Journal of Computational and Applied Mathematics 268 (2014) 82–92

Contents lists available at ScienceDirect

Journal of Computational and Applied Mathematics

journal homepage: www.elsevier.com/locate/cam

# A matrix-free approach to build band preconditioners for large-scale bound-constrained optimization☆

V. De Simone [a], D. di Serafino [a,b,*]

[a] Dipartimento di Matematica e Fisica, Seconda Università degli Studi di Napoli, viale A. Lincoln 5, I-81100 Caserta, Italy

[b] Istituto di Calcolo e Reti ad Alte Prestazioni (ICAR), CNR, via P. Castellino 111, I-80131 Napoli, Italy

ARTICLE INFO

*Article history:*
Received 7 May 2013
Received in revised form 30 December 2013

*MSC:*
65F08
90C30

*Keywords:*
Band preconditioners
Matrix-free approach
Bound-constrained nonlinear optimization

ABSTRACT

We propose a procedure for building symmetric positive definite band preconditioners for large-scale symmetric, possibly indefinite, linear systems, when the coefficient matrix is not explicitly available, but matrix–vector products involving it can be computed. We focus on linear systems arising in Newton-type iterations within matrix-free versions of projected methods for bound-constrained nonlinear optimization. In this case, the structure and the size of the matrix may significantly change in subsequent iterations, and preconditioner updating algorithms that exploit information from previous steps cannot be easily applied. Our procedure is based on a recursive approach that incrementally improves the quality of the preconditioner, while requiring a modest number of matrix–vector products. A strategy for dynamically choosing the bandwidth of the preconditioners is also presented. Numerical results are provided, showing the performance of our preconditioning technique within a trust-region Newton method for bound-constrained optimization.



## 1. Introduction

We are interested in the solution of linear systems

$$Hd = -g \tag{1}$$

that arise in Newton-type iterations within projected methods for large-scale optimization problems of the form

$$\begin{array}{ll} \text{minimize} & f(x), \\ \text{s.t.} & l \le x \le u, \end{array}$$

where $f \in C^2([l, u])$, $l \in \{\mathbb{R} \cup \{-\infty\}\}^n$, $u \in \{\mathbb{R} \cup \{+\infty\}\}^n$. In these methods $H$ is usually (a symmetric approximation to) the reduced Hessian of $f$, and $g$ is the reduced gradient with respect to the free variables at the current step (see, e.g., [1–6]).

We assume that the matrix $H$ is sparse, is not available in a fully assembled form and/or is too large to fit into the main memory, but a procedure for computing matrix–vector products $Hv$ is available. In this context, computing inexact Newton steps through variants of the Conjugate Gradient (CG) method able to deal with negative curvature directions [7–9] is a

☆ This work was partially supported by INdAM-GNCS, under the 2012 Project *Advanced Numerical Methods for Preconditioning Linear Systems Arising from PDE and Optimization Problems* and the 2013 Project *Numerical Methods and Software for Large-Scale Optimization with Applications to Image Processing*.

* Corresponding author at: Dipartimento di Matematica e Fisica, Seconda Università degli Studi di Napoli, viale A. Lincoln 5, I-81100 Caserta, Italy. Tel.: +39 0823274742; fax: +39 0823274753.

*E-mail addresses:* valentina.desimone@unina2.it (V. De Simone), daniela.diserafino@unina2.it (D. di Serafino).

http://dx.doi.org/10.1016/j.cam.2014.02.035
0377-0427/

natural choice for the solution of system (1), since CG only uses the matrix through matrix–vector products. On the other hand, the convergence of CG depends on the spectral properties of the matrix, and preconditioning is usually needed to achieve reliable solutions in a reasonable time.

Preconditioning in a matrix-free approach can be performed without explicitly assembling the preconditioner, i.e., in a fully matrix-free regime, or by explicitly building a preconditioner that uses a modest amount of memory, possibly $O(m)$, where $m$ is the dimension of $H$. In both cases, $p \ll m$ matrix–vector products should be performed to obtain the preconditioner, to avoid that the improvement of the convergence speed is hidded by the cost of the preconditioner. When considering projected methods, it must be also taken into account that the working set, and hence the corresponding free variables, may significantly change from an iteration to the next one, hence reusing information from previous steps may not be easy.

A fully matrix-free approach for preconditioning sequences of linear systems is based on *quasi-Newton updating* techniques. The basic idea is to build an approximation to the Hessian or its inverse at the current outer iteration through suitable BFGS updates. For example, at iteration $k$, the (inverse) Hessian approximation $\widetilde{H}_k$ to be used as preconditioner can be obtained by BFGS updates of some initial approximation $\widetilde{H}$, by using pairs of correction vectors built from previous Newton iterates. Alternatively, the preconditioner for iteration $k$ can be obtained by applying BFGS updates based on correction vectors obtained as byproduct of the CG method applied to the Newton system at iteration $k-1$ [10]. Different BFGS approaches can be combined, as in the TN software [11]. The updated matrices are not computed explicitly, but only the vectors needed to perform the updates are stored; a limited-memory approach is generally used, where only $p$ pairs of correction vectors, with $p \ll m$, are considered. The resulting preconditioners are low cost, but are usually effective on sequences of slowly varying systems. Furthermore, in a projected framework, where the set of free variables changes until the active constraints at the solution have been identified, Hessian submatrices corresponding to different variables are considered at different iterations, thus the BFGS updates performed at a certain iteration may not be fully exploited at subsequent iterations.

A different approach, aimed at explicitly building preconditioners, employs *sparse matrix computation techniques*. Such techniques, which require the knowledge of the matrix sparsity pattern, date back to [12], and since then they have been largely used to compute sparse Hessians (and Jacobians) when only matrix–vector products with these matrices, or approximations to the matrix–vector products via finite differences or automatic differentiation, are available (see the survey paper [13]). Generally speaking, the problem of efficiently computing the nonzero entries of a matrix $H$, with known sparsity pattern $\mathcal{S}(H)$, can be formulated as follows: given $\mathcal{S}(H)$, find a set of binary vectors $\{v_1, v_2, \ldots, v_p\}$, of minimum cardinality, such that the nonzero entries of $H$ can be determined from the products $Hv_1, Hv_2, \ldots, Hv_p$. This problem is equivalent to finding a so-called structurally orthogonal partition of the columns of $H$ that has the minimum number of column groups; if $H$ is symmetric, as in the case we are interested in, symmetrically structurally orthogonal partitions can be used to reduce the number of matrix–vector products. As first recognized in [14], these problems can be formulated as graph coloring problems; their solution is NP-hard, but several algorithms are available to obtain practically effective colorings.

The previous concepts and techniques can be slightly modified for preconditioning purposes, to build an approximation to $H$ made of a subset of the nonzero entries of the matrix, i.e., to perform a *partial matrix computation*. A further approximation can be obtained by choosing a simple sparsity pattern, e.g., through some sparsification of $H$, and then applying graph coloring algorithms to this pattern with matrix–vector products involving the whole matrix $H$ [15]. The result is a so-called *partial matrix estimation*, i.e., an approximation to the matrix entries belonging to the selected pattern. Of course, the closer the sparsity pattern to the original one, the better the preconditioner is expected to be; on the other hand, more complete patterns usually correspond to higher computational costs for the computation and application of the preconditioner. We note that the computed preconditioner is usually applied through a factorization of it, therefore the choice of the sparsity pattern should take into account the cost for such factorization. We also observe that the preconditioner obtained through partial matrix computation or approximation may not preserve some property of the matrix $H$, e.g., positive definiteness, and a correction procedure must be applied if we wish to recover it. Finally, we note that the strategies proposed in [16] and [17, Approach (A3)] to build diagonal and band preconditioners, respectively, can be regarded as special cases of partial matrix estimation plus suitable corrections to obtain positive-definite matrices.

Recently, a preconditioner based on a *limited-memory partial Cholesky factorization* has been proposed in the context of matrix-free interior point methods for linear and quadratic programming [18] and extended to least squares problems [19]. The preconditioner is obtained by applying the Cholesky factorization to the $k$ columns of the matrix $H$ corresponding to the $p$ largest pivots, with $p \ll m$, and using a suitable diagonal approximation to the remaining part of the matrix. If $H$ is sparse, these columns can be computed with few matrix–vector products. However, we note that this approach requires the knowledge of the complete diagonal of $H$, thus limiting the application of the preconditioner.

We note that, when a sequence of linear systems must be solved, the explicit construction of a preconditioner in a matrix-free regime in principle allows for the application of preconditioner updating techniques such as those presented in [20–25], provided that information on the difference between the matrices of the sequence is available or can be estimated at a reasonable cost. This approach has been recently used in [26], where an approximation to the difference matrix needed by the updating algorithm is either explicitly computed by partial matrix estimation, or, in the case of separable functions providing the matrix–vector products, is implicitly obtained by exploiting separability. However, in projected methods, the part of the Hessian considered at each iteration may significantly change, thus making the previous techniques not easily applicable, and recomputing a cheap preconditioner for each system may result more efficient.