}^T \\ \hline R_{n,h+1}^T \end{array}\right], \end{aligned}$$

where we used the fact that, by (2.2), we have $R_h^T A R_h = T_h$, $R_h^T A u_{h+1} = \rho_{h+1} e_h$ and

$$R_h^T A R_{n,h+1} = \left(R_{n,h+1}^T A R_h\right)^T = \left(R_{n,h+1}^T \left(R_h T_h + \rho_{h+1} u_{h+1} e_h^T\right)\right)^T = 0.$$

Now, any eigenvalue of $M_h^{-1/2} A M_h^{-1/2}$ solves the *characteristic equation*

$$\det \left[\begin{array}{c|c|c} |T_h|^{-1/2} T_h |T_h|^{-1/2} - \lambda I_h & \rho_{h+1} |T_h|^{-1/2} e_h & 0_{h \times [n-(h+1)]} \\ \hline \rho_{h+1} e_h^T |T_h|^{-1/2} & u_{h+1}^T A u_{h+1} - \lambda & u_{h+1}^T A R_{n,h+1} \\ \hline 0_{[n-(h+1)] \times h} & R_{n,h+1}^T A u_{h+1} & R_{n,h+1}^T A R_{n,h+1} - \lambda I_{n-(h+1)} \end{array}\right] = 0. \tag{2.7}$$

For sake of brevity, let us denote the latter equation by

$$\det\left[\begin{array}{c|c}\bar{A} & \bar{B}\\ \hline \bar{B}^T & \bar{C}\end{array}\right]=0 \tag{2.8}$$

where $\bar{A}\in\mathbb{R}^{(h+1)\times(h+1)}$, $\bar{B}\in\mathbb{R}^{(h+1)\times[n-(h+1)]}$ and $\bar{C}\in\mathbb{R}^{[n-(h+1)]\times[n-(h+1)]}$. Now, to find the n real zeroes of (2.8) (i.e. the eigenvalues of the matrix), we distinguish two cases: the case (1) in which we prove that at least $(h-2)$ eigenvalues are in the set $\{-1,+1\}$, and the case (2) in which we prove that possibly, among the remaining $n-(h-2)$ eigenvalues there might be further values in the set $\{-1,+1\}$.

The case (1) corresponds to assume $\bar{C}$ in (2.8) non singular. By using Proposition 2.8.4 (or equivalently Fact 2.17.3) in [2], we have

$$\det\left[\begin{array}{c|c}\bar{A} & \bar{B}\\ \hline \bar{B}^T & \bar{C}\end{array}\right]=\det\left[\begin{array}{c|c}\bar{A}-\bar{B}\bar{C}^{-1}\bar{B}^T & 0\\ \hline 0 & \bar{C}\end{array}\right]=\det\left[\bar{A}-\bar{B}\bar{C}^{-1}\bar{B}^T\right]\det[\bar{C}].$$

The matrix $\bar{A}-\bar{B}\bar{C}^{-1}\bar{B}^T$ is known as the *Schur complement* of $\bar{C}$ in $\left[\begin{array}{c|c}\bar{A} & \bar{B}\\ \hline \bar{B}^T & \bar{C}\end{array}\right]$. Now, note that it results

$$\begin{aligned}\left[\bar{A}-\bar{B}\bar{C}^{-1}\bar{B}^T\right]&=\left[\left[\begin{array}{c|c}|T_h|^{-1/2}T_h|T_h|^{-1/2}-\lambda I_h & \rho_{h+1}|T_h|^{-1/2}e_h\\ \hline \rho_{h+1}e_h^T|T_h|^{-1/2} & u_{h+1}^TAu_{h+1}-\lambda\end{array}\right]-\left[\begin{array}{c|c}0_{h\times h} & 0_h\\ \hline 0_h^T & \xi\end{array}\right]\right]\\ &=\left[\begin{array}{c|c}|T_h|^{-1/2}T_h|T_h|^{-1/2}-\lambda I_h & \rho_{h+1}|T_h|^{-1/2}e_h\\ \hline \rho_{h+1}e_h^T|T_h|^{-1/2} & u_{h+1}^TAu_{h+1}-\lambda-\xi\end{array}\right],\end{aligned}$$

for some $\xi\in\mathbb{R}$. Therefore, since λ is such that $\det(\bar{C})\neq 0$, λ is an eigenvalue of $M_h^{-1/2}AM_h^{-1/2}$ if and only if

$$\det\left[\begin{array}{c|c}|T_h|^{-1/2}T_h|T_h|^{-1/2}-\lambda I_h & \rho_{h+1}|T_h|^{-1/2}e_h\\ \hline \rho_{h+1}e_h^T|T_h|^{-1/2} & u_{h+1}^TAu_{h+1}-\lambda-\xi\end{array}\right]=0. \tag{2.9}$$

As a consequence, if we denote

$$G_{h+1}=\left[\begin{array}{c|c}|T_h|^{-1/2}T_h|T_h|^{-1/2} & \rho_{h+1}|T_h|^{-1/2}e_h\\ \hline \rho_{h+1}e_h^T|T_h|^{-1/2} & u_{h+1}^TAu_{h+1}-\xi\end{array}\right]\in\mathbb{R}^{(h+1)\times(h+1)}$$

and we order the eigenvalues of G_{h+1} as

$$\lambda_{h+1}(G_{h+1})\le\lambda_h(G_{h+1})\le\cdots\le\lambda_1(G_{h+1}),$$

by the Cauchy interlacing Theorem (see e.g. [2] Lemma 8.4.4), it results

$$\lambda_{i+1}(G_{h+1})\le\lambda_i\left(|T_h|^{-1/2}T_h|T_h|^{-1/2}\right)\le\lambda_i(G_{h+1}),\quad i=1,\dots,h,$$

that is

$$\begin{aligned}\lambda_{h+1}(G_{h+1})\le\lambda_h\left(|T_h|^{-1/2}T_h|T_h|^{-1/2}\right)\le\lambda_h(G_{h+1})\le\lambda_{h-1}\left(|T_h|^{-1/2}T_h|T_h|^{-1/2}\right)\\ \le\cdots\le\lambda_2(G_{h+1})\le\lambda_1\left(|T_h|^{-1/2}T_h|T_h|^{-1/2}\right)\le\lambda_1(G_{h+1}),\end{aligned} \tag{2.10}$$

where

$$\lambda_h\left(|T_h|^{-1/2}T_h|T_h|^{-1/2}\right) \le \lambda_{h-1}\left(|T_h|^{-1/2}T_h|T_h|^{-1/2}\right) \le \cdots \le \lambda_1\left(|T_h|^{-1/2}T_h|T_h|^{-1/2}\right)$$

are the ordered eigenvalues of $|T_h|^{-1/2}T_h|T_h|^{-1/2}$.

Now, consider that $\Lambda(|T_h|^{-1/2}T_h|T_h|^{-1/2}) \equiv \Lambda(|T_h|^{-1}T_h)$ and $|T_h|^{-1}T_h = L_h^{-T}|B_h|^{-1}L_h^{-1}L_hB_hL_h^T = L_h^{-T}\hat{I}_hL_h^T$, where $\hat{I}_h$ is block diagonal with eigenvalues σ_i, $i = 1, \ldots, h$ and $\sigma_i \in \{-1, +1\}$. Thus, if (μ, v) is an eigenpair of $|T_h|^{-1}T_h$, then $\hat{I}_h(L_h^Tv) = \mu(L_h^Tv)$, so that μ is an eigenvalue of $\hat{I}_h$, i.e. $\mu \in \{-1, 1\}$.

Therefore, if the matrix $\hat{I}_h$ has σ^+ eigenvalues equal to $+1$ and σ^- eigenvalues equal to -1 (with $h = \sigma^+ + \sigma^-$), it immediately follows by (2.10) that the matrix G_{h+1} has $(\sigma^- - 1)$ eigenvalues equal to -1, $(\sigma^+ - 1)$ eigenvalues equal to $+1$ and one eigenvalue in the interval $[-1, 1]$. This proves by (2.9) that at least $(\sigma^+ - 1) + (\sigma^- - 1) = h - 2$ eigenvalues of $M_h^{-1/2}AM_h^{-1/2}$ are in the set $\{-1, +1\}$.

The case (2) corresponds to assume $\bar{C}$ in (2.8) singular. In this case, we give some information about the remaining $n - (h - 2)$ eigenvalues of the matrix in (2.8). On this purpose we distinguish two subcases: (2a) λ is such that $\bar{C}$ is singular and $\bar{A}$ is nonsingular, (2b) λ is such that $\bar{C}$ is singular and $\bar{A}$ is singular too. In case (2a) it means that λ is not an eigenvalue of

$$\left[\begin{array}{c|c} |T_h|^{-1/2}T_h|T_h|^{-1/2} & \rho_{h+1}|T_h|^{-1/2}e_h \\ \hline \rho_{h+1}e_h^T|T_h|^{-1/2} & u_{h+1}^TAu_{h+1} \end{array}\right],$$

so that, since $\Lambda(|T_h|^{-1/2}T_h|T_h|^{-1/2}) \subseteq \{-1, +1\}$, by the Cauchy interlacing Theorem, λ is possibly not in the set $\{-1, +1\}$. Thus, in this subcase λ is possibly among the $n - (h - 2)$ zeroes of (2.8) which are not in the set $\{-1, +1\}$.

On the other hand, in case (2b), observe preliminarily that $u_{h+1}^TAR_{n,h+1} \in \mathbb{R}^{1\times[n-(h+1)]}$ and by (2.2) we obtain (for some $t_{h+1,h+1} \in \mathbb{R}$)

$$A(R_h \mid u_{h+1}) = (R_h \mid u_{h+1})\left(\begin{array}{c|c} T_h & \rho_{h+1}e_h \\ \hline \rho_{h+1}e_h^T & t_{h+1,h+1} \end{array}\right) + \rho_{h+2}u_{h+2}e_{h+1}^T,$$

so that $u_{h+1}^TAR_{n,h+1} = (\rho_{h+2}\ 0\ \cdots\ 0)$ in (2.7). Now, computing the determinant in (2.7) with respect to the elements of the $(h+1)$-th row, we obtain that it is the sum of $h + 2$ terms. Considering that $\det(R_{n,h+1}^TAR_{n,h+1} - \lambda I_{n-(h+1)}) = \det(\bar{C}) = 0$, after a few computation the determinant in (2.7) reduces to ('*' indicates 'non relevant' entries)

$$(-1)^{2h+3}\rho_{h+2}\det\left[\begin{array}{c|c} |T_h|^{-1/2}T_h|T_h|^{-1/2} - \lambda I_h \quad \rho_{h+1}|T_h|^{-1/2}e_h & 0_{h\times[n-(h+2)]} \\ \hline 0\cdots\cdots\cdots\cdots 0 \qquad \rho_{h+2} & *\cdots\cdots\cdots\cdots * \\ \hline 0_{[n-(h+2)]\times(h+1)} & \begin{array}{cccc} (*-\lambda) & * & \cdots & * \\ * & \ddots & & \vdots \\ \vdots & & \ddots & * \\ * & \cdots & * & (*-\lambda) \end{array} \end{array}\right]$$

$$+ (-1)^{2(h+1)}\left(u_{h+1}^TAu_{h+1} - \lambda\right)\det\left[\begin{array}{c|c} |T_h|^{-1/2}T_h|T_h|^{-1/2} - \lambda I_h & 0_{h\times[n-(h+1)]} \\ \hline 0_{[n-(h+1)]\times h} & R_{n,h+1}^TAR_{n,h+1} - \lambda I_{n-(h+1)} \end{array}\right].$$

Recalling that $|T_h|^{-1/2}T_h|T_h|^{-1/2}$ has σ^+ eigenvalues equal to $+1$ and σ^- eigenvalues equal to -1 (with $h=\sigma^+ + \sigma^-$), the submatrix

$$\begin{pmatrix} |T_h|^{-1/2}T_h|T_h|^{-1/2} - \lambda I_h & \rho_{h+1}|T_h|^{-1/2}e_h \\ 0 \cdots\cdots\cdots\cdots\cdots 0 & \rho_{h+2} \end{pmatrix}$$

in the previous expression has a determinant with at least h zeroes (values of λ) in the set $\{-1,+1\}$. Thus, in the case (2b) at least h eigenvalues of the matrix in (2.8) are in the set $\{-1,+1\}$. In the end, considering the cases (1), (2a) and (2b), among the n zeroes of (2.8) at least $h-2$ are in the set $\{-1,+1\}$, which completes the proof of (d1).

Item (d2) follows reasoning as in item (d1) and considering that $\Lambda(|T_h|^{-1/2}T_h|T_h|^{-1/2}) \subseteq \{+1\}$.

Item (e) immediately follows from (2.2), (2.4) and the fact that R_n is orthogonal. □

Observe that the matrix M_h^{-1}aims to provide an approximation of the inverse of the matrix A. In particular, whenever A is positive definite and the iterative process performs $h=n$ iterations (see item (b)), then $M_n^{-1}=A^{-1}$.

We want to adopt the matrix M_h^{-1}, with $h \ll n$, as preconditioner, as we will describe in the sequel. We observe that M_h^{-1} is therefore a by product of the Krylov subspace method. Note that this is not the first attempt in the literature to use a preconditioner of this form. In fact, a similar approach has been considered also in the context of GMRES methods via the Arnoldi process (see [1, 6, 19]). However, our result is more general with respect to the result reported in [1], where it is required that the columns of the matrix R_h span an invariant subspace of A.

Remark 2.1 It is worth to highlight that in case $\rho_{k+1}u_{h+1}=0$ in (2.2) and T_k is positive definite (A positive definite implies that T_k is positive definite, but the converse is not necessarily true), then our preconditioner (2.5) coincides with a Limited Memory Preconditioner (LMP) [16, 28]. However, our proposal can cope with the indefinite case, too. Moreover, whenever $\rho_{k+1}u_{h+1} \neq 0$ then the two preconditioners do not coincide and represent different preconditioning strategies.

Furthermore, since the columns of R_h are orthonormal, they can be seen as approximate eigenvectors of the matrix A, so that the theory developed in [12, 16] may be suitably applied.

3 Computing a new preconditioner via the Conjugate Gradient algorithm

As remarked at the end of the previous section, the approach of the papers [1, 6, 19] to compute a preconditioner is based on the use of the Arnoldi process, to generate a basis of orthogonal vectors. In case of symmetric matrix A, the latter approach yields the use of the Lanczos process. Here, the basis of orthogonal vectors is given by the (normalized) Lanczos vectors $Q_h=(q_1,\ldots,q_h)$ and the iterative process yields the relation

$$AQ_h = Q_hT_h + \delta_h q_h e_h^T, \tag{3.1}$$

where T_h is a tridiagonal and nonsingular matrix and $\delta_h \in \mathbb{R}$ (see, e.g. [27]). The formula (3.1) similarly to (2.2) leads to the expression (see (2.4))

$$\tilde{M}_h = \left(I_n - Q_h Q_h^T\right) + Q_h T_h Q_h^T, \tag{3.2}$$

whose inverse can be used as a possible preconditioner. Note that in case the symmetric matrix A is not positive definite, $\tilde{M}_h^{-1}$ could be indefinite, so that it cannot be directly used as a preconditioner with some Krylov subspace methods (e.g. the CG algorithm and the Lanczos process).

By using the well know relations between the Lanczos process and the CG method (see, e.g. [27]), and considering the motivations in Sect. 1, we propose instead to adopt the CG method in order to iteratively build a preconditioning matrix of the form (2.4), which is also positive definite and such that M_h^{-1} is easily computed. For the sake of clarity we report a scheme of the CG algorithm.

Algorithm CG

Step 1: Set $k = 1$; $d_1 = 0$; $r_1 = b$.
If a stopping rule is satisfied then STOP, else compute $p_1 = r_1$.

Step k: Set $d_{k+1} = d_k + a_k p_k$, $r_{k+1} = r_k - a_k A p_k$, where $a_k = \dfrac{\|r_k\|^2}{p_k^T A p_k}$.
If a stopping rule is satisfied then STOP else compute $p_{k+1} = r_{k+1} + \beta_k p_k$, where $\beta_k = \dfrac{\|r_{k+1}\|^2}{\|r_k\|^2}$.
Set $k = k + 1$ and go to ***Step k***.

After $h \leq n$ steps of the Algorithm CG the following matrices can be defined (see also [27]):

$$R_h = \left(\frac{r_1}{\|r_1\|} \cdots \frac{r_h}{\|r_h\|}\right) \in \mathbb{R}^{n\times h}, \qquad P_h = \left(\frac{p_1}{\|r_1\|} \cdots \frac{p_h}{\|r_h\|}\right) \in \mathbb{R}^{n\times h},$$

and the following relations hold

$$R_h^T R_h = I_h \tag{3.3}$$

$$A R_h = R_h T_h + \rho_{h+1} r_{h+1} e_h^T, \quad \text{for some } \rho_{h+1} \neq 0, \tag{3.4}$$

where T_h is tridiagonal and is given by

$$T_h = L_h D_h L_h^T, \tag{3.5}$$

with

$$L_h = \begin{pmatrix} 1 & & & & \\ -\sqrt{\beta_1} & 1 & & & \\ & -\sqrt{\beta_2} & \ddots & & \\ & & \ddots & \ddots & \\ & & & -\sqrt{\beta_{h-1}} & 1 \end{pmatrix} \tag{3.6}$$

and

$$D_h = \operatorname{diag}\left\{ \frac{1}{a_1}, \frac{1}{a_2}, \ldots, \frac{1}{a_h} \right\}.$$

If $r_i \neq 0$, $i = 1, \ldots, h$ and $r_{h+1} = 0$ then d_{h+1} solves (2.1), i.e. $Ad_{h+1} = b$ so that no preconditioner is needed to be built. Otherwise, whenever $r_{h+1} \neq 0$ then (3.4) holds and from (3.3) we have

$$R_h^T A R_h = T_h. \tag{3.7}$$

Moreover, since A is nonsingular, T_h is nonsingular too, and irreducible. In addition we have

$$P_h L_h^T = R_h. \tag{3.8}$$

Therefore, by means of the Algorithm CG the matrices R_h, T_h, L_h and B_h satisfying Assumption 2.1 can be iteratively constructed. Therefore, the preconditioner M_h^{-1} obtained by using expressions (3.8) and (3.5) in (2.5) can be rewritten as

$$M_h^{-1} = \left(I_n - P_h L_h^T L_h P_h^T\right) + P_h L_h^T |T_h|^{-1} L_h P_h^T \tag{3.9}$$

$$= \left(I_n - P_h L_h^T L_h P_h^T\right) + P_h |D_h|^{-1} P_h^T, \tag{3.10}$$

where

$$|T_h| = L_h |D_h| L_h^T \quad \text{and} \quad |D_h| = \operatorname{diag}\left\{ \frac{1}{|a_1|}, \frac{1}{|a_2|}, \ldots, \frac{1}{|a_h|} \right\}.$$

It is fundamental to notice that the construction of the preconditioner by the Algorithm CG *does not require* any matrix inversion (apart from $|D_h|^{-1}$ which is a diagonal matrix). In particular, the storage of the matrices P_h (in place of the matrix R_h) and L_h suffices to compute the preconditioner M_h^{-1}. Finally, observe that though P_h must be fully stored, however L_h is sparse and has a very simple structure. In the context of preconditioning indefinite systems, the use of the absolute values of the entries of D_h in order to handle indefiniteness has been already proposed in [11].

4 Computational cost of preconditioning

In this section we analyze the overall computational cost of introducing our preconditioner in the Algorithm CG. For the sake of completeness we now report the scheme

of Prec-CG(M), which is a standard preconditioned CG scheme, using M as preconditioning matrix.

Algorithm Prec-CG(M)

Step 1: Set $k=1$; $d_1=0$; $r_1=b$; $M^{-1}\in\mathbb{R}^{n\times n}$; $z_1=M^{-1}r_1$.
If a stopping rule is satisfied then STOP, else compute $p_1=z_1$.

Step k: Set $d_{k+1}=d_k+\tilde{a}_k p_k$, $r_{k+1}=r_k-\tilde{a}_k A p_k$, where $\tilde{a}_k=\dfrac{r_k^T z_k}{p_k^T A p_k}$.
If a stopping rule is satisfied then STOP
else compute $z_{k+1}=M^{-1}r_{k+1}$, $p_{k+1}=z_{k+1}+\tilde{\beta}_k p_k$,

where $\tilde{\beta}_k=\dfrac{r_{k+1}^T z_{k+1}}{r_k^T z_k}$.
Set $k=k+1$ and go to ***Step k***.

Observe that, as well known, none of the coefficients in Algorithm Prec-CG(M) depends *explicitly* on the preconditioner M^{-1}, and the only effect of the preconditioner is through the product $M^{-1}r$, where r is an n-real vector.

Let us now consider the Algorithm Prec-CG(M_h), which uses the preconditioner M_h^{-1} defined in (2.5). We have in particular

$$M_h^{-1}r=\left[\left(I_n-R_hR_h^T\right)+R_h|T_h|^{-1}R_h^T\right]r=r+R_h\left(|T_h|^{-1}-I_n\right)R_h^T r,$$

so that the overall computational cost $\mathcal{C}(M_h^{-1}r)$ to compute the product $M_h^{-1}r$ is given by

$$\mathcal{C}\left(M_h^{-1}r\right)=hn+\mathcal{C}\left(|T_h|^{-1}\right)+nh=2hn+\mathcal{C}\left(|T_h|^{-1}\right)\le 2hn+O\left(h^3\right),\quad(4.1)$$

where $\mathcal{C}(|T_h|^{-1})$ includes the computational cost of calculating $|T_h|^{-1}$ and the cost to compute the product $|T_h|^{-1}v$, with $v\in\mathbb{R}^h$. In addition, we can take advantage of the structure of T_h in (3.5), in order to estimate $\mathcal{C}(|T_h|^{-1})$ in (4.1) more precisely. Indeed, after a short computation, we can set

$$|T_h|^{-1}=L_h^{-T}|D_h|^{-1}L_h^{-1}$$

and L_h^{-1} can be easily computed. In fact, we have the following result.

Proposition 4.1 *Let $L_h\in\mathbb{R}^{h\times h}$ in (3.6) be nonsingular, with the structure*

$$L_h=\left(\begin{array}{c|c}L_{h-1} & 0\\ \hline -\sqrt{\beta_{h-1}}e_{h-1}^T & 1\end{array}\right),$$

where $\beta_{h-1}>0$. Then we have

$$L_h^{-1}=\left(\begin{array}{c|c}I_{h-1} & 0\\ \hline \sqrt{\beta_{h-1}}e_{h-1}^T & 1\end{array}\right)\left(\begin{array}{c|c}L_{h-1}^{-1} & 0\\ \hline 0 & 1\end{array}\right),\quad(4.2)$$

and given the matrix L_{h-1}^{-1}, the computation of L_h^{-1} requires $(h-1)$ additional multiplications.

Proof By direct computation we obtain relation (4.2). Moreover, from (4.2) we see that the product $\sqrt{\beta_{h-1}} e_{h-1}^T L_{h-1}^{-1}$ requires just $(h-1)$ additional multiplications. □

From Proposition 4.1 we deduce that the overall cost $\mathcal{C}(|T_h|^{-1})$ in (4.1) is given by the sum of the following two costs:

- the cost to compute L_h^{-1}, which is (from the iterative application of Proposition 4.1)

$$(h-1) + (h-2) + \cdots + 1 = \frac{h}{2}(h-1),$$

- the cost to perform the computation $|T_h|^{-1} v$, with $v \in \mathbb{R}^h$, which is

$$2\big[(h-1) + (h-2) + \cdots + 1\big] + h = h^2$$

(Observe that by Proposition 4.1 the matrix L_h^{-1} is a lower triangular matrix and not just a lower bidiagonal matrix).

Therefore, the overall computational cost $\mathcal{C}(M_h^{-1} r)$ in (4.1) to perform the computation of $M_h^{-1} r$ is given by

$$\mathcal{C}\big(M_h^{-1} r\big) = 2hn + \mathcal{C}\big(|T_h|^{-1}\big) = 2hn + \frac{3}{2} h \left(h - \frac{1}{3}\right), \tag{4.3}$$

which is $\approx 2hn$ when $h \ll n$.

Note that the parameter h (in this paper) and the parameter m (in [21]) play the same role: they define the 'memory' of the preconditioner, i.e. equivalently the number of vectors to be stored. So that, by using the same 'memory' (i.e. $h = m$), the computational cost $\approx 2hn$ of our proposal is approximately one half of the computational cost $4mn$ of the proposal in [21].

5 Preconditioned Truncated Newton methods

In the previous sections we described how to iteratively construct the preconditioner defined in (2.5). Even if this preconditioner can be exploited in different contexts involving the solution of large scale linear systems, our main interest is to fruitfully use it within the framework of Truncated Newton methods, for large scale optimization. Of course, both unconstrained and constrained nonlinear optimization problems are of interest. Here we investigate its use within unconstrained optimization, namely for solving the problem

$$\min_{x \in \mathbb{R}^n} f(x),$$

where $f : \mathbb{R}^n \to \mathbb{R}$ is twice continuously differentiable and n is large. In particular, we aim at defining some preconditioning strategies which lead to an improvement of the overall efficiency of the method.

The structure of any (linesearch-based) Truncated Newton method for unconstrained optimization is well known (see e.g. [23]). It is based on two nested loops: the iterations of the Newton method (*outer iterations*) and the iterations of the solvers used, at each outer iteration j, to approximately solving the Newton system (*inner iterations*)

$$H_j\, s = -g_j, \tag{5.1}$$

where $H_j = \nabla^2 f(x_j)$ and $g_j = \nabla f(x_j)$. Then, a linesearch procedure is performed along the approximate solution $s^{(j)}$ of (5.1). Thus, since a sequence of linear systems must be solved, in many cases, it is crucial to have at one's disposal a preconditioning strategy, which should enable a considerable computational saving in terms of number of inner iterations. This motivates the fact that to define general purpose preconditioning strategies, within Truncated Newton methods, is usually still considered one the main issues which is worthwhile investigating, especially in dealing with large scale problems. As well known, the main difficulty in this context relies on the fact that, in the large scale setting, the Hessian matrix cannot be stored or handled. Hence, any Truncated Newton implementation gains information on the Hessian matrix by means of a routine, which provides the product of the Hessian matrix times a vector. This prevents from using any preconditioner based on the knowledge of the actual elements of the Hessian matrix. Unfortunately, few preconditioners have been proposed so far, which do not require the full Hessian matrix. To our knowledge, the first proposal of such a preconditioner is in [22]. Here a diagonal scaling which uses a diagonal approximation of the Hessian obtained by means of BFGS updating is introduced. Afterwards, in [20, 21] the automatic preconditioner PREQN based on m-step limited memory quasi-Newton updating has been proposed, without requiring the knowledge of the Hessian matrix. Noticeable is also the generalization of the quasi-Newton approach proposed in [16], with the class of LM Preconditioners, whose construction involves only few vectors and their product with the matrix to precondition. A preconditioning strategy based on a diagonal scaling, which only gains the information of the Hessian matrix by means of the product of the Hessian times a vector, has been proposed in [26]. The other preconditioners proposed, usually require the knowledge of the entries of the Hessian matrix and hence may not be suited for large scale setting.

The preconditioner defined in Sect. 2, actually is generated during the iterations of the iterative Krylov solver, gaining the information on the Hessian matrix as by product. Moreover, as already noticed in Sect. 3, its application only requires to store h real vectors, i.e. the columns of matrix R_h.

5.1 Our preconditioning strategy

Now we describe the preconditioning strategy we adopt, at each outer iteration, which is applied in two sequential stages. In the first one we perform a very small number of iterations of the Algorithm CG, to possibly construct the preconditioner. Then, in

case a sufficient number of CG iterations (say $h = h_{\max} \ll n$) are performed to build the preconditioner, and the approximate solution $d_{h_{\max}}$ is computed, we continue CG iterations applying Prec-CG($M_{h_{\max}}$) with $d_1 = 0$. As we show in the proof of Proposition 5.2 (see formula (A.1) in the Appendix), by setting $d_1 = 0$ in Prec-CG($M_{h_{\max}}$) the first preconditioned residual $M_{h_{\max}}^{-1} r_1$ does not need to be computed, since it is directly obtained using the $h_{\max}$ directions given so far by the CG. In fact, it results

$$z_1 = M_{h_{\max}}^{-1} r_1 = \sum_{i=1}^{h_{\max}} |a_i| p_i$$

and the first direction of Prec-CG($M_{h_{\max}}$) is $p_1 = z_1$. In this regard, the information collected by the CG is completely retrieved when Prec-CG($M_{h_{\max}}$) is started.

In case no preconditioner was built ($h < h_{\max}$), we consider the current solution obtained by the Algorithm CG. In the following scheme we summarize the latter strategy.

1. Set the integer $h_{\max}$ (say $h_{\max} \leq 15$).
2. Perform the Algorithm CG. Terminate the CG after h iterations, computing the approximate solution d_h, if either a stopping criterion is satisfied or $h = h_{\max}$.
   - If $h = h_{\max}$ store P_h, L_h, D_h and continue the CG iterations performing the *preconditioned* Algorithm Prec-CG($M_{h_{\max}}$) with $d_1 = 0$.
   - If $h < h_{\max}$ consider the current solution d_h obtained by the Algorithm CG.

Observe that, if during the first h iterations of the Algorithm CG the termination criterion is satisfied before performing $h_{\max}$ iterations, of course the inner iterations are stopped and no preconditioner is considered.

Remark 5.1 As we will show in Proposition 5.2, this two stages strategy, in principle, does not imply a "waste of time". Furthermore, note that the idea of rerunning an iterative Krylov subspace method in order to retrieve information on the current problem was already adopted in [14].

Remark 5.2 One of the most relevant point of such a strategy is the fact that, at the j-th outer iteration (i.e. when we are solving the j-th system of the sequence (5.1)), the preconditioner is computed on the basis of the current Hessian matrix H_j. On the contrary, the strategies usually adopted (see, e.g. [21]) compute the preconditioner, to be used in the j-th outer iteration, during the inner CG iterations performed at the previous outer iteration. This means that the preconditioner is computed on the basis of H_{j-1}, which could be a serious drawback in case the Hessian matrix should drastically change from x_{j-1} to x_j.

5.2 The stopping criterion to compute the search direction

In this section we describe how we compute the search direction $s^{(j)}$ which approximately solves (5.1), at outer iteration j of the Truncated Newton method. We refer

to the application of the Algorithm Prec-CG(M_h) to solve the Newton system (5.1), assuming that the preconditioner M_h^{-1} was already computed. *We recall that we are dealing with the indefinite case, so that particular safeguard is needed in computing the search direction whenever a negative curvature direction is encountered*, namely a direction p such that $p^T H_j p < 0$. In Sect. 6 we are going to give more details on the indefinite case, by introducing another Krylov method that we can adopt alternatively to the CG.

Here, getting our inspiration from [17], we adopt the following strategy: we do not terminate the i-th CG inner iteration of either Algorithm CG or the Algorithm Prec-CG(M_h), whenever a negative curvature direction is encountered, provided that

$$\left| p_i^T H_j p_i \right| > \epsilon \| p_i \|^2,$$

where $\epsilon > 0$ is a suitable parameter. Observe that, if $p_i^T H_j p_i < 0$ for some i, the current approximate solution $s^{(j)}$, generated at the i-th CG iteration, could be no longer a descent direction for the quadratic model

$$Q_j(s) = \frac{1}{2} s^T H_j s + g_j^T s.$$

To overcome this drawback, which arises in dealing with nonconvex problems, we proceed by suitably adapting the computation of the search direction. Hereafter, for the sake of simplicity, when there is no ambiguity we omit the subscript j and use $Q(s)$ in place of $Q_j(s)$.

Let us now consider the k-th iteration of Prec-CG(M_h); let $\epsilon > 0$ and define the following index sets

$$I_k^+ = \left\{ i \in \{1, \ldots, k\} : p_i^T H_j p_i > \epsilon \| p_i \|^2 \right\},$$
$$I_k^- = \left\{ i \in \{1, \ldots, k\} : p_i^T H_j p_i < -\epsilon \| p_i \|^2 \right\},$$

where $|I_k^+| + |I_k^-| = k$, and the following vectors

$$s_k^P = \sum_{i \in I_k^+} \tilde{a}_i p_i = \sum_{i \in I_k^+} \frac{r_i^T M_h^{-1} r_i}{p_i^T H_j p_i} p_i,$$

$$s_k^N = -\sum_{i \in I_k^-} \tilde{a}_i p_i = -\sum_{i \in I_k^-} \frac{r_i M_h^{-1} r_i}{p_i^T H_j p_i} p_i.$$

The direction s_k^P can be viewed as the minimizer of $Q(s)$ over the subspace 'span$_{i \in I_k^+}\{p_i\}$'. Conversely s_k^N is a negative curvature direction. Then, at each preconditioned CG iteration we define the vector $s_k = s_k^P + s_k^N$ (see [17]). Observe that the vector s_k, in general, might not be an approximate solution of (5.1). However, with the latter choice we guarantee the monotonic decrease of $\{Q(s_k)\}$ as k increases. We remark that $s_k^P - s_k^N$, is possibly not a descent direction for $Q_j(s)$ when H_j is indefinite. Hence, suitable strategies have been proposed in the literature to cope with

the latter issue (see e.g. [3]). We adopt the approach in [17], since the direction s_k has several other appealing properties within optimization frameworks. The following result holds.

Proposition 5.1 *Suppose that the Algorithm* Prec-CG(M_h) *is applied for solving the system* (5.1). *At each iteration* k *of* Prec-CG(M_h) *consider the following vector*

$$s_k = s_k^P + s_k^N. \tag{5.2}$$

Then the sequence $\{Q(s_k)\}_{k=1,2,\ldots}$ *is strictly decreasing, i.e.*

$$Q(s_{k+1}) < Q(s_k), \quad k = 1, 2, \ldots.$$

Proof By definition, we have

$$s_k = s_k^P + s_k^N = \sum_{i=1}^{k} |\tilde{a}_i| p_i = s_{k-1} + |\tilde{a}_k| p_k.$$

Now, using the fact that $r_1 = -g_j$, $r_1^T p_k = r_k^T M_h^{-1} r_k$, $p_k^T H_j p_i = 0$, $i \le k-1$, and $s_{k-1} = s_{k-2} + |\tilde{a}_{k-1}| p_{k-1}$ we obtain

$$\begin{aligned}
Q(s_k) &= \frac{1}{2}[s_{k-1} + |\tilde{a}_k| p_k]^T H_j [s_{k-1} + |\tilde{a}_k| p_k] + g_j^T [s_{k-1} + |\tilde{a}_k| p_k] \\
&= \frac{1}{2}\big[s_{k-1}^T H_j s_{k-1} + \tilde{a}_k^2 p_k^T H_j p_k\big] + |\tilde{a}_k| p_k^T H_j s_{k-1} + g_j^T s_{k-1} + |\tilde{a}_k| g_j^T p_k \\
&= \frac{1}{2}\big[s_{k-1}^T H_j s_{k-1} + \tilde{a}_k^2 p_k^T H_j p_k\big] + g_j^T s_{k-1} + |\tilde{a}_k| g_j^T p_k \\
&= Q(s_{k-1}) + \frac{1}{2}\operatorname{sgn}\big(p_k^T H_j p_k\big) \frac{(r_k^T M_h^{-1} r_k)^2}{|p_k^T H_j p_k|} - \frac{r_k^T M_h^{-1} r_k}{|p_k^T H_j p_k|} r_1^T p_k \\
&= Q(s_{k-1}) + \left(\frac{1}{2}\operatorname{sgn}\big(p_k^T H_j p_k\big) - 1\right) \frac{(r_k^T M_h^{-1} r_k)^2}{|p_k^T H_j p_k|} < Q(s_{k-1}).
\end{aligned}$$

□

The position (5.2) and Proposition 5.1 play a fundamental rule for the choice of the stopping criterion of Prec-CG(M_h), in the Truncated Newton scheme. In fact, from Proposition 5.1 we can use the standard truncation rule based on the reduction of the quadratic model [24]. Therefore, at the current outer iteration k, the CG iterations are terminated if

$$\frac{Q(s_k) - Q(s_{k-1})}{Q(s_k)/k} \le \alpha, \tag{5.3}$$

where α is a suited parameter. We highlight that the stopping rule (5.3) was also adopted in [21] to apply the preconditioner PREQN. Observe that setting $M_h = I$, i.e. in the unpreconditioned case, s_k coincides with the choice of the Newton-type direction in [17]. As well known in the literature of unconstrained optimization (see

e.g. [23, 24]), though not immediately intuitive, the criterion (5.3) is equivalent to the stopping criterion based on the relative residual, not requiring the combined use of the two criteria. Note that Proposition 5.1, ensuring the monotonic decrease of the quadratic model, enables to extend the use of the stopping criterion (5.3) to the nonconvex case. Hence, the criterion (5.3) is used alternatively to the residual-based standard criterion $\|r_j\|/\|g_j\| \leq \eta_j$, where $\eta_j \to 0$ as $j \to \infty$.

5.3 Preconditioned Newton step

Now we aim at proving some properties which arise from applying the preconditioner M_h^{-1} as in (3.9)–(3.10). We recall that, according to Sect. 5.1, our strategy is based on continuing the iterative process (after computing the preconditioner in the first $h_{\max}$ unpreconditioned CG iterations) with the starting point $d_1 = 0$. In the following proposition we prove that, even if $d_1 = 0$ in both Algorithm CG and Algorithm Prec-CG(M_h) (i.e. a simple restart), one iteration of Algorithm Prec-CG(M_h) improves the quadratic model of the function not less than h iterations of Algorithm CG. To this aim, consider the vector

$$s_2^{PR} = s_1 + |\tilde{a}_1| p_1 = |\tilde{a}_1| M_h^{-1} r_1, \tag{5.4}$$

obtained after one iteration of Algorithm Prec-CG(M_h).

Proposition 5.2 *Let M_h be the matrix in (3.9)–(3.10), computed after h iterations of the Algorithm CG. Let $p_1, \ldots, p_h$ be the conjugate directions generated, with $p_\ell^T H_j p_m = 0$, $1 \leq \ell \neq m \leq h$, and $p_1 = -g_j$. Then we have*

$$Q\left(s_2^{PR}\right) \leq Q(s_h), \tag{5.5}$$

where s_2^{PR} is given in (5.4), and s_h is defined in (5.2) with $M_h = I$.

Proof See the Appendix. □

This proposition clearly shows that, the use of our preconditioning strategy (see Sect. 5.1) does not imply an additional effort to decrease the quadratic model. In fact, in one iteration of the Algorithm Prec-CG(M_h) the quadratic model is improved not less than the improvement obtained after h iterations of the Algorithm CG. Moreover, using relation (A.1) we actually do not need to compute the first direction p_1 in Algorithm Prec-CG(M_h).

6 On computing the preconditioner in the indefinite case

In the previous sections we detailed how to build our preconditioner M_h^{-1} after h iterations of the Algorithm CG. We can easily extend our theory to the use of the Conjugate Gradient-type method FLR introduced in [7]. Unlike the CG, it copes with the indefinite case too, without the risk of pivot breakdown. It is an iterative method

for solving indefinite linear systems, and is a modification of the standard CG algorithm.

Further details on the Algorithm FLR can be found in [7–9]; here we simply consider some relevant results, which can be used in order to obtain relations (2.2)–(2.3).

Unlike the Algorithm CG, if a pivot breakdown occurs at Step i (i.e. if $p_i^T A p_i \approx 0$), the Algorithm FLR, similarly to the Lanczos process, does not stop untimely.

In the remainder of this section we give evidence that the Algorithm FLR can also provide the matrices R_h, T_h, L_h and B_h in the Assumption 2.1, so that (2.2) and (2.3) hold. To this aim we report the results in [10] and a few simple consequences.

Theorem 6.1 *Consider the Algorithm* FLR *in* [7] *to solve the linear system* $Ax = b$, *where* A *is symmetric, indefinite and nonsingular. Suppose the Algorithm* FLR *performs* $h \le n$ *steps. Then the following matrices can be defined*: $\hat{L}_h \in \mathbb{R}^{h\times h}$ *a nonsingular lower triangular matrix*, $\hat{B}_h \in \mathbb{R}^{h\times h}$ *a* 2×2 *block diagonal matrix*, $\hat{R}_h \in \mathbb{R}^{n\times h}$ *whose columns are orthogonal, and* $\hat{T}_h \in \mathbb{R}^{h\times h}$ *an irreducible symmetric tridiagonal and nonsingular matrix. The following relations hold*:

$$A\hat{R}_h = \hat{R}_h \hat{T}_h + \hat{\rho}_{h+1}\hat{u}_{h+1}e_h^T, \quad \hat{\rho}_{h+1} \in \mathbb{R},$$
$$\hat{T}_h = \hat{L}_h \hat{B}_h \hat{L}_h^T.$$

Moreover, the following matrix can be defined

$$\hat{M}_h = \left(I - \hat{R}_h \hat{R}_h^T\right) + \hat{R}_h |\hat{T}_h| \hat{R}_h^T, \tag{6.1}$$

where $|\hat{T}_h| = \hat{L}_h |\hat{B}_h| \hat{L}_h^T$, *and* $|\hat{B}_h| = \mathrm{diag}_{i\le h}\{V_i |\hat{D}_i| V_i^T\}$, $\hat{D}_i$ *is a* 2×2 *or* 1×1 *diagonal matrix and* V_i *is a* 2×2 *or* 1×1 *orthogonal matrix. The properties* (*a*), (*b*), (*c*), (*d*), (*e*) *of Theorem* 2.1, *still hold replacing* M_h, R_h, T_h, L_h *respectively with* $\hat{M}_h$, $\hat{R}_h$, $\hat{T}_h$, $\hat{L}_h$.

$\hat{M}_h$ can be used as preconditioning matrix in the indefinite case, similarly to M_h. It is fundamental to notice that, as for the Algorithm CG, the construction of the preconditioner $\hat{M}_h^{-1}$ by the Algorithm FLR *does not require* any matrix inversion (apart from $|\hat{B}_h|^{-1}$ which is a 2×2 block diagonal matrix). In fact, $\hat{M}_h^{-1}$ can be rewritten in the same form of (3.10), replacing $|D_h|^{-1}$ with $|\hat{B}_h|^{-1}$. Finally, observe that $\hat{L}_h$ is sparse and has a very simple structure (see [10]).

All the considerations we proved in obtaining the preconditioner by means of the Algorithm CG, in the previous sections, still hold also in case Algorithm FLR is used. For the sake of clarity and in order to preserve the simplicity of reading, we decided to detail in the previous sections the theory by using the Algorithm CG and not the Algorithm FLR.

7 Numerical results

In this section we report the results of a preliminary numerical investigation obtained by embedding the new preconditioner in a linesearch based Truncated Newton

method for unconstrained optimization. In particular, we performed a standard implementation of a Truncated Newton method which uses the Algorithm CG as a tool for constructing the preconditioner, and by adopting a standard monotone linesearch procedure. The stopping criterion adopted for the CG inner iterations is (5.3) with the standard choice $\alpha = 1/2$ (see [24]). Then, the Algorithm Prec-CG is used as preconditioned scheme to solve the Newton equation, according to the strategy described in Sect. 5.1.

The aim of our numerical investigation is, firstly, to assess if the preconditioning approach proposed is reliable. In particular, we adopt the same perspective of [21], that is the key point is to check if the use of the preconditioner leads to a computational saving in terms of the overall number of inner iterations. Indeed, quoting from [21], "*Our main interest in these results lies in the number of CG iterations*". This is due to the fact that each inner iteration requires a matrix–vector product of the Hessian times a vector, which usually represents the main computational effort in large scale setting. Nevertheless, we also report the CPU time needed to solve each problem.

As test problems we used all the unconstrained large problems included in the CUTEr collection [15]. Whenever a problem has variable dimension, we used the problem with two different dimensions (usually 1000 and 10000 variables). As result we have an overall test set of 112 problems. *In performing a comparison among different algorithms, we include all those problems where all the algorithms converge to the same stationary point*. Furthermore, on the guideline of [21], for each outer iteration we allowed at most $2n$ inner iterations, where n is the dimension of the problem.

All the algorithms were coded in FORTRAN 90 compiled with GFortran under Linux UBUNTU 9.10, 64 bit. All the runs were performed on a PC Intel Core i7 870 at 2.93 GHz with 8 Gb RAM. Moreover, a failure was declared if the number of function evaluations/inner iterations/outer iterations exceeded the value 100000, or the CPU time exceeded 900 seconds.

As regards the parameter $h_{\max}$, we experienced different values ranging from 5 to 15: the values $h_{\max} = 5$ and $h_{\max} = 7$ seem to provide a good trade-off between the computational burden required to compute the preconditioner and its effectiveness. The latter conclusion seems to match with the results in [21], where a satisfactory approximation of the inverse Hessian matrix requires a similar number of vector pairs as memory of the quasi-Newton update. We report here only the results obtained with the value $h_{\max} = 7$. For sake of completeness, we report the complete results of our extensive numerical testing in terms of number of iterations (ITER), number of function evaluations (FUNCT), number of CG-inner iterations (CG-it), CPU time in seconds (TIME), along with the function value. In particular, in Tables 1 and 2 we report the results obtained by using the Algorithm Prec-CG and in Tables 3 and 4 the results obtained by using the (unpreconditioned) Algorithm CG.

Then, in order to perform a comparison with the use of preconditioner PREQN introduced in [21], in Table 5 and Table 6 we report the complete results obtained by using the preconditioner PREQN.

To our knowledge, PREQN is a well recognized approximate inverse preconditioner for large scale problems, which is both iteratively built and is for a general

Table 1 (*Part I*) Complete results relative to 112 test problems, using the Algorithm Prec-CG

Problem	n	Iter	Funct	CG-it	F. value	Time
ARWHEAD	1000	34	364	37	0.000000D+00	0.02
ARWHEAD	10000	10	102	11	1.332134D−11	0.05
BDQRTIC	1000	46	293	84	3.983818D+03	0.05
BDQRTIC	10000	121	1217	204	4.003431D+04	0.89
BROYDN7D	1000	458	1788	1676	3.823419D+00	0.85
BROYDN7D	10000	845	4138	3190	3.644797D+03	15.48
BRYBND	1000	20	64	26	6.709348D−12	0.02
BRYBND	10000	20	64	26	6.226697D−12	0.11
CHAINWOO	1000	166	335	554	1.000001D+00	0.18
CHAINWOO	10000	267	797	764	1.000001D+00	1.86
COSINE	1000	22	64	40	−9.990000D+02	0.03
COSINE	10000	19	65	32	−9.999000D+03	0.08
CRAGGLVY	1000	49	216	94	3.364231D+02	0.06
CRAGGLVY	10000	116	776	173	3.377956D+03	0.89
CURLY10	1000	11227	11514	37918	−1.003163D+05	14.35
CURLY10	10000	59999	61159	203494	−1.003163D+06	640.95
CURLY20	1000	13468	13650	45616	−1.001379D+05	22.71
CURLY20	10000	–	–	–	–	>900
CURLY30	1000	17007	17198	56998	−1.003163D+05	37.89
DIXMAANA	1500	8	13	9	1.000000D+00	0.01
DIXMAANA	3000	8	14	8	1.000000D+00	0.01
DIXMAANB	1500	5	10	6	1.000000D+00	0.01
DIXMAANB	3000	5	10	6	1.000000D+00	0.00
DIXMAANC	1500	5	11	6	1.000000D+00	0.01
DIXMAANC	3000	5	11	6	1.000000D+00	0.00
DIXMAAND	1500	5	8	5	1.000000D+00	0.01
DIXMAAND	3000	5	8	5	1.000000D+00	0.01
DIXMAANE	1500	76	79	168	1.000000D+00	0.09
DIXMAANE	3000	114	117	258	1.000000D+00	0.25
DIXMAANF	1500	52	57	136	1.000000D+00	0.09
DIXMAANF	3000	54	59	143	1.000000D+00	0.19
DIXMAANG	1500	43	86	121	1.000000D+00	0.09
DIXMAANG	3000	74	142	257	1.000000D+00	0.23

continued on next page

Table 1 (*Continued*)

Problem	n	Iter	Funct	CG-it	F. value	Time
DIXMAANH	1500	54	56	134	1.000000D+00	0.11
DIXMAANH	3000	74	76	209	1.000000D+00	0.27
DIXMAANI	1500	215	218	693	1.000001D+00	0.39
DIXMAANI	3000	235	238	714	1.000003D+00	0.76
DIXMAANJ	1500	64	117	164	1.086254D+00	0.08
DIXMAANJ	3000	82	175	245	1.165166D+00	0.24
DIXMAANK	1500	60	74	173	1.000000D+00	0.15
DIXMAANK	3000	62	75	199	1.000000D+00	0.21
DIXMAANL	1500	53	55	130	1.000001D+00	0.09
DIXMAANL	3000	55	57	149	1.000000D+00	0.19
DQDRTIC	1000	33	274	34	7.461713D−26	0.03
DQDRTIC	10000	102	868	103	2.426640D−27	0.44
DQRTIC	1000	22	81	40	2.784985D−02	0.02
DQRTIC	10000	31	111	60	4.932478D−01	0.06
EDENSCH	1000	21	89	27	6.003285D+03	0.02
EDENSCH	10000	18	85	23	6.000328D+04	0.09
ENGVAL1	1000	11	34	16	1.108195D+03	0.01
ENGVAL1	10000	12	36	19	1.109926D+04	0.05
FLETCBV2	1000	1	1	0	−5.013384D−01	0.00
FLETCBV2	10000	1	1	0	−5.001341D−01	0.00
FLETCBV3	1000	9	9	22	−8.470408D+04	0.02
FLETCBV3	10000	112	112	136	−2.534893D+10	0.38
FLETCHCR	1000	52	344	87	6.453457D−07	0.04

unconstrained problem. It is based on limited memory quasi-Newton updating. In particular, in [21] at the outer iteration j a new preconditioner M_j is generated, using the information available when solving the system $H_{j-1}d = -g_{j-1}$, i.e. at the $(j-1)$-th outer iteration. In particular, m correction pairs of L-BFGS updating are stored and used to define the new preconditioner.

In comparing the two preconditioning strategies, first observe that we are dealing with large scale problems. Moreover, as long as $h_{\max} \leq n^{1/3}$ (we adopted $h_{\max} \in [5, 15]$ in our testing and $h_{\max} = 7$ for the reported results) the cost for computing the preconditioned residual $M_h^{-1}r$ by Prec-CG(M_h) is approximately given by $\approx 2hn$ (see (4.3)), and it is significantly smaller than the corresponding computational cost in applying the preconditioner proposed in [21]. In fact, the latter requires $4mn$ floating point operations, where m plays the same role of h, for computing the preconditioned

Table 2 (*Part II*) Complete results relative to 112 test problems, using the Algorithm Prec-CG

Problem	n	Iter	Funct	CG-it	F. value	Time
FLETCHCR	10000	117	1085	143	2.745120D−06	0.54
FMINSURF	1024	93	207	288	1.000000D+00	0.17
FMINSURF	5625	235	710	680	1.000000D+00	2.65
FREUROTH	1000	38	300	50	1.214697D+05	0.04
FREUROTH	10000	107	1052	119	1.216521D+06	0.63
GENHUMPS	1000	902	3559	3278	2.468237D−12	1.67
GENHUMPS	10000	393	1456	1223	6.205140D−13	5.68
GENROSE	1000	851	2601	2615	1.000000D+00	0.98
GENROSE	10000	8093	24306	24914	1.000000D+00	81.45
LIARWHD	1000	42	251	61	8.352643D−19	0.03
LIARWHD	10000	112	1107	133	1.455368D−20	0.55
MOREBV	1000	8	8	28	2.148161D−08	0.02
MOREBV	10000	2	2	7	2.428066D−09	0.01
MSQRTALS	1024	1346	1575	4682	3.616388D−04	23.73
MSQRTBLS	1024	1371	1608	4675	5.288396D−04	25.02
NONCVXUN	1000	796	1342	2740	2.331277D+03	1.37
NONCVXUN	10000	2692	12058	10957	2.333085D+04	42.24
NONCVXU2	1000	479	1247	1666	2.317124D+03	0.81
NONCVXU2	10000	2163	10064	8793	2.319231D+04	33.20
NONDIA	1000	22	256	27	6.680969D−21	0.04
NONDIA	10000	78	1515	82	5.507180D−13	0.64
NONDQUAR	1000	45	111	111	1.425631D−04	0.04
NONDQUAR	10000	46	175	98	3.744353D−04	0.18
PENALTY1	1000	31	32	59	9.686175D−03	0.02
PENALTY1	10000	54	81	121	9.900151D−02	0.20
POWELLSG	1000	46	257	86	1.992056D−08	0.05
POWELLSG	10000	114	783	151	7.735314D−08	0.25
POWER	1000	65	189	142	5.912729D−09	0.08
POWER	10000	233	891	559	9.025072D−09	1.08
QUARTC	1000	22	81	40	2.784985D−02	0.02
QUARTC	10000	31	111	60	4.932478D−01	0.07
SCHMVETT	1000	14	35	37	−2.994000D+03	0.03
SCHMVETT	10000	19	69	38	−2.999400D+04	0.20

continued on next page

Table 2 (*Continued*)

Problem	n	Iter	Funct	CG-it	F. value	Time
SINQUAD	1000	37	310	49	−2.942505D+05	0.04
SINQUAD	10000	104	1517	111	−2.642315D+07	1.01
SPARSINE	1000	2860	3280	9616	9.042411D−03	7.81
SPARSINE	10000	–	–	–	–	>900
SPARSQUR	1000	22	66	34	6.266490D−09	0.02
SPARSQUR	10000	22	67	39	1.069594D−08	0.18
SPMSRTLS	1000	62	218	132	6.219291D+00	0.09
SPMSRTLS	10000	6181	6604	20578	5.713622D+01	119.40
SROSENBR	1000	35	309	40	2.842418D−22	0.02
SROSENBR	10000	104	920	108	9.421397D−12	0.24
TESTQUAD	1000	12401	12950	42766	1.636783D−05	12.76
TOINTGSS	1000	2	3	1	1.001002D+01	0.00
TOINTGSS	10000	2	3	1	1.000100D+01	0.00
TQUARTIC	1000	21	185	27	3.767509D−10	0.02
TQUARTIC	10000	14	144	18	1.145916D−11	0.05
TRIDIA	1000	244	635	738	7.979032D−06	0.27
TRIDIA	10000	1764	3391	5764	6.817977D−06	10.49
VARDIM	1000	37	37	72	1.058565D−20	0.03
VARDIM	10000	54	298	99	4.475275D−18	0.17
VAREIGVL	1000	24	49	74	2.351034D−08	0.04
VAREIGVL	10000	21	179	22	3.924839D−16	0.15
WOODS	1000	64	377	141	3.857513D−08	0.06
WOODS	10000	139	1095	223	5.031534D−08	0.51

residual at each inner iteration, and the parameter m is set by the user in the range [4, 16].

In separate figures we also include the results of these comparisons by using performance profiles [5]. In particular, in Fig. 1 the Prec-CG and the (unpreconditioned) CG algorithms are compared in terms of inner iterations.

Fig. 2 reports the comparison (in terms of inner iterations) among the Prec-CG, the PREQN and the (unpreconditioned) CG algorithms. In order to carry on a fair comparison, in this latter figure we include only those test problems where negative curvature directions are not detected. This is due to the fact that in a standard use of PREQN, a termination of the CG iterations is expected whenever a negative curvature is encountered (see Sect. 3 of [21]).

Table 3 (*Part I*) Complete results relative to 112 test problems, using the Algorithm CG

Problem	n	Iter	Funct	CG-it	F. value	Time
ARWHEAD	1000	34	364	37	0.000000D+00	0.04
ARWHEAD	10000	10	102	11	1.332134D−11	0.05
BDQRTIC	1000	46	293	84	3.983818D+03	0.05
BDQRTIC	10000	121	1217	204	4.003431D+04	0.94
BROYDN7D	1000	449	1923	2499	3.823419D+00	0.87
BROYDN7D	10000	853	3917	4394	3.613248D+03	14.72
BRYBND	1000	20	64	26	6.709348D−12	0.02
BRYBND	10000	20	64	26	6.226697D−12	0.12
CHAINWOO	1000	217	395	824	1.000001D+00	0.24
CHAINWOO	10000	278	814	872	1.000001D+00	1.92
COSINE	1000	22	64	40	−9.990000D+02	0.03
COSINE	10000	19	65	32	−9.999000D+03	0.08
CRAGGLVY	1000	45	212	102	3.364231D+02	0.05
CRAGGLVY	10000	115	775	177	3.377956D+03	0.86
CURLY10	1000	130	417	5651	−1.003163D+05	0.63
CURLY10	10000	135	1295	55801	−1.003163D+06	58.52
CURLY20	1000	202	405	6083	−1.001379D+05	1.15
CURLY20	10000	162	994	64081	−1.001313D+06	108.23
CURLY30	1000	322	520	6544	−1.003163D+05	1.76
DIXMAANA	1500	8	13	9	1.000000D+00	0.01
DIXMAANA	3000	8	14	8	1.000000D+00	0.00
DIXMAANB	1500	5	10	6	1.000000D+00	0.01
DIXMAANB	3000	5	10	6	1.000000D+00	0.00
DIXMAANC	1500	5	11	6	1.000000D+00	0.01
DIXMAANC	3000	5	11	6	1.000000D+00	0.00
DIXMAAND	1500	5	8	5	1.000000D+00	0.00
DIXMAAND	3000	5	8	5	1.000000D+00	0.01
DIXMAANE	1500	63	66	204	1.000000D+00	0.07
DIXMAANE	3000	88	91	277	1.000000D+00	0.15
DIXMAANF	1500	33	38	198	1.000000D+00	0.06
DIXMAANF	3000	32	37	238	1.000000D+00	0.13
DIXMAANG	1500	41	80	223	1.000000D+00	0.07
DIXMAANG	3000	65	133	400	1.000000D+00	0.26

continued on next page

Table 3 (*Continued*)

Problem	n	Iter	Funct	CG-it	F. value	Time
DIXMAANH	1500	26	28	194	1.000000D+00	0.07
DIXMAANH	3000	28	30	256	1.000000D+00	0.13
DIXMAANI	1500	60	63	1997	1.000000D+00	0.40
DIXMAANI	3000	83	86	2585	1.000000D+00	1.09
DIXMAANJ	1500	58	118	213	1.089260D+00	0.08
DIXMAANJ	3000	63	135	291	1.176995D+00	0.16
DIXMAANK	1500	24	38	335	1.000000D+00	0.10
DIXMAANK	3000	28	41	333	1.000000D+00	0.16
DIXMAANL	1500	34	36	1648	1.000000D+00	0.40
DIXMAANL	3000	30	32	282	1.000000D+00	0.15
DQDRTIC	1000	33	274	34	7.461713D−26	0.03
DQDRTIC	10000	102	868	103	2.426640D−27	0.43
DQRTIC	1000	22	81	40	2.784985D−02	0.01
DQRTIC	10000	31	111	60	4.932478D−01	0.06
EDENSCH	1000	21	89	27	6.003285D+03	0.02
EDENSCH	10000	18	85	23	6.000328D+04	0.07
ENGVAL1	1000	11	34	16	1.108195D+03	0.01
ENGVAL1	10000	12	36	19	1.109926D+04	0.04
FLETCBV2	1000	1	1	0	−5.013384D−01	0.00
FLETCBV2	10000	1	1	0	−5.001341D−01	0.00
FLETCBV3	1000	11	11	33	−4.962265D+04	0.02
FLETCBV3	10000	200	200	487	−3.886849D+09	1.14
FLETCHCR	1000	47	339	101	6.067863D−06	0.04

From Tables 1–6 we can observe that on test problems where either Prec-CG or PREQN take a long run, PREQN seems more efficient. The latter fact is possibly the consequence of the choices in the optimization framework, and considering that when PREQN is used the computation of inner iterations is stopped when a negative curvature is encountered.[1] However, it can be observed from Figs. 1 and 2 that our approach allows on average a significant reduction of the number of the inner iterations, proving that the choice $h_{\max} = 7$ can be efficient. Thus, in the few $h_{\max}$ CG

[1] We remark, indeed, that the numerical results reported in [21] are pretty different from the results using PREQN in our Truncated Newton scheme, proving that the two optimization frameworks are likely different, and not immediately comparable.

Table 4 (*Part II*) Complete results relative to 112 test problems, using the Algorithm CG

Problem	n	Iter	Funct	CG-it	F. value	Time
FLETCHCR	10000	113	1082	163	7.578535D−07	0.51
FMINSURF	1024	32	88	1193	1.000000D+00	0.27
FMINSURF	5625	29	122	5062	1.000000D+00	5.32
FREUROTH	1000	38	300	50	1.214697D+05	0.05
FREUROTH	10000	107	1052	119	1.216521D+06	0.62
GENHUMPS	1000	625	1966	3463	4.356186D−14	1.05
GENHUMPS	10000	396	1299	1552	6.559308D−15	5.23
GENROSE	1000	836	2976	6203	1.000000D+00	1.07
GENROSE	10000	8113	28446	62729	1.000000D+00	84.28
LIARWHD	1000	42	251	61	8.352643D−19	0.04
LIARWHD	10000	112	1107	133	1.455368D−20	0.49
MOREBV	1000	6	6	70	9.088088D−09	0.02
MOREBV	10000	2	2	7	2.428066D−09	0.01
MSQRTALS	1024	153	522	4133	6.034837D−05	8.76
MSQRTBLS	1024	172	556	4375	3.642336D−07	9.31
NONCVXUN	1000	180	664	6040	2.325913D+03	1.08
NONCVXUN	10000	2173	10978	16270	2.323860D+04	40.39
NONCVXU2	1000	191	878	2028	2.317579D+03	0.46
NONCVXU2	10000	1869	9496	11198	2.316937D+04	30.32
NONDIA	1000	22	256	27	6.680969D−21	0.04
NONDIA	10000	78	1515	82	5.507180D−13	0.59
NONDQUAR	1000	43	112	215	1.135243D−04	0.05
NONDQUAR	10000	50	182	208	1.745194D−04	0.19
PENALTY1	1000	31	32	59	9.686175D−03	0.03
PENALTY1	10000	54	81	121	9.900151D−02	0.20
POWELLSG	1000	46	257	86	1.992056D−08	0.03
POWELLSG	10000	114	783	151	7.735314D−08	0.23
POWER	1000	56	180	221	2.472989D−09	0.05
POWER	10000	139	797	683	2.426355D−09	0.57
QUARTC	1000	22	81	40	2.784985D−02	0.01
QUARTC	10000	31	111	60	4.932478D−01	0.06
SCHMVETT	1000	14	35	37	−2.994000D+03	0.02
SCHMVETT	10000	19	69	38	−2.999400D+04	0.20

continued on next page

Table 4 (*Continued*)

Problem	n	Iter	Funct	CG-it	F. value	Time
SINQUAD	1000	37	310	49	−2.942505D+05	0.04
SINQUAD	10000	104	1517	111	−2.642315D+07	0.98
SPARSINE	1000	108	583	3027	2.543028D−04	0.93
SPARSINE	10000	465	2362	57180	3.845129D−03	193.31
SPARSQUR	1000	22	66	34	6.266490D−09	0.02
SPARSQUR	10000	22	67	39	1.069594D−08	0.17
SPMSRTLS	1000	56	219	211	6.219291D+00	0.07
SPMSRTLS	10000	936	1389	10134	5.703552D+01	24.31
SROSENBR	1000	35	309	40	2.842418D−22	0.03
SROSENBR	10000	104	920	108	9.421397D−12	0.23
TESTQUAD	1000	135	684	2057	2.339186D−06	0.17
TOINTGSS	1000	2	3	1	1.001002D+01	0.00
TOINTGSS	10000	2	3	1	1.000100D+01	0.01
TQUARTIC	1000	21	185	27	3.767509D−10	0.02
TQUARTIC	10000	14	144	18	1.145916D−11	0.04
TRIDIA	1000	57	448	470	7.356860D−07	0.06
TRIDIA	10000	162	1789	2238	6.273872D−08	1.66
VARDIM	1000	37	37	72	1.058565D−20	0.03
VARDIM	10000	54	298	99	4.475275D−18	0.18
VAREIGVL	1000	20	45	167	3.903597D−10	0.04
VAREIGVL	10000	21	179	22	3.924839D−16	0.17
WOODS	1000	64	377	141	3.857513D−08	0.04
WOODS	10000	139	1095	223	5.031534D−08	0.51

iterations, used for constructing our preconditioner at the j-th outer iteration, we possibly span the subspace generated by "significant" eigenvectors of the Hessian matrix H_j. We observe that our approach may be a winning strategy when dealing with large scale problems. Indeed, here the main purpose is likely to reduce the overall number of inner iterations, rather than accurately solving Newton's equation.

Though from Figs. 1 and 2 Prec-CG seems efficient in terms of the number of inner iterations, it is important to note that over the test problems our preconditioned algorithm leads to 2 additional failures, with respect to the CG and PREQN. As regards a comparison in terms of CPU time, we do not report the relative performance profiles, since the results obtained by the algorithms are very similar and the profiles do not give any evidence of the comparisons.

Table 5 (*Part I*) Complete results relative to 112 test problems, using the preconditioner PREQN

Problem	n	Iter	Funct	CG-it	F. value	Time
ARWHEAD	1000	34	364	37	0.000000D+00	0.03
ARWHEAD	10000	10	102	11	1.332134D−11	0.05
BDQRTIC	1000	45	292	75	3.983818D+03	0.04
BDQRTIC	10000	118	1215	154	4.003431D+04	0.77
BROYDN7D	1000	416	1407	2404	3.823419D+00	0.82
BROYDN7D	10000	956	3437	5159	3.732349D+03	16.85
BRYBND	1000	20	64	26	2.077977D−12	0.02
BRYBND	10000	20	64	26	1.952957D−12	0.11
CHAINWOO	1000	83	247	261	1.000000D+00	0.11
CHAINWOO	10000	154	677	331	1.000000D+00	0.94
COSINE	1000	22	62	36	−9.990000D+02	0.03
COSINE	10000	20	64	30	−9.999000D+03	0.08
CRAGGLVY	1000	47	214	99	3.364231D+02	0.06
CRAGGLVY	10000	114	774	156	3.377956D+03	0.86
CURLY10	1000	96	385	5335	−1.003163D+05	0.76
CURLY10	10000	134	1294	37434	−1.003163D+06	51.57
CURLY20	1000	171	364	5355	−1.001379D+05	1.11
CURLY20	10000	178	1003	42661	−1.001313D+06	84.31
CURLY30	1000	203	401	4667	−1.003163D+05	1.27
DIXMAANA	1500	8	13	9	1.000000D+00	0.01
DIXMAANA	3000	8	14	8	1.000000D+00	0.01
DIXMAANB	1500	5	10	6	1.000000D+00	0.01
DIXMAANB	3000	5	10	6	1.000000D+00	0.01
DIXMAANC	1500	5	11	6	1.000000D+00	0.01
DIXMAANC	3000	5	11	6	1.000000D+00	0.01
DIXMAAND	1500	5	8	5	1.000000D+00	0.00
DIXMAAND	3000	5	8	5	1.000000D+00	0.01
DIXMAANE	1500	65	68	167	1.000000D+00	0.08
DIXMAANE	3000	86	89	229	1.000000D+00	0.19
DIXMAANF	1500	33	38	130	1.000000D+00	0.05
DIXMAANF	3000	35	40	198	1.000000D+00	0.12
DIXMAANG	1500	47	95	280	1.000000D+00	0.12
DIXMAANG	3000	75	145	439	1.000000D+00	0.30

continued on next page

Table 5 (*Continued*)

Problem	*n*	Iter	Funct	CG-it	F. value	Time
DIXMAANH	1500	27	29	144	1.000000D+00	0.04
DIXMAANH	3000	31	33	186	1.000000D+00	0.12
DIXMAANI	1500	67	70	2260	1.000000D+00	0.61
DIXMAANI	3000	88	91	2746	1.000000D+00	1.39
DIXMAANJ	1500	67	106	220	1.086254D+00	0.11
DIXMAANJ	3000	62	113	272	1.183102D+00	0.19
DIXMAANK	1500	27	41	307	1.000000D+00	0.10
DIXMAANK	3000	29	42	320	1.000000D+00	0.20
DIXMAANL	1500	31	33	1326	1.000000D+00	0.38
DIXMAANL	3000	30	32	265	1.000000D+00	0.17
DQDRTIC	1000	33	274	34	7.461713D−26	0.03
DQDRTIC	10000	102	868	103	2.426640D−27	0.48
DQRTIC	1000	26	100	38	1.292595D−01	0.02
DQRTIC	10000	31	126	48	3.659146D+00	0.07
EDENSCH	1000	20	88	26	6.003285D+03	0.02
EDENSCH	10000	18	85	23	6.000328D+04	0.08
ENGVAL1	1000	11	34	16	1.108195D+03	0.01
ENGVAL1	10000	12	36	19	1.109926D+04	0.05
FLETCBV2	1000	1	1	0	−5.013384D−01	0.00
FLETCBV2	10000	1	1	0	−5.001341D−01	0.00
FLETCBV3	1000	144	144	150	−5.665012D+04	0.15
FLETCBV3	10000	99118	99118	99120	−3.668911D+09	482.26
FLETCHCR	1000	44	335	74	5.472471D−09	0.03

Finally, for a complete comparison between Prec-CG and PREQN some challenging applications should be considered, where possibly the role of negative curvatures is essential for the overall progress of optimization scheme.

8 An alternative preconditioning technique

As we remarked in the previous sections, the main purpose of our preconditioning strategy relies on using information from the *current outer iteration* of the Truncated Newton scheme adopted. This approach seems theoretically more promising than the standard approach currently used in the literature. Our choice avoids to delay the use of the current information to subsequent outer iterations. Indeed, as we already

Table 6 (*Part II*) Complete results relative to 112 test problems, using the preconditioner PREQN

Problem	n	Iter	Funct	CG-it	F. value	Time
FLETCHCR	10000	111	1079	134	2.871836D−08	0.49
FMINSURF	1024	46	193	1105	1.000000D+00	0.26
FMINSURF	5625	59	264	3083	1.000000D+00	3.63
FREUROTH	1000	37	299	46	1.214697D+05	0.03
FREUROTH	10000	106	1051	115	1.216521D+06	0.62
GENHUMPS	1000	1234	2931	3919	1.776020D−14	1.94
GENHUMPS	10000	568	1554	1565	4.201989D−11	6.63
GENROSE	1000	632	1263	3245	1.000000D+00	0.75
GENROSE	10000	5785	9849	31127	1.000000D+00	52.79
LIARWHD	1000	42	251	55	8.352643D−19	0.04
LIARWHD	10000	112	1107	125	1.455368D−20	0.49
MOREBV	1000	7	7	69	8.980894D−09	0.03
MOREBV	10000	2	2	7	2.428066D−09	0.01
MSQRTALS	1024	153	403	4197	3.239409D−08	9.17
MSQRTBLS	1024	144	392	2929	4.512623D−08	6.46
NONCVXUN	1000	136	407	3686	2.321100D+03	0.82
NONCVXUN	10000	635	2126	6510	2.325976D+04	15.55
NONCVXU2	1000	122	354	1328	2.316824D+03	0.35
NONCVXU2	10000	461	1522	3755	2.316903D+04	9.32
NONDIA	1000	22	256	27	6.680969D−21	0.00
NONDIA	10000	68	1305	71	2.373690D−17	0.51
NONDQUAR	1000	59	161	276	4.714600D−05	0.05
NONDQUAR	10000	51	204	184	1.738501D−04	0.22
PENALTY1	1000	31	35	50	9.686175D−03	0.04
PENALTY1	10000	54	81	115	9.900151D−02	0.22
POWELLSG	1000	46	257	70	3.067232D−08	0.04
POWELLSG	10000	114	783	137	7.735453D−08	0.23
POWER	1000	66	182	147	4.436287D−09	0.07
POWER	10000	169	803	444	9.479817D−10	0.54
QUARTC	1000	26	100	38	1.292595D−01	0.02
QUARTC	10000	31	126	48	3.659146D+00	0.06
SCHMVETT	1000	14	35	36	−2.994000D+03	0.04
SCHMVETT	10000	18	68	32	−2.999400D+04	0.19

continued on next page

Table 6 (*Continued*)

Problem	n	Iter	Funct	CG-it	F. value	Time
SINQUAD	1000	38	313	47	−2.942505D+05	0.05
SINQUAD	10000	104	1517	109	−2.642315D+07	0.98
SPARSINE	1000	111	581	2864	2.156776D−05	0.99
SPARSINE	10000	409	2172	33699	1.829184D−03	122.14
SPARSQUR	1000	21	65	27	2.806241D−08	0.03
SPARSQUR	10000	23	68	36	3.854467D−09	0.19
SPMSRTLS	1000	56	205	192	6.219291D+00	0.08
SPMSRTLS	10000	1914	2339	23969	5.686345D+01	67.17
SROSENBR	1000	35	309	40	2.842418D−22	0.04
SROSENBR	10000	104	920	108	9.421397D−12	0.25
TESTQUAD	1000	126	675	1396	1.766825D−07	0.17
TOINTGSS	1000	2	3	1	1.001002D+01	0.01
TOINTGSS	10000	2	3	1	1.000100D+01	0.00
TQUARTIC	1000	122	367	207	3.975538D−16	0.10
TQUARTIC	10000	14	144	18	1.145916D−11	0.05
TRIDIA	1000	54	445	302	8.428542D−08	0.05
TRIDIA	10000	139	1766	1243	2.611580D−08	1.38
VARDIM	1000	37	37	72	7.308972D−21	0.04
VARDIM	10000	54	298	99	4.622696D−18	0.18
VAREIGVL	1000	23	48	155	1.271565D−10	0.06
VAREIGVL	10000	21	179	22	3.924839D−16	0.17
WOODS	1000	54	334	102	3.938291D−16	0.04
WOODS	10000	124	1066	170	8.120429D−10	0.43

remarked, the scheme in [21] generates at the outer iteration j a preconditioner which is used in the $(j+1)$-th outer iteration.

In order to carry on a full comparison with the scheme in the paper [21], we also investigated the possibility to adopt the following strategy: to compute our preconditioner at the j-th outer iteration, then to use it at the $(j+1)$-th outer iteration. This implies that now in order to solve Newton's equation (5.1) we adopted the preconditioned solver Prec-CG, which generated directions to build the preconditioner for the $(j+1)$-th outer iteration. On this guideline, can we really replace the role of the Algorithm CG with the Algorithm Prec-CG, and still obtain a result similar to Theorem 2.1? The next theorem answers the last question.

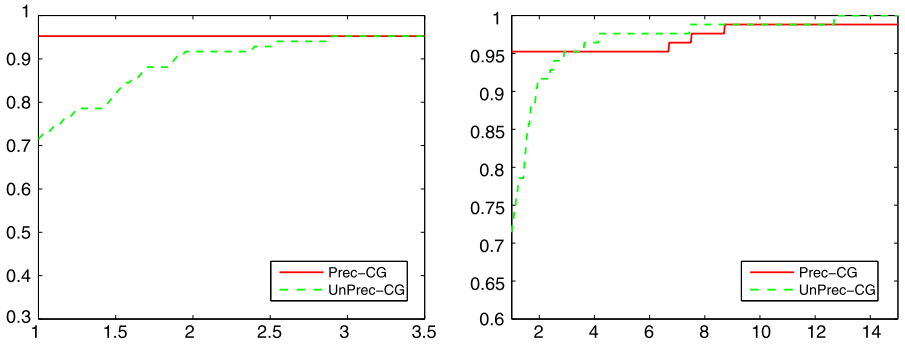


Fig. 1 Comparison (*detail* on the *left* and *full* on the *right*) between Prec-CG vs CG, in terms of number of inner iterations. The comparison includes all the test problems

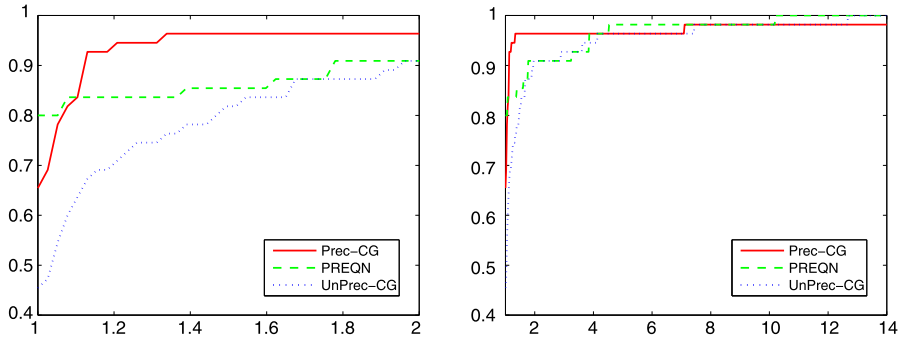


Fig. 2 Comparison (*detail* on the *left* and *full* on the *right*) among Prec-CG vs PREQN vs CG, in terms of number of inner iterations. The comparison includes only those test problems where negative curvature directions are not detected

Theorem 8.1 *Consider the preconditioned method* Prec-CG($\mathcal{M}$) *to solve Newton's equation* (5.1)*; suppose it performs* $h \leq n$ *iterations, using the preconditioner* $\mathcal{M}^{-1}$*. Then, the following matrices are generated*

$$\tilde{T}_h = \tilde{L}_h \tilde{D}_h \tilde{L}_h^T \tag{8.1}$$

$$\tilde{D}_h = \operatorname{diag}\left\{\frac{1}{\tilde{a}_1}, \ldots, \frac{1}{\tilde{a}_h}\right\}$$

$$\tilde{L}_h = \begin{pmatrix} 1 & & & \\ -\sqrt{\tilde{\beta}_1} & 1 & & \\ & -\sqrt{\tilde{\beta}_2} & \ddots & \\ & & \ddots & \ddots \\ & & & -\sqrt{\tilde{\beta}_{h-1}}\ 1 \end{pmatrix}$$

$$\tilde{R}_h = \left[\frac{z_1}{(r_1^T z_1)^{1/2}}, \ldots, \frac{z_h}{(r_h^T z_h)^{1/2}} \right] \tag{8.2}$$

such that Assumption 2.1 holds. Moreover, the matrix

$$\tilde{M}_h = \left[I - \left(\mathcal{M}^{1/2}\tilde{R}_h\right)\left(\mathcal{M}^{1/2}\tilde{R}_h\right)^T \right] + \left(\mathcal{M}^{1/2}\tilde{R}_h\right)|\tilde{T}_h|\left(\mathcal{M}^{1/2}\tilde{R}_h\right)^T \tag{8.3}$$

satisfies the properties

(a) $\tilde{M}_h$ *is symmetric and nonsingular*;
(b) $\tilde{M}_h^{-1} = [I - (\mathcal{M}^{1/2}\tilde{R}_h)(\mathcal{M}^{1/2}\tilde{R}_h)^T] + (\mathcal{M}^{1/2}\tilde{R}_h)|\tilde{T}_h^{-1}|(\mathcal{M}^{1/2}\tilde{R}_h)^T$;
(c) $\tilde{M}_h$ *is positive definite and its spectrum* $\Lambda(\tilde{M}_h)$ *is given by*

$$\Lambda(\tilde{M}_h) = \Lambda(|\tilde{T}_h|) \cup \Lambda(I_{n-h}).$$

Proof The proof follows the same guidelines used to prove Theorem 2.1. □

We remark that in the previous theorem the items (d) and (e) of Theorem 2.1 are not included, since in general they do not hold. Moreover, observe that unlike the matrix M_h^{-1} defined in Theorem 2.1, the matrix $\tilde{M}_h^{-1}$ in Theorem 8.1 cannot be directly used as a preconditioner, since it requires the computation of $\mathcal{M}^{1/2}$ (see (8.3)), which is unavailable. This implies that, as long as we use the same preconditioning strategy, a full comparison between the preconditioner PREQN in [21] and our preconditioner can be hardly carried on.

9 Conclusions and future work

In this paper we propose a new preconditioning technique for efficiently solving indefinite linear systems arising in large scale optimization. Krylov subspace methods are considered for the iterative solution of the linear systems, and the preconditioner is obtained as by product of the Krylov methods iterates. In fact, the preconditioner is built by means of an iterative decomposition of the system matrix, without storing or handling the system matrix. The only information on this matrix is gained by a routine which computes the product of the matrix times a vector. The numerical results obtained showed that the proposed strategy may often reduce the number of inner iterations needed to solve the optimization problem, within the framework of Newton-type methods. In a future work we could consider extensions of our approach by introducing ad hoc adaptive preconditioning rules. On this purpose, let us consider the relation $Q(s_2^{PR}) \leq Q(s_h)$ in Proposition 5.2; we can develop an adaptive procedure which decides whenever our preconditioner has to be used in the Algorithm Prec-CG. Indeed, though in principle the condition $Q(s_2^{PR}) \leq Q(s_h)$ indicates that preconditioning is always promising, it does not ensure that the overall preconditioned Truncated Newton method will outperform the unpreconditioned one. To this aim, recalling the Algorithms CG and Prec-CG, we can set

$$z = |\tilde{a}_1|, \qquad \beta_h = \frac{1}{2}\left(\sum_{i=1}^{h} |a_i| p_i\right)^T H_j \left(\sum_{i=1}^{h} |a_i| p_i\right),$$

$$\gamma_h = -r_1^T \left(\sum_{i=1}^{h} |a_i| p_i \right), \quad h \geq 1.$$

Then, we can find the values of the parameter δ ($\delta > 1$), which satisfy the condition $Q(s_2^{PR}) \leq \delta Q(s_h)$, i.e.

$$\beta_h z^2 + \gamma_h z - \delta(\beta_h + \gamma_h) \leq 0. \tag{9.1}$$

From Proposition 5.2 observe that since $Q(s_2^{PR}) < 0$ and $\beta_h + \gamma_h = Q(s_h) < 0$, the condition $\delta > 1$ imposes that preconditioning yields a *sufficient decrease* of the quadratic model of the objective function. Also observe that $\gamma_h < 0$ for any $h \geq 1$, but β_h may have any real value. Thus, in a more general setting, we can assess a suitable adaptive preconditioning strategy based on the values of γ_h and β_h.

Acknowledgements The authors wish to thank the national research program "Programma PRIN 20079PLLN7 *Nonlinear Optimization, Variational Inequalities, and Equilibrium Problems*". The first author wishes also to thank the CNR-INSEAN (Italian Ship Model Basin) research program "*RITMARE*".

Appendix

Proof of Proposition 5.2 From the definition of M_h^{-1}, we have

$$\begin{aligned}
M_h^{-1} r_1 &= \left[\left(I - R_h R_h^T \right) + R_h |T|^{-1} R_h^T \right] r_1 \\
&= r_1 - R_h \begin{pmatrix} \|r_1\| \\ 0 \\ \vdots \\ 0 \end{pmatrix} + R_h \left[L_h |D_h| L_h^T \right]^{-1} \begin{pmatrix} \|r_1\| \\ 0 \\ \vdots \\ 0 \end{pmatrix} \\
&= R_h L_h^{-T} |D_h|^{-1} L_h^{-1} \begin{pmatrix} \|r_1\| \\ 0 \\ \vdots \\ 0 \end{pmatrix} \\
&= R_h L_h^{-T} \operatorname{diag}_{1 \leq i \leq h} \{ |a_i| \} \begin{pmatrix} 1 \\ \sqrt{\beta_1} \\ \vdots \\ \sqrt{\beta_1 \cdots \beta_{h-1}} \end{pmatrix} \|r_1\| \\
&= P_h \operatorname{diag}_{1 \leq i \leq h} \{ |a_i| \} \begin{pmatrix} \|r_1\| \\ \|r_2\| \\ \vdots \\ \|r_h\| \end{pmatrix} = \sum_{i=1}^{h} |a_i| p_i .
\end{aligned} \tag{A.1}$$

From the latter relation, if $\tilde{a}_1 = 1$ in (5.4), then the directions s_h and s_2^{PR} coincide. Now, by definition we have

$$Q(s_h) = \frac{1}{2} s_h^T H_j s_h + g_j^T s_h = \frac{1}{2} \left(\sum_{i=1}^{h} |a_i| p_i \right)^T H_j \left(\sum_{i=1}^{h} |a_i| p_i \right) - r_1^T \left(\sum_{i=1}^{h} |a_i| p_i \right)$$
$$= \frac{1}{2} \sum_{i=1}^{h} a_i^2 p_i^T H_j p_i - \sum_{i=1}^{h} |a_i| \|r_i\|^2 = \sum_{i=1}^{h} \left[\frac{1}{2} \operatorname{sgn}\left(p_i^T H_j p_i\right) - 1 \right] |a_i| \|r_i\|^2. \tag{A.2}$$

Moreover, from (A.1)

$$|\tilde{a}_1| = \left| \frac{r_1^T z_1}{\tilde{p}_1^T H_j \tilde{p}_1} \right| = \left| \frac{r_1^T M_h^{-1} r_1}{z_1^T H_j z_1} \right| = \left| \frac{r_1^T (\sum_{i=1}^{h} |a_i| p_i)}{(\sum_{i=1}^{h} |a_i| p_i)^T H_j (\sum_{i=1}^{h} |a_i| p_i)} \right|$$
$$= \left| \frac{\sum_{i=1}^{h} |a_i| \|r_i\|^2}{\sum_{i=1}^{h} a_i^2 p_i^T H p_i} \right| = \frac{\sum_{i=1}^{h} |a_i| \|r_i\|^2}{|\sum_{i=1}^{h} a_i^2 p_i^T H_j p_i|};$$

thus, we have also

$$Q\left(s_2^{PR}\right) = \frac{1}{2} \left(\frac{\sum_{i=1}^{h} |a_i| \|r_i\|^2}{\sum_{i=1}^{h} a_i^2 p_i^T H_j p_i} \right)^2 \left(\sum_{i=1}^{h} |a_i| p_i \right)^T H_j \left(\sum_{i=1}^{h} |a_i| p_i \right)$$
$$- \frac{\sum_{i=1}^{h} |a_i| \|r_i\|^2}{|\sum_{i=1}^{h} a_i^2 p_i^T H_j p_i|} r_1^T \left(\sum_{i=1}^{h} |a_i| p_i \right)$$
$$= \frac{1}{2} \frac{[\sum_{i=1}^{h} |a_i| \|r_i\|^2]^2}{\sum_{i=1}^{h} a_i^2 p_i^T H_j p_i} - \frac{[\sum_{i=1}^{h} |a_i| \|r_i\|^2]^2}{|\sum_{i=1}^{h} a_i^2 p_i^T H_j p_i|}$$
$$= \frac{1}{2} \frac{[\sum_{i=1}^{h} |a_i| \|r_i\|^2]^2}{\sum_{i=1}^{h} a_i \|r_i\|^2} - \frac{[\sum_{i=1}^{h} |a_i| \|r_i\|^2]^2}{|\sum_{i=1}^{h} a_i \|r_i\|^2|}.$$

Furthermore, observe that $Q(s_2^{PR}) \le Q(s_h)$ if and only if

$$\frac{1}{2} \frac{[\sum_{i=1}^{h} |a_i| \|r_i\|^2]^2}{\sum_{i=1}^{h} a_i \|r_i\|^2} - \frac{[\sum_{i=1}^{h} |a_i| \|r_i\|^2]^2}{|\sum_{i=1}^{h} a_i \|r_i\|^2|} \le \frac{1}{2} \sum_{i=1}^{h} a_i \|r_i\|^2 - \sum_{i=1}^{h} |a_i| \|r_i\|^2,$$

or equivalently

$$\frac{\frac{1}{2}[\sum_{i=1}^{h} |a_i| \|r_i\|^2]^2 - \operatorname{sgn}(\sum_{i=1}^{h} a_i \|r_i\|^2)[\sum_{i=1}^{h} |a_i| \|r_i\|^2]^2}{\sum_{i=1}^{h} a_i \|r_i\|^2}$$
$$\le \frac{1}{2} \sum_{i=1}^{h} a_i \|r_i\|^2 - \sum_{i=1}^{h} |a_i| \|r_i\|^2. \tag{A.3}$$

To prove the latter relation we separately consider two cases: the case $\sum_{i=1}^{h} a_i \|r_i\|^2 > 0$ and the case $\sum_{i=1}^{h} a_i \|r_i\|^2 < 0$. In the first case the relation (A.3) holds if and only if

$$\frac{1}{2}\left[\sum_{i=1}^{h} |a_i| \|r_i\|^2\right]^2 - \operatorname{sgn}\left(\sum_{i=1}^{h} a_i \|r_i\|^2\right)\left[\sum_{i=1}^{h} |a_i| \|r_i\|^2\right]^2 \leq \frac{1}{2}\left[\sum_{i=1}^{h} a_i \|r_i\|^2\right]^2 - \left[\sum_{i=1}^{h} |a_i| \|r_i\|^2\right]\sum_{i=1}^{h} a_i \|r_i\|^2$$

or equivalently if and only if

$$-\frac{1}{2}\left[\sum_{i=1}^{h} |a_i| \|r_i\|^2\right]^2 \leq \left[\frac{1}{2}\sum_{i=1}^{h} a_i \|r_i\|^2 - \sum_{i=1}^{h} |a_i| \|r_i\|^2\right]\sum_{i=1}^{h} a_i \|r_i\|^2,$$

and the latter inequality holds since

$$-\frac{1}{2}\left[\left(\sum_{i=1}^{h} |a_i| \|r_i\|^2\right)^2 + \left(\sum_{i=1}^{h} a_i \|r_i\|^2\right)^2\right] + \sum_{i=1}^{h} |a_i| \|r_i\|^2 \sum_{i=1}^{h} a_i \|r_i\|^2 \leq 0.$$

In the second case the relation (A.3) is equivalent to

$$\frac{3}{2}\left[\sum_{i=1}^{h} |a_i| \|r_i\|^2\right]^2 \geq \frac{1}{2}\left[\sum_{i=1}^{h} a_i \|r_i\|^2\right]^2 + \left|\sum_{i=1}^{h} a_i \|r_i\|^2\right| \sum_{i=1}^{h} |a_i| \|r_i\|^2,$$

which holds since

$$\begin{aligned}\frac{3}{2}\left[\sum_{i=1}^{h} |a_i| \|r_i\|^2\right]^2 &= \frac{1}{2}\left[\sum_{i=1}^{h} |a_i| \|r_i\|^2\right]^2 + \left|\sum_{i=1}^{h} |a_i| \|r_i\|^2\right| \sum_{i=1}^{h} |a_i| \|r_i\|^2 \\ &\geq \frac{1}{2}\left[\sum_{i=1}^{h} a_i \|r_i\|^2\right]^2 + \left|\sum_{i=1}^{h} a_i \|r_i\|^2\right| \sum_{i=1}^{h} |a_i| \|r_i\|^2.\end{aligned}$$

This finally proves that

$$Q\left(s_2^{PR}\right) \leq Q(s_h). \qquad \square$$

References

1. Baglama, J., Calvetti, D., Golub, G.H., Reichel, L.: Adaptively preconditioned GMRES algorithms. SIAM J. Sci. Comput. **20**, 243–269 (1998)
2. Bernstein, D.S.: Matrix Mathematics, 2nd edn. Princeton University Press, Princeton (2009)
3. Conn, A.R., Gould, N.I.M., Toint, P.L.: Trust-Region Methods. MPS—SIAM Series on Optimization. SIAM, Philadelphia (2000)

4. Dembo, R.S., Steihaug, T.: Truncated-Newton algorithms for large-scale unconstrained optimization. Math. Program. **26**, 190–212 (1983)
5. Dolan, E.D., Moré, J.: Benchmarking optimization software with performance profiles. Math. Program. **91**, 201–213 (2002)
6. Erhel, J., Burrage, K., Pohl, B.: Restarted GMRES preconditioned by deflation. J. Comput. Appl. Math. **69**, 303–318 (1996)
7. Fasano, G.: Planar-conjugate gradient algorithm for large-scale unconstrained optimization, Part 1: Theory. J. Optim. Theory Appl. **125**, 523–541 (2005)
8. Fasano, G.: Planar-conjugate gradient algorithm for large-scale unconstrained optimization, Part 2: Application. J. Optim. Theory Appl. **125**, 543–558 (2005)
9. Fasano, G.: Lanczos-conjugate gradient method and pseudoinverse computation, in unconstrained optimization. J. Optim. Theory Appl. **132**, 267–285 (2006)
10. Fasano, G., Roma, M.: Iterative computation of negative curvature directions in large scale optimization. Comput. Optim. Appl. **38**, 81–104 (2007)
11. Gill, P.E., Murray, W., Ponceleon, D.B., Saunders, M.A.: Preconditioners for indefinite systems arising in optimization. SIAM J. Matrix Anal. Appl. **13**, 292–311 (1992)
12. Giraud, L., Gratton, S.: On the sensitivity of some spectral preconditioners. SIAM J. Matrix Anal. Appl. **27**, 1089–1105 (2006)
13. Golub, G.H., Van Loan, C.F.: Matrix Computations, 3rd edn. The John Hopkins Press, Baltimore (1996)
14. Gould, N.I.M., Lucidi, S., Roma, M., Toint, Ph.L.: Exploiting negative curvature directions in linesearch methods for unconstrained optimization. Optim. Methods Softw. **14**, 75–98 (2000)
15. Gould, N.I.M., Orban, D., Toint, Ph.L.: CUTEr (and sifdec), a constrained and unconstrained testing environment, revised. ACM Trans. Math. Softw. **29**, 373–394 (2003)
16. Gratton, S., Sartenaer, A., Tshimanga, J.: On a class of limited memory preconditioners for large scale linear systems with multiple right-hand sides. SIAM J. Optim. **21**, 912–935 (2011)
17. Grippo, L., Lampariello, F., Lucidi, S.: A truncated Newton method with nonmonotone linesearch for unconstrained optimization. J. Optim. Theory Appl. **60**, 401–419 (1989)
18. Kelley, C.T.: Iterative Methods for Optimization. SIAM Frontiers in Applied Mathematics. SIAM, Philadelphia (1999)
19. Loghin, L., Ruiz, D., Touhami, A.: Adaptive preconditioners for nonlinear systems of equations. J. Comput. Appl. Math. **189**, 362–374 (2006)
20. Morales, J.L., Nocedal, J.: Algorithm PREQN: Fortran 77 subroutine for preconditioning the conjugate gradient method. ACM Trans. Math. Softw. **27**, 83–91 (2001)
21. Morales, J.L., Nocedal, J.: Automatic preconditioning by limited memory quasi-Newton updating. SIAM J. Optim. **10**, 1079–1096 (2000)
22. Nash, S.G.: Preconditioning of truncated-Newton methods. SIAM J. Sci. Stat. Comput. **6**, 599–616 (1985)
23. Nash, S.G.: A survey of truncated-Newton methods. J. Comput. Appl. Math. **124**, 45–59 (2000)
24. Nash, S.G., Sofer, A.: Assessing a search direction within a truncated-Newton method. Oper. Res. Lett. **9**, 219–221 (1990)
25. Nocedal, J.: Large scale unconstrained optimization. In: Watson, A., Duff, I. (eds.) The State of the Art in Numerical Analysis, pp. 311–338. Oxford University Press, Oxford (1997)
26. Roma, M.: A dynamic scaling based preconditioning for truncated Newton methods in large scale unconstrained optimization. Optim. Methods Softw. **20**, 693–713 (2005)
27. Stoer, J.: Solution of large linear systems of equations by conjugate gradient type methods. In: Bachem, A., Grötschel, M., Korte, B. (eds.) Mathematical Programming. the State of the Art, pp. 540–565. Springer, Berlin (1983)
28. Tshimanga, J., Gratton, S., Weaver, A.T., Sartenaer, A.: Limited-memory preconditioners with applications to incremental four-dimensional variational data assimilation. Q. J. R. Meteorol. Soc. **134**, 751–769 (2008)