ISTITUTO DI ANALISI DEI SISTEMI ED INFORMATICA
CONSIGLIO NAZIONALE DELLE RICERCHE

M. Roma

DYNAMIC SCALING BASED PRECONDITIONING FOR TRUNCATED NEWTON METHODS IN LARGE SCALE UNCONSTRAINED OPTIMIZATION: THE COMPLETE RESULTS

R. 579 Dicembre 2002

Massimo Roma - Dipartimento di Informatica e Sistemistica "A. Ruberti", Università di Roma "La Sapienza", via Buonarroti 12 - 00185 Roma, Italy and Istituto di Analisi dei Sistemi ed Informatica del CNR "A. Ruberti", viale Manzoni 30 - 00185 Roma, Italy.
Email : `roma@dis.uniroma1.it` URL: `www.dis.uniroma1.it/~roma`.

This work was supported by Progetto FIRB "Ottimizzazione non lineare su larga scala"

Istituto di Analisi dei Sistemi ed Informatica, CNR
viale Manzoni 30
00185 ROMA, Italy

tel. ++39-06-77161
fax ++39-06-7716461
email: iasi@iasi.rm.cnr.it
URL: http://www.iasi.rm.cnr.it

Abstract

This paper deals with the preconditioning of truncated Newton methods for the solution of large scale nonlinear unconstrained optimization problems. We focus on preconditioners which can be naturally embedded in the framework of truncated Newton methods, i.e. which can be built without storing the Hessian matrix of the function to be minimized, but only based upon information on the Hessian obtained by the product of the Hessian matrix times a vector. In particular we propose a diagonal preconditioning which enjoys this feature and which enables us to examine the effect of diagonal scaling on truncated Newton methods. In fact, this new preconditioner carries out a scaling strategy and it is based on the concept of equilibration of the data in linear systems of equations. An extensive numerical testing has been performed showing that the diagonal preconditioning strategy proposed is very effective. In fact, on most problems considered, the resulting diagonal preconditioned truncated Newton method performs better than both the unpreconditioned method and the one using an automatic preconditioner based on limited memory quasi–Newton updating (PREQN) recently proposed in [23].

Key words: truncated Newton method, conjugate gradient method, preconditioning, row–column scaling, equilibrated matrix.

1. Introduction

This paper deals with large scale nonlinear unconstrained optimization problems, i.e. problems of finding a local minimizer of a real valued function $f : \mathbb{R}^n \longrightarrow \mathbb{R}$, namely

$$\begin{array}{c} \min f(x) \\ x \in \mathbb{R}^n \end{array}$$

where the number of variables n is large. The function f is assumed to be twice continuously differentiable.

The growing interest in solving large scale problems is mainly due to the fact that in many and different contexts and applications such problems arise very frequently. Moreover, as well known, efficiently solving unconstrained problems is very important in the framework of constrained optimization too, for instance when a penalty–barrier approach is used.

Among the existing methods in large scale unconstrained optimization (see, e.g. [28] [31] for a survey) the truncated Newton methods represent one of the most powerful, reliable and flexible tool for solving large scale problems. In fact, they have a solid convergence theory, they retain the good Newton–like convergence properties exhibiting an excellent rate of convergence; moreover, they are robust and efficient also in solving "difficult problems" as highly nonlinear and ill–conditioned problems. Moreover truncated Newton methods have been successfully used in solving real world problems arising in many applications and several algorithmic enhancements have been developed and studied in the last years, too (see the excellent survey of truncated Newton methods [25] recently published by Nash and the references reported therein).

Although truncated Newton methods have been widely studied and extensively tested a number of important open questions still exist (see [28] Section 3.2) and motivate this work. One of these open questions addresses the formulation of an effective preconditioning strategy which enables to improve the behaviour of a truncated Newton method in tackling large scale problems. In this paper this question is considered, focusing on "general purpose" preconditioners suitable within the truncated Newton method framework, that is preconditioners which do not require the explicit knowledge of the Hessian matrix. The information about the Hessian matrix are gained only by means of a routine which provides us with the matrix–vector products of the Hessian matrix times a vector, i.e. by using a tool already required by any truncated Newton method implementation. Of course, this makes this kind of preconditioners also suitable within discrete truncated Newton methods where these matrix–vector products are approximated by finite differences [29]. Moreover, we focus our attention on preconditioners which are also "dynamic", i.e. which change during the iterations of the method.

Very few preconditioners enjoying all these features have been developed up to now. The most recent and efficient proposal of such a preconditioner is due to Morales and Nocedal [23] which proposed an automatic preconditioning strategy based on limited memory quasi–Newton updating.

In this work we propose a diagonal preconditioning strategy which enjoys all these desired features. It is based on the equilibration of data in linear systems and it consists of scaling the column vectors of the Hessian matrix. Generally speaking, column scaling is carried out by dividing each column of the matrix by the norm of the column, where different norms may be considered. Equilibrating matrices is a topic of great importance in the numerical solution of linear systems, and, as far as the author is aware, it has not yet been fully exploited as a useful tool for building good "general purpose" preconditioners for truncated Newton methods, even though the optimal properties of the scaled matrices—in terms of condition numbers—were well known since thirty year ago (see, e.g. [35], [5]). The need of equilibrating matrices in solving linear systems arising from real world problems is, in general, very evident; in fact, many times, these matrices have entries that vary over many orders of magnitude and their are very spread. In this case, a simple diagonal scaling would greatly reduce the condition number of the matrices. These general considerations are of fundamental importance in the framework of truncated Newton methods, since at each iteration of these methods a linear system must be (approximately) solved.

We embedded the diagonal preconditioner proposed within a linesearch based implementation of truncated Newton method and firstly, to check its reliability, we numerically tested it on a some real problems arising as finite dimensional approximation models taken form the MINPACK-2 collection [1]. Then we performed an extensive numerical testing on a very large set of test problems of the CUTE collection [6]. The results obtained confirmed our expectation: in fact this preconditioning strategy revealed inexpensive and very effective, reducing both the number of conjugate gradient iterations and the CPU time needed for solving most of the problems considered with respect to the unpreconditioned method. Moreover, a comparison with the results obtained on the same set of test problems by using the automatic preconditioner based on limited memory quasi–Newton updating PREQN proposed in [23] has been performed. Also from this comparison it is clearly pointed out the efficiency of the proposed diagonal preconditioner.

The paper is organized as follows: in Section 2 truncated Newton methods are briefly recalled pointing out those questions which are still considered open problems and, in particular, concerning preconditioning. In Section 3 the new diagonal preconditioning strategy is described and in Section 4 the results of the extensive numerical investigation are reported.

Throughout the paper the following notations are used: $g(x) = \nabla f(x)$ denotes the gradient and $H(x) = \nabla^2 f(x)$ the Hessian matrix of the function f. Moreover, $\|v\|_p$ denotes the ℓ_p–norm of a vector $v \in \mathbb{R}^n$ and for a $n \times n$ matrix A, $\|A\|_p$ denotes the induced matrix norm. Moreover, $\kappa_p(A) = \|A\|_p \|A^{-1}\|_p$ denotes the condition number of A in the ℓ_p–norm.

2. Preconditioned truncated Newton methods

In this section, truncated Newton methods (TN) are briefly recalled, focusing on their major components and in particular on preconditioning (a more precise description of truncated Newton methods can be found in [11] or, e.g., in the survey paper [25]).

As well known, given a guess x_k of a solution $x_\star$, the truncated Newton method is based on the quadratic model of the objective function f given by

$$q_k(d) = f(x_k) + g(x_k)^T d + \frac{1}{2} d^T H(x_k) d \tag{2.1}$$

and it is defined by iterations of the form

$$x_{k+1} = x_k + d_k$$

where the search direction d_k is obtained by approximately minimizing the quadratic model of the objective function (2.1) over $\mathbb{R}^n$. Of course, whenever the Hessian matrix $H(x_k)$ is positive definite, to minimize the quadratic model q_k is equivalent to solve the linear system

$$H(x_k) d = -g(x_k). \tag{2.2}$$

A good convergence rate of the method is still ensured by using a particular trade-off rule between the computational burden required to solve the system (2.2) and the accuracy with which it is solved [10]. A natural measure of this accuracy is the relative residual. In the truncated Newton methods an approximation of the Newton direction d_k is computed by applying an iterative method for approximately solving the linear system (2.2) and "truncating" the iterates whenever a required accuracy is achieved [11]. To this aim, the most frequently used algorithm is the linear conjugate gradient method (CG) since it is an efficient iterative method for solving positive definite linear systems.

The components of a truncated Newton method are several, providing the method with a great flexibility. In particular three key aspects for the overall efficiency of a truncated Newton method can be still considered open problems (see [28] Section 3.2): *i)* the formulation of an effective "truncation criterion" for the inner loop; *ii)* how to handle the case when the Hessian matrix is not positive definite; *iii)* the formulation and handling of a preconditioner to accelerate the convergence in the inner CG iterates.

In this paper we concentrate on preconditioning, which is considered one of the most important issues of the current research on truncated Newton methods. As well known preconditioning the inner CG iterations can considerably accelerate the convergence of the conjugate gradient method affecting the overall efficiency of a truncated Newton method. In fact, the convergence rate of the CG inner iterations is based on the condition number and the number of distinct eigenvalues of the Hessian matrix $H(x_k)$ (see, e.g. [13]). In particular, the larger the condition number, the slower the convergence of the method is; moreover, the more the eigenvalues are clustered, the sooner the error is reduced. Preconditioning enables to reduce either the condition number and the number of distinct eigenvalues thus accelerating the convergence of the CG method. A scheme of the Preconditioned Conjugate Gradient (PCG) algorithm applied to the solution of the system (2.2) is the following (see, e.g. [13] [18] [14]):

Preconditioned Conjugate Gradient Algorithm

PCG(M)

Given $d^0 = 0$ *and* M^{-1}, *set* $r^0 = g(x_k)$, $z^0 = M^{-1} r^0$ *and* $p^0 = -z^0$.

For $i = 0, 1, 2, \ldots$, *until convergence, perform the iteration*

$$
\begin{aligned}
d^{i+1} &= d^i + \alpha^i p^i \\
r^{i+1} &= r^i + \alpha^i H(x_k) p^i \\
&\quad \text{where} \quad \alpha^i = \frac{{r^i}^T z^i}{{p^i}^T H(x_k) p^i} \\
z^{i+1} &= M^{-1} r^{i+1} \\
p^{i+1} &= -z^{i+1} + \beta^i p^i \\
&\quad \text{where} \quad \beta^i = \frac{{r^{i+1}}^T z^{i+1}}{{r^i}^T z^i}
\end{aligned}
\tag{2.3}
$$

As well knonw, this PCG algorithm is obtained by applying the CG method to the transformed system $\widetilde{H}_k \widetilde{d} = -\widetilde{g}_k$, where $\widetilde{H}_k = CH(x_k)C^T$, $\widetilde{d} = C^{-T}d$, $\widetilde{g}_k = Cg(x_k)$ and C is a non-singular matrix. The matrix $M = (CC^T)^{-1}$ is the *preconditioner* and, due to similarity, $\widetilde{H}_k$ and $H(x_k)M^{-1}$ have the same conditioning. The computational burden of the PCG algorithm is more expensive than the unpreconditioned one, but it is now the matrix $H(x_k)M^{-1}$ which determines the convergence properties of the PCG method. More precisely the convergence depends on the condition number $\kappa_2(H(x_k)M^{-1})$ and on the number of distinct eigenvalues of $H(x_k)M^{-1}$.

It is out of the scope of this paper to discuss the well known features of preconditioning or to survey the very broad field of preconditioners (see, e.g. [4] [14] [36] [2] [21] for some references). We only mention in the sequel of this section the main preconditioning strategies up to now proposed in the framework of truncated Newton methods.

A preconditioning strategy which has been successfully used within truncated Newton methods is based on the incomplete (modified) Cholesky factorization [19] [33] [7]. It is a valuable choice since it is "general purpose" and a great reduction of the number of the inner CG iterations can be obtained. However, unfortunately, it requires the knowledge of the Hessian matrix. Another approach is based on efficiently exploiting the problem structure when it exists; in particular, many large scale problems possess the so called "partial separability" property, i.e. the objective function can be written as sum of simpler functions and this property can be exploited for defining "ad hoc" preconditioners [9].

In the context of truncated Newton methods, a first key point in developing an efficient preconditioning strategy for solving the sequence of systems (2.2) is to use a "dynamic preconditioner", i.e. a preconditioner which changes with the outer iterations. In particular, since at each outer iteration k the Hessian matrix changes, consequently the preconditioner should be changed. One possibility is to define, at each outer iteration k, a new preconditioner on the basis of the information gained during the solution of the system at the $(k-1)$–th iteration. This idea is not new, and its importance, as a challenge topic in large scale optimization, has already been clearly pointed out (see [28] Section 3.2).

Another key point is the availability of a preconditioner which can be effectively used within a truncated Newton method that is, which does not require to store and handle any matrix. By using this kind of preconditioners, the PCG algorithm is still free from the storage and handling of the Hessian matrix, like the unpreconditioned CG algorithm. Of course this feature is enjoyed by a preconditioner which can be built by using only information on the Hessian matrix obtained by performing products of the Hessian matrix times a vector. We remark that preconditioners of this kind can be used within discrete Newton methods [29] or when automatic differentiation [15] is used.

The first proposal of a preconditioned truncated Newton method which satisfies both these requirements was performed in [24]. It is based on a two–step limited memory BFGS updating; in particular, a diagonal scaling which uses a diagonal approximation of the Hessian matrix, obtained by means of BFGS updating is defined. This preconditioning strategy has been embedded in a discrete truncated Newton method which has been numerically tested and compared with other methods for large scale unconstrained optimization [26].

Recently, an automatic preconditioning based on m–step limited memory quasi–Newton updating has been proposed [23] [22]. This is a dynamic preconditioner which does not require the explicit knowledge of the Hessian matrix, too. This preconditioner has been numerically tested within a discrete Newton method and in the solution of linear systems arising in some finite element models; the results obtained in the solution of large scale unconstrained optimization problems indicate that this preconditioning strategy enables to reduce the number of CG inner iterations but it is expensive, due to the considerable computational cost of building and handling the preconditioner. This makes this particular preconditioner suited for problems for which the matrix–vector products of the Hessian matrix times a vector are expensive to compute. However, today this automatic preconditioning technique seems to be the best proposal within the class of dynamic preconditioners that we are considering.

3. Diagonal Preconditioning Truncated Newton Methods

On the basis of the remarks contained in the previous section, we now focus on diagonal preconditioning techniques since they can be efficiently used in the context of truncated Newton methods. In fact, we propose a diagonal preconditioner which possesses the desired features which makes its use appealing within a truncated Newton methods. Before going into details, we briefly report the main features of a generic diagonal preconditioner considering the solution of a generic linear system $As = b$ where A is a $n \times n$ nonsingular symmetric matrix and a_{ij} denotes its entries. First of all, we recall the well known concept of *equilibrated matrix* which will be used in the sequel

Definition 3.1. *A matrix A is said* row (column) equilibrated *if all its rows (columns) have the same length in some norm and it is said* equilibrated *if all its rows and columns have the same length in some norm.*

Diagonal preconditioning is the simplest form of preconditioning and it is very inexpensive to handle. The matrix M is chosen as a diagonal matrix

$$M = \mathrm{diag}\{d_1 \dots d_n\} = \begin{pmatrix} d_1 & & 0 \\ & \ddots & \\ 0 & & d_n \end{pmatrix}$$

where $d_i \in \mathrm{I\!R}$, $d_i > 0$, $i = 1, \dots, n$. This preconditioning corresponds to a *scaling* of the columns of the matrix A, aiming at obtaining a system with a column equilibrated matrix, that is a better conditioned system. Analogously a system with a row equilibrated matrix can be considered, while it is more complicated to obtain an equilibrated matrix. The advantages of having linear systems with equilibrated matrix is well known and its importance in numerical numerical analysis is witnessed by the number of papers devoted to this topic (see, e.g., [12] [34] [32] and the references therein). Moreover, there exist algorithms for equilibrating a matrix in a specific norm; some examples were proposed in [8], [30], [32]. Furthermore, optimality properties of scaling in ℓ_p–norm equilibrating the rows or the columns for minimizing the condition number are considered in [17] Section 7.3.

Note that to apply the preconditioned conjugate gradient method with this diagonal preconditioner, corresponds to apply the CG method to the system with the transformed matrix $D^{-1}AD^{-1}$ where $D = \mathrm{diag}\{d_1^{\frac{1}{2}}, \dots, d_n^{\frac{1}{2}}\}$.

It is well known that diagonal preconditioning presents some very good features; in fact it would greatly reduce the condition number of the matrix and it does not destroy the sparsity of the matrix; moreover it requires a minimal additional work and it is suitable to handle practical problems whose matrices entries frequently vary over many orders of magnitude.

A choice which is "optimal" with respect to diagonal preconditioners is to use the diagonal of the matrix A as diagonal preconditioner matrix (Jacobi preconditioner); in fact this choice enables to minimize the condition number of the matrix of the transformed system with respect to all the diagonal preconditioners. In fact, the following result holds [35] (see also, e.g., [14] [17] [21]):

Theorem 3.2. *Let A be a $n \times n$ symmetric positive definite matrix. Let $\mathcal{D}_n$ denote the set of $n \times n$ nonsingular diagonal matrices and $\Delta = \mathrm{diag}\{a_{11}^{\frac{1}{2}}, \dots, a_{nn}^{\frac{1}{2}}\}$. Then*

$$\kappa_2(\Delta^{-1} A \Delta^{-1}) \le p \min_{D \in \mathcal{D}_n} \kappa_2(D^{-1} A D^{-1}),$$

where p is the maximum number of non-zero elements in any row of A.

It is important to note that, even in this special case of the Jacobi preconditioner no results concerning the variation of the condition number or the eigenvalue distribution—and hence a possible improvement of the convergence rate—is available. In fact, Theorem 3.2 states only the existence of a lower bound but does not provide us with information about the variation of the condition number of the transformed system. This result, however, states that this kind of preconditioner is effective for sparse matrices. As regards the numerical efficiency of Jacobi preconditioner, it is known that it typically works well whenever the matrix A is very diagonally dominant.

Simple diagonal scaling of the variables has been already proved to be very effective in the context of L–BFGS methods [20] and within partitioned quasi–Newton methods [16] especially in tackling large scale problems. Moreover in [3] the effect of diagonal scaling with projected gradient methods for bound constrained problems has been investigated.

We can conclude these brief general considerations on diagonal scaling claiming that, even if there exist cases in which diagonal preconditioning does not lead to an improvement of the conditioning, however, in many cases an improvement, even if small, is observed and therefore the use of a diagonal preconditioning is usually recommended.

Now we investigate the possibility to define and efficiently use a diagonal preconditioning strategy within truncated Newton methods. In particular, we note that it corresponds to a scaling of the Hessian matrix which appears in the sequence of linear systems (2.2) and, as well known, to have "good" scaling in defining algorithms is always very important, even if the algorithm we are dealing with is a scaling invariant algorithm as the Newton method. More in details, as already observed, diagonal preconditioning corresponds to an equilibration of the data in the large linear systems to be solved at each iteration of the truncated Newton method.

A classical approach for row (column) scaling is to divide each row (column) by the norm of the row (column) vector. The most commonly used norms are the ℓ_1–norm and the ℓ_∞–norm. To scale matrices such that the norm of all rows (or columns) are equal to one is an heavy task from the computational point of view, and this should be the reason for which, as far as the author is aware, this numerical tool has not yet been fully exploited for building preconditioner in the context of truncated Newton methods, tackling large scale problems. In fact to compute the norm of each row (or column) of the Hessian matrix at each outer iteration of the method is computationally too expensive and thus impracticable, whichever norm is chosen. Also the use of the diagonal of the Hessian matrix itself as diagonal preconditioner is not practicable to derive efficiently by means of a routine which provides the product of the Hessian matrix times a vector.

On the basis of these remarks, we propose a strategy based on columns scaling of the Hessian matrix $H(x_k)$ in the solution of the systems (2.2) and, in particular, we focus on an equilibration strategy which uses the ℓ_1–norm.

In the sequel of this section, for the sake of simplicity, we remove the dependency on k and hence we refer to the system $Hd = -g$. Let h_{ij}, $i, j = 1, \dots, n$, denote the entries of the Hessian matrix H and for each $j = 1, \dots, n$ let $w_j \in \mathbb{R}^n$ defined by $w_j = He_j$, where $e_j = (0, \dots, 1, \dots, 0)^T$ is the j-th unit vector, that is, for each column j, w_j denote the column vectors of the matrix H (which are also the row–vectors for the symmetry). We consider the diagonal preconditioning generated by the matrix

$$D = \operatorname{diag}\left\{ \|w_1\|_1^{\frac{1}{2}}, \dots, \|w_n\|_1^{\frac{1}{2}} \right\} = \begin{pmatrix} \|w_1\|_1^{\frac{1}{2}} & & 0 \\ & \ddots & \\ 0 & & \|w_n\|_1^{\frac{1}{2}} \end{pmatrix}$$

Obviously, in terms of the preconditioning matrix, this corresponds to choose

$$M = \operatorname{diag}\{\|w_1\|_1, \dots, \|w_n\|_1\},$$

and it results that the matrix HM^{-1} is a *column equilibrated matrix* in the ℓ_1–norm.

It is now important to analyze the condition number and the eigenvalues distribution of the preconditioned matrix to check if the use of such a preconditioner leads to a better conditioned system. Note that the (unsymmetric) matrix HM^{-1} and the (symmetric) matrix $M^{-1/2}HM^{-1/2}$ are similar and hence they have the same eigenvalues. Firstly we address to the condition number of the preconditioned matrix showing that the particular column scaling proposed has a very important feature, since it is an optimal column scaling strategy. In fact, the following proposition holds.

Proposition 3.3. *If H is nonsingular and $\mathcal{D}_n$ denotes the set of $n \times n$ nonsingular diagonal matrices, then*

$$\kappa_1(HM^{-1}) \le \min_{D \in \mathcal{D}_n} \kappa_1(HD).$$

Proof: The results immediately follows from Theorem 7.5 in [17]. □

Therefore the proposed column scaling is the best column scaling in terms of condition number in ℓ_1–norm and due to the well known norm equivalence on a finite–dimensional space, it is possible to establish a relation between the condition number in ℓ_1–norm and in ℓ_2–norm, which differ by at most a constant that depends on the dimension n.

As regards the eigenvalues distribution of the preconditioned matrix, by exploiting the fact that the matrix HM^{-1} is column equilibrated in the ℓ_1–norm, it is possible to prove the following result.

Proposition 3.4. *Let $\rho(HM^{-1}) = \max\{\, |\lambda_i| : \lambda_i$ eigenvalues of $HM^{-1}\}$ be the spectral radius of the matrix HM^{-1}. Then it results*

$$\rho(HM^{-1}) \le 1.$$

Proof: The result can be obtained by observing that for a square matrix A it results $\rho(A) \leq \|A\|_p$ for every ℓ_p–norm. In fact, we have

$$\rho(HM^{-1}) \leq \|HM^{-1}\|_1 = 1,$$

where the last equality follows from the fact that $\|HM^{-1}e_j\|_1 = 1$ for all $j = 1, \dots, n$, i.e. all the columns of HM^{-1} have unitary ℓ_1–norm. $\square$

The result stated in this proposition shows how the use of the proposed preconditioner enables to have clustered eigenvalues and hence the PCG iterates are expected to converge quickly.

Now, we prove a general result concerning diagonal preconditioning.

Proposition 3.5. *Let $D = \text{diag}\{d_1, \dots, d_n\}$ be a $n \times n$ diagonal matrix with $d_j > 0$, $j = 1, \dots, n$ and $z \in \mathbb{R}^n$ such that $Hz = Dz$. Then the matrix $D^{-1/2}HD^{-1/2}$ admits an eigenvalue equal to one and the corresponding eigenvector is given by $v = D^{1/2}z$.*

Proof: Since $Hz = Dz$, we have

$$D^{-1/2}HD^{-1/2}v = D^{-1/2}Hz = D^{1/2}z = v$$

that is, v is an eigenvector of $D^{-1/2}HD^{-1/2}$ corresponding to the eigenvalue equal to one. $\square$

By applying this proposition, assuming that H is a nonnegative matrix, i.e. $h_{ij} \geq 0$ for all $i, j = 1, \dots, n$, it is possible to said more about the eigenvalues of the preconditioned matrix $M^{-1/2}HM^{-1/2}$. In fact, the following result holds.

Proposition 3.6. *If H is a nonnegative matrix, the preconditioned matrix $M^{-1/2}HM^{-1/2}$ admits an eigenvalue equal to one, and the corresponding eigenvector is given by $v = M^{1/2}e$, where $e = (1, \dots, 1)^T$.*

Proof: It is enough to apply Proposition 3.5 with $z = e$. In fact, in this case, the equality $He = Me$ holds. $\square$

Proposition 3.4 shows that the proposed preconditioner correspond to normalize the preconditioned matrix in such a way that its largest eigenvalue is less than or equal to one. Moreover, if the Hessian is a nonnegative matrix, Proposition 3.6 implies that the largest eigenvalue of the preconditioned matrix is equal to one.

Finally, Proposition 3.5 shows that this diagonal preconditioner can be viewed as a particular case of a more general setting in which in the definition of the preconditioner M, the vector e in $He = Me$ is replaced with any vector $z \in \mathbb{R}^n$ provided that $Hz = Mz$. However, note that if a vector different from e is used in the equality $He = Me$, the optimal properties of the ℓ_1–norm scaling do not necessarily hold, even if H is a nonnegative matrix.

Our aim is now to embed this diagonal preconditioning strategy in the framework of truncated Newton methods, but it is not straightforward, since the matrix H is not available. In order to obtain information on the ℓ_1–norm of the columns of the Hessian matrix we propose to use the only tool available in this context for providing us with information on the Hessian matrix, that is the routine for the product of H times a vector. In fact, we observe that an estimate of the ℓ_1–norm of the columns of the Hessian matrix can be obtained as follows: for each j-th column, $j = 1, \dots, n$ let us consider

$$\sigma_j = |\sum_{i=1}^{n} h_{ij}|. \tag{3.1}$$

These quantities have the following properties:

- they are a lower estimate of the ℓ_1–norm of the j-th column vector;
- due to the symmetry of H, all the σ_j, $j = 1, \dots, n$, can be very easily computed by means of the product of the Hessian matrix H times the vector $e = (1, 1, \dots, 1)^T$. In fact, by taking the absolute value of each component of the vector resulting from this product, we obtain all the σ_j. Therefore, only one call to the routine which provides us with the product of the Hessian matrix times a vector is enough to compute all the σ_j.

Obviously, for each column j for which $h_{ij} \geq 0$ for all $i = 1, \dots, n$, we have $\sigma_j = \|w_j\|_1$. Therefore, if H is a nonnegative matrix, then it results $\sigma_j = \|w_j\|_1$, for all $j = 1, \dots, n$.

The quantities (3.1) can be used to build the diagonal preconditioner defined by the following diagonal positive definite matrix

$$\widetilde{M} = \mathrm{diag}\{\tilde{\sigma}_1, \dots, \tilde{\sigma}_n\} = \begin{pmatrix} \tilde{\sigma}_1 & & 0 \\ & \ddots & \\ 0 & & \tilde{\sigma}_n \end{pmatrix} \tag{3.2}$$

where, for $j = 1, \dots n$,

$$\tilde{\sigma}_j = \begin{cases} \sigma_j & \text{if} \quad \sigma_j > \delta \\ 1 & \text{otherwise} \end{cases}$$

with $\delta > 0$ small.

By using this diagonal preconditioning, each column of the matrix is scaled by a quantity which represents an estimate of the ℓ_1–norm of the column vectors and no scaling is performed in correspondence of those columns for which σ_j is small.

The proposed diagonal preconditioner has the following important features:

- the explicit knowledge of the Hessian matrix is not required, but only a routine which provides the product of the Hessian matrix times a vector is needed, enabling its use within a truncated Newton method;
- it carries out a "dynamical" preconditioning technique, i.e. the preconditioner changes from one outer iterate to the next, thus overcoming the drawback of having the same preconditioner during the solution of each of the sequence of Newton's systems (2.2), whereas the Hessian matrix can change drastically;
- the solution of the auxiliary system (2.3) needed to compute the preconditioned residual is very inexpensive;

- the additional cost of preconditioning is essentially due to one additional call, at each outer iteration, to the subroutine which provides the matrix–vector product of the Hessian matrix times the vector $e = (1, \dots, 1)^T$.

This diagonal preconditioner represents a useful tool that can be viewed as a *"dynamic scaling"* since it is derived from *dynamic* preconditioning and *scaling* and we name it DSPREC.

We note that if the Hessian matrix H is a nonnegative matrix, the quantities σ_j in (3.1) are the ℓ_1–norm of the j-th column vector and therefore the results stated in Proposition 3.3, Proposition 3.4 and Proposition 3.6 apply; in this case this diagonal preconditioner posses the important theoretical properties stated. Otherwise, as it occurs in most cases when dealing with preconditioning, in the general setting, it is impossible to derive results which apply to any matrix. Therefore, since we are interested in "general purpose" preconditioner, the best assessment tool is represented by an extensive numerical study.

4. Numerical experiments

If a wide numerical investigation is usually considered an essential tool for assessing the efficiency of an algorithm, it is so much important in dealing with preconditioning strategies when, as often occurs, some choices are mainly based on numerical experiments rather than on solid theoretical results. Keeping in mind the central role of the numerical experiments in evaluating the effectiveness of different preconditioning strategies, we have been trying to perform a very extensive numerical testing. To this aim we embedded the proposed preconditioning strategy in a linesearch–based truncated Newton method. Our implementation of the method is now briefly described.

A preconditioned conjugate gradient algorithm is used for computing the search direction, terminating the inner PCG iterations whenever the residual–based criterion is satisfied or a negative curvature direction is encountered [11]. This criterion is preferred here with respect to the quadratic model–based criterion proposed in [27] since it allows to easily control the rate of convergence of the truncated Newton method. The method is outlined in the following scheme:

Linesearch–based truncated Newton method

`OUTER ITERATIONS`
For $k = 0, 1, \ldots$
Compute $g(x_k)$ and test for convergence

`INNER ITERATIONS` — *Computation of the search direction* d_k
Set $r^0 = g(x_k), \quad p^0 = \widetilde{M}_0^{-1} r^0$
For $i = 0, 1, \ldots$
perform an iteration of $\mathsf{PCG}(\widetilde{M}_k)$ algorithm until
the residual r^i satisfies $\quad \|r^i\|_2 \leq \|g(x_k)\|_2 \min\left\{\dfrac{1}{k+1}\ ,\ \|g(x_k)\|_2\right\}$
or a negative curvature direction p^i is found, namely ${p^i}^T H(x_k) p^i \leq \varepsilon \|p^i\|_2^2$

Compute a stepsize α_k which satisfies the Armijo condition, i.e., given $\eta \in (0,1)$ and $\gamma \in (0, 1/2)$, set $\alpha_k = \eta^h$, where h is the smallest nonnegative integer such that

$$f(x_k + \alpha_k d_k) \leq f(x_k) + \gamma \alpha_k d_k^T g(x_k)$$

Set $x_{k+1} = x_k + \alpha_k d_k$

The values of the parameters used are the following: $\gamma = 10^{-3}$, $\varepsilon = 10^{-6}$. In order to evaluate well the effect of preconditioning, we allow a large number of inner iterations, in fact, we take n as maximum number of inner PCG iterations allowed; moreover we set $\delta = 10^{-6}$ in the definition of the preconditioner (3.2). The termination criterion of the outer iterations is the standard test $\|g(x_k)\|_2 < 10^{-5} \max(1, \|x_k\|_2)$.

All the tests have been performed on a Pentium III 700 with 512 Mb RAM, and all the codes used are double precision FORTRAN codes. All the results are reported in terms of number of outer iterations (it) —which is the same as the number of gradient evaluation (ng)— number of function evaluations (nf), total number of inner iterations (CG-it) and CPU time (time).

As regards a different preconditioning strategy available in literature needed for making a comparison of the results, we have been seeking among those preconditioners which can be efficiently used within the truncated Newton framework and the state–of–the–art seems to be represented by the automatic preconditioner proposed in [23] [22], implemented in the available software PREQN that we already mentioned in the previous section. It has the form of a limited memory quasi-Newton matrix and it is generated using information from the CG iterations without requiring the explicit knowledge of the Hessian matrix but only products of this matrix times a vector. The preconditioner is automatically generated as follows: the first system of the sequence is solved by the unpreconditioned CG method; then, at each outer iteration k, a new preconditioner M_k is built based on the information gained during the solution of the most recent system $H(x_{k-1})d = -g(x_{k-1})$. More precisely, during the CG iterations m correction pairs are stored and then used to apply m times the BFGS updating formula to generate a new preconditioner which is the preconditioner for the subsequent system (see [23] [22] for details). This preconditioner belongs to the class of preconditioners we are considering, since it is a dynamic preconditioner which exploits only the matrix–vector product of the Hessian matrix times a vector as information on the Hessian matrix and therefore a coherent comparison of the results can be performed.

We embedded the preconditioner PREQN within our implementation of the linesearch–based truncated Newton method using the "uniform sampling" strategy for selecting the m vectors pairs, with $m = 8$ and refreshing the preconditioner at each outer iteration. In fact, according to the authors and to our numerical experiences, this seems to be the best choice. Note that the implementation we use is different from that used in [23]; in fact the method we implemented is not discrete, i.e. it does not uses finite differences for approximating the product of the Hessian times a vector; moreover we use a different truncation criterion of the inner iterations and a different linesearch procedure. A common feature is to terminate the inner iterations whenever a negative curvature direction is encountered; moreover the same stopping test for the outer iterations is used.

Since the proposed preconditioning strategy is also motivated by the possibility of more efficiently solving real problems, we start our numerical investigation by considering real unconstrained problems taken from the MINPACK-2 collection [1] for which a routine which computes the product of the Hessian matrix times a vector is available. The aim of this preliminary test is to check the reliability of the diagonal preconditioning strategy DSPREC as well as to compare the results with those obtained by means of unpreconditioned CG and preconditioner PREQN. We report these results in Table 4.1. As it can be observed from this table, on the problem GL2 (which is the most difficult in terms of computational burden needed to solve it) the use of the proposed diagonal preconditioning leads to a significative reduction of the total number of the inner iterations with respect to the results obtained by the unpreconditioned CG and by PREQN; moreover, a significant saving of the computing time is also noticed. This situation is evidenced in both the instances, that is the beneficial effect of diagonal preconditioning is not reduced as a consequence of an increase of the dimension of the problem; moreover, note that a reduction of number of the iterations is obtained, too. As regards the other two problems, they are solved very quickly by all the three algorithms even if, on the instances of 900 variables, the diagonal preconditioning does not seems to be effective since an increase of the number of CG iterations is observed; however, note that this does not lead to a significative increase of CPU time needed to solve the problems.

Now, after this preliminary test, we turn to a more general numerical investigation. In particular, we are interested in evaluating the effectiveness of the proposed diagonal preconditioning strategy as "general purpose" rather than an "ad hoc" strategy for a specific problem. To this aim, we now consider a very large set of test problems; in fact we use all the large scale

		Unpreconditioned CG				PREQN				DSPREC			
Problem	n	it/ng	nf	CG-it	time	it/ng	nf	CG-it	time	it/ng	nf	CG-it	time
GL2	400	28	19	6282	30.13	34	146	6031	33.54	17	60	4017	18.59
	900	37	149	11755	126.91	49	234	16927	211.47	16	50	9275	97.04
MSA	400	4	4	52	0.10	4	4	35	0.10	4	4	75	0.12
	900	5	5	116	0.32	5	5	79	0.35	5	5	223	0.58
ODC	400	11	15	268	0.47	11	16	168	0.49	12	15	269	0.48
	900	14	18	297	0.86	14	20	233	1.06	15	22	595	1.62

Table 4.1: Results on MINPACK-2 problems: 2-dimensional Ginzburg–Landau model for superconductivity with $n_v = 8$ (GL2); Minimal Surfaces (MSA); Optimal Design with Composite Materials with $\lambda = 0.008$ (ODC).

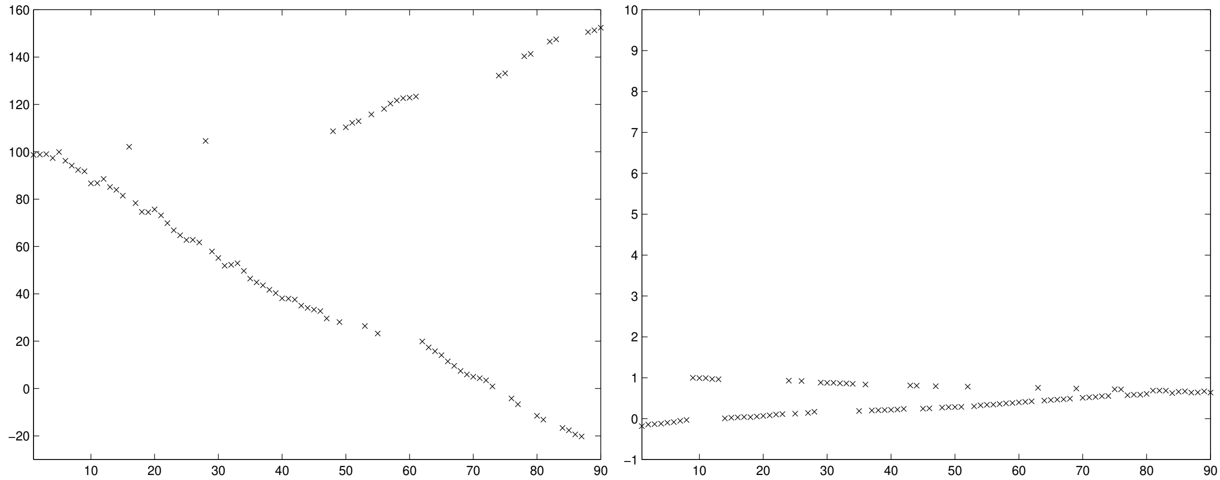

Figure 4.1: Distribution of the eigenvalues of the Hessian matrix H (on the left) and the preconditioned matrix $\widetilde{M}^{-1/2}H\widetilde{M}^{-1/2}$ (on the right) for the problem DIXMAANK $n = 90$

($n \geq 1000$) unconstrained problems available in the CUTE collection [6]. Moreover, for those problems with variable dimension, we consider two instances, usually $n = 1000$ and $n = 10000$ or different if a problem has assigned dimensions. These leads to consider 117 test problems. The computation of the matrix–vector products of the Hessian matrix times a vector is performed directly by using the routine `uprod` available from the CUTE environment.

The complete results obtained by our implementation of the linesearch–based truncated Newton using the unpreconditioned CG, the quasi–Newton preconditioner PREQN and the diagonal preconditioner DSPREC are reported in the Appendix. Here we report some summary of the results. Preliminarily, we observe that there are problems which are not solved by some of the algorithms, in the sense that convergence towards a stationary point is not observed within the prescribed number of 3000 outer iterations or within the time limit of 900 seconds. In particular, the unpreconditioned algorithm presents 14 failures while both the preconditioned algorithms fail to converge in 12 problems; moreover, on some problems the algorithms converge towards different points. Now, in order to make a comparison, in Tables 4.2 and 4.3, we report a summary of the results considering all those problems coherently solved by the three algorithms and where they converges towards to the same point. As it can be observed, the use of the diagonal preconditioner DSPREC enables a considerable computational saving on most problems with respect to both the unpreconditioned method and the one using PREQN, both in terms of number of inner iterations and in terms of CPU time and on some test problems the gain is impressive (see, e.g. problems DIXMAANE, DIXMAANF, DIXMAANG, DIXMAANH, DIXMAANI, DIXMAANJ, DIXMAANK, DIXMAANL, GENHUMPS, NONDQUAR, POWER ($n = 10000$), SPARSINE, TESTQUAD, TQUARTIC ($n = 1000$), TRIDIA). We have been investigating the distribution of the eigenvalues for these problems and, for sake of brevity, we report here only the plot of the distributions which seem to be typical and significative (in the plot we reduce the dimension in order to be able to compute and plot the spectrum of the matrices). Figure 4.1 reports a typical distribution of the eigenvalues of the Hessian matrix and the Hessian matrix preconditioned via DSPREC for the DIXMAANα problems. The effect of clustering due to preconditioning is clearly evidenced. Figure 4.2 reports the spectrum of the Hessian and the preconditioned Hessian for NONDQUAR problem ($n = 100$): the Hessian matrix presents 99 eigenvalues in

		Unpreconditioned CG				PREQN				DSPREC			
	n	it/ng	nf	CG-it	time	it/ng	nf	CG-it	time	it/ng	nf	CG-it	time
ARWHEAD	1000	6	6	6	0.46	6	6	6	0.08	7	7	7	0.09
ARWHEAD	10000	6	6	6	0.97	6	6	6	1.11	7	7	7	1.42
BDQRTIC	1000	13	13	59	0.26	14	14	59	0.37	10	10	13	0.16
BDQRTIC	10000	14	32	64	5.31	14	32	53	5.42	10	10	12	2.35
BRYBND	1000	11	11	73	0.39	11	11	71	0.51	11	11	69	0.45
BRYBND	10000	17	24	168	11.33	16	20	141	11.33	12	12	78	6.34
COSINE	1000	7	8	9	0.07	7	8	9	0.05	9	13	12	0.07
COSINE	10000	7	8	7	0.87	7	8	9	0.89	9	13	12	1.36
CRAGGLVY	1000	15	15	102	0.32	16	16	92	0.45	20	21	149	0.48
CRAGGLVY	10000	17	17	133	7.94	16	16	75	6.25	20	21	168	10.29
CURLY10	1000	17	31	8008	6.74	14	28	4806	10.84	18	25	8295	8.51
CURLY20	1000	17	31	6783	20.38	16	26	5798	24.86	18	29	6903	22.06
CURLY30	1000	19	36	7126	35.25	18	37	5955	34.95	18	37	6317	32.60
DIXMAANA	1500	7	7	8	0.12	8	8	11	0.14	8	8	8	0.15
DIXMAANA	3000	8	8	9	0.34	8	8	10	0.31	8	8	8	0.33
DIXMAANB	1500	8	8	8	0.14	8	8	8	0.14	8	8	8	0.17
DIXMAANB	3000	8	8	8	0.34	8	8	8	0.30	8	8	8	0.36
DIXMAANC	1500	9	9	9	0.15	9	9	9	0.16	9	9	9	0.17
DIXMAANC	3000	9	9	9	0.38	9	9	9	0.34	9	9	9	0.40
DIXMAAND	1500	11	11	13	0.20	11	11	12	0.20	10	10	10	0.20
DIXMAAND	3000	11	11	13	0.48	11	11	12	0.48	10	10	10	0.52
DIXMAANE	1500	10	10	188	0.55	11	11	285	1.48	9	9	9	0.16
DIXMAANE	3000	11	11	427	3.04	11	11	345	3.84	9	9	9	0.37
DIXMAANF	1500	15	19	406	1.40	18	29	482	2.64	14	14	24	0.30
DIXMAANF	3000	15	20	600	4.69	17	20	626	7.18	14	14	23	0.70
DIXMAANG	1500	15	20	450	1.57	16	23	439	2.36	14	14	23	0.30
DIXMAANG	3000	16	21	370	3.15	18	34	464	5.59	16	16	34	0.87
DIXMAANH	1500	19	27	423	1.57	18	21	426	2.32	16	21	34	0.38
DIXMAANH	3000	21	33	637	5.14	25	50	886	9.91	16	16	32	0.89
DIXMAANI	1500	11	11	3255	7.73	11	11	2988	13.68	9	9	9	0.16
DIXMAANI	3000	11	11	6218	37.81	11	11	5799	57.38	9	9	9	0.39
DIXMAANJ	1500	26	58	4069	12.60	37	132	7028	34.55	16	16	25	0.33
DIXMAANJ	3000	35	114	8621	58.41	30	83	3787	40.38	16	16	24	0.82
DIXMAANK	1500	20	43	1865	5.94	39	121	3986	19.96	16	16	24	0.34
DIXMAANK	3000	48	185	8359	57.70	38	111	9679	101.94	16	16	23	0.80
DIXMAANL	1500	21	42	3721	11.58	26	50	2988	13.38	17	17	26	0.40
DIXMAANL	3000	42	152	5505	38.57	62	234	8883	94.93	17	17	26	0.86
DQDRTIC	1000	7	7	14	0.06	6	6	9	0.07	2	2	2	0.01
DQDRTIC	10000	8	8	16	1.39	7	7	12	1.24	2	2	2	0.23
DQRTIC	1000	23	23	23	0.07	23	23	23	0.08	23	23	23	0.08
DQRTIC	10000	27	27	27	1.84	27	27	27	1.86	27	27	27	2.19
EDENSCH	1000	14	15	29	0.21	15	17	33	0.33	13	13	19	0.20
EDENSCH	10000	17	17	40	4.46	15	16	31	3.79	13	13	19	3.24

Table 4.2: Results for unpreconditioned CG, PREQN and DSPREC – Part A

	n	*Unpreconditioned CG* it/ng	nf	CG-it	time	PREQN it/ng	nf	CG-it	time	DSPREC it/ng	nf	CG-it	time
EIGENALS	930	39	57	811	21.81	32	38	688	18.99	38	52	138	8.75
ENGVAL1	1000	10	10	25	0.11	10	10	19	0.18	9	9	13	0.10
ENGVAL1	10000	10	10	20	2.13	10	10	16	2.03	9	9	13	1.95
FLETCBV2	1000	1	1	1	0.01	1	1	1	0.01	1	1	1	0.00
FLETCBV2	10000	1	1	1	0.08	1	1	1	0.08	1	1	1	0.07
FLETCHCR	1000	1484	1699	24522	32.15	1477	1683	18937	60.57	1475	1681	25207	36.67
FMINSURF	1024	39	243	6854	10.12	34	223	6162	18.58	26	167	5547	9.03
FMINSURF	5625	62	601	33799	641.33	37	285	21253	514.60	41	362	27395	538.76
FREUROTH	1000	12	17	30	0.19	12	17	25	0.26	13	18	24	0.26
FREUROTH	10000	11	16	23	2.76	11	16	18	2.68	13	18	24	3.75
GENHUMPS	1000	2154	2703	5657	24.67	1162	3204	3686	18.58	615	1696	1873	8.38
GENROSE	1000	550	1713	9388	12.69	638	2407	8289	22.48	560	1589	7528	12.52
LIARWHD	1000	16	16	23	0.18	15	15	24	0.20	12	12	20	0.13
LIARWHD	10000	17	19	23	2.91	14	14	22	2.47	13	13	17	2.44
MOREBV	1000	2	2	185	0.12	2	2	185	0.15	2	2	185	0.15
MOREBV	10000	2	2	1200	32.33	2	2	1200	34.83	2	2	1200	33.32
NCB20B	1000	18	55	975	18.82	21	69	1538	30.07	18	70	3263	59.19
NONDQUAR	1000	61	170	10793	5.38	68	203	12669	16.94	38	78	3814	2.17
NONDQUAR	10000	46	105	3222	74.90	60	154	3743	134.79	25	34	801	20.77
PENALTY1	10000	47	49	56	5.75	47	49	56	5.70	52	54	494	18.00
POWELLSG	1000	20	20	66	0.10	20	20	52	0.13	19	19	68	0.11
POWELLSG	10000	21	21	74	2.77	21	21	52	2.72	20	20	75	3.01
POWER	1000	32	32	937	0.57	32	32	296	0.51	31	37	406	0.33
POWER	10000	38	38	2608	49.88	38	38	1002	32.93	36	64	121	4.87
SCHMVETT	1000	7	7	36	0.17	7	7	29	0.19	7	7	29	0.17
SCHMVETT	10000	8	8	46	3.28	7	7	26	2.62	8	9	32	2.98
SPARSINE	1000	17	21	3729	9.10	36	205	3234	12.70	5	5	5	0.07
SPARSQUR	1000	20	20	36	0.28	20	20	27	0.29	20	20	20	0.27
SPARSQUR	10000	23	23	38	9.08	23	23	30	8.59	23	23	23	9.47
SPMSRTLS	1000	12	16	143	0.59	12	15	133	0.75	18	58	727	2.59
SPMSRTLS	10000	18	41	303	17.53	15	32	319	21.78	51	276	7436	360.34
SROSENBR	1000	8	8	9	0.04	8	8	10	0.05	9	9	11	0.05
SROSENBR	10000	8	8	9	0.79	8	8	10	0.85	9	9	11	1.02
TESTQUAD	1000	14	14	1188	0.62	14	14	1454	1.66	2	2	2	0.01
TOINTGSS	1000	5	5	9	0.05	5	5	9	0.06	3	3	3	0.04
TOINTGSS	10000	4	4	4	0.47	4	4	4	0.59	3	3	3	0.35
TQUARTIC	1000	464	469	921	3.23	16	35	26	0.13	10	11	14	0.07
TQUARTIC	10000	8	12	9	1.00	9	13	11	1.18	7	11	9	1.03
TRIDIA	1000	12	12	674	0.35	12	12	488	0.60	9	9	47	0.10
TRIDIA	10000	13	13	1910	37.93	13	13	1479	48.06	9	9	47	1.70
VARDIM	1000	19	177	18	0.11	19	177	18	0.11	19	151	179	0.20
VAREIGVL	1000	16	18	1501	4.08	32	98	393	2.11	27	73	4849	13.37
WOODS	1000	296	305	1159	2.38	47	79	136	0.45	82	100	298	0.72
WOODS	10000	283	296	1108	53.19	46	75	134	9.41	83	100	302	17.72

Table 4.3: Results for unpreconditioned CG, PREQN and DSPREC – Part B

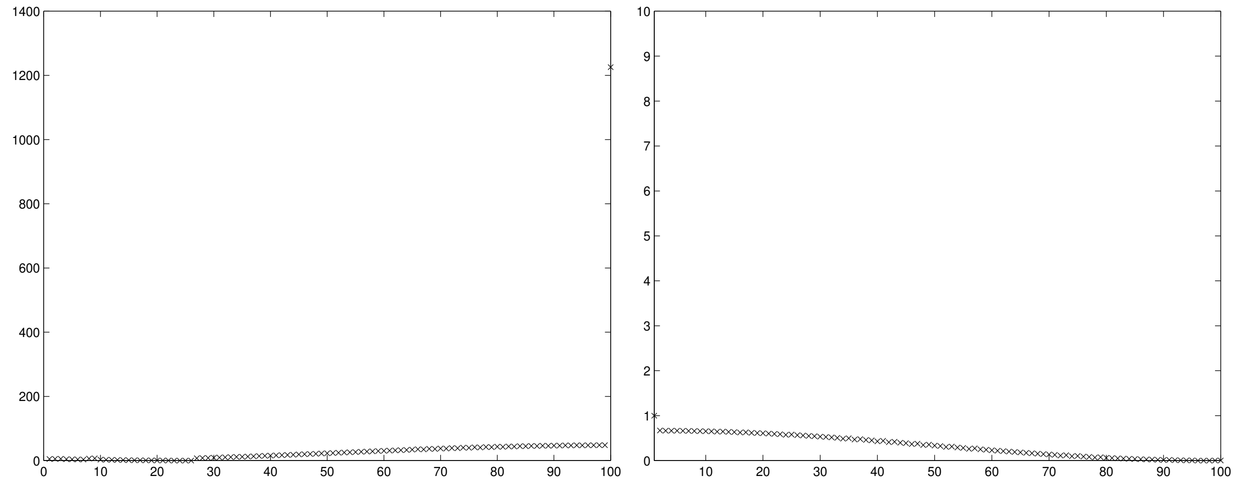

Figure 4.2: Distribution of the eigenvalues of the Hessian matrix H (on the left) and the preconditioned matrix $\widetilde{M}^{-1/2}H\widetilde{M}^{-1/2}$ (on the right) for the problem NONDQUAR $n = 100$

the range $[0, 48]$ and one eigenvalue over 10^3, while all the eigenvalues of the preconditioned Hessian matrix belong to $[0, 1]$. Finally, we report the plot of the eigenvalue distribution for the problem TESTQUAD on which the diagonal preconditioner DSPREC is really very effective. The plot reported in Figure 4.3 gives a clear justification of this. The eigenvalues distributions now reported are only examples, but they represent the typical situation which occurs whenever the preconditioner works well; in particular, the capability of clustering the eigenvalues is clearly showed. It is at the basis of the preconditioning strategy and explains the great improvements obtained on many problems compared with the unpreconditioned method.

Moreover, it is very important to note that on those problems where DSPREC leads to a reduction of the number of the inner iterations with respect to the unpreconditioned case, a saving of CPU time is always observed, too. This means that DSPREC possesses the fundamental feature of a good preconditioner, that is, the cost of building and applying it is very low, thus succeeding not only to offset the additional computational effort, but also allowing to obtain an overall computational saving. On the contrary, as regards PREQN, in many cases, even if it enables a reduction of the inner iterations, an increase of the time needed to solve the problem is observed with respect to the unpreconditioned case, due to the cost of preconditioning (see, e.g. problems CURLY10, DIXMAANI, DIXMAANL ($n = 1500$), FLETCHCR, FMINSURF ($n = 1024$), GENROSE, SPARSINE ($n = 1000$), TRIDIA ($n = 10000$)).

Furthermore, by observing Table 4.2 and Table 4.3 it is worthwhile noticing that there are problems (see e.g. CURLY30, POWER, FLETCHCR) on which both the preconditioning strategies allows to obtain a reduction of the number of inner iterations with respect to the unpreconditioned method, and moreover PREQN behaves better that DSPREC in terms of inner iterations. However, if we observe the CPU time needed to solve these problems, we discover that PREQN is more expensive, pointing out a general consideration on the fact that a preconditioning strategy more elaborate with respect to a diagonal preconditioning can lead to a great reduction of the number of inner iterations but with an heavy computational effort (note that the differences obtained on these problems in terms of CPU time are only due to the different preconditioners, since the number of iterations is nearly the same).

Finally there are four problems (NCB20B, SPMSRTLS, VARDIM, VAREIGVL) on which

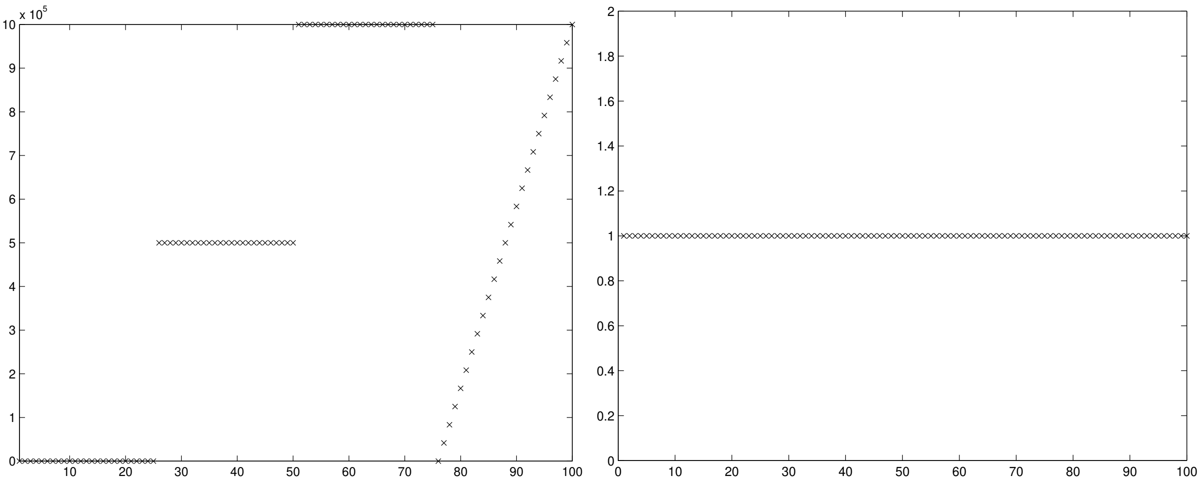

Figure 4.3: Distribution of the eigenvalues of the Hessian matrix H (on the left) and the preconditioned matrix $\widetilde{M}^{-1/2}H\widetilde{M}^{-1/2}$ (on the right) for the problem TESTQUAD $n = 100$

DSPREC has a poor behaviour in terms of inner iterations and CPU time, in comparison with the other two methods. We have been studying the distribution of the eigenvalues for those problems where the preconditioner does not work well. The plot for two typical significative situations are now reported: the spectrum of the Hessian and the Hessian matrix preconditioned via DSPREC for the problem NCB20B and for the problem SPMSRTLS are plotted in Figure 4.4 and in Figure 4.5, respectively. Figure 4.4 points out that the poor behaviour is mainly due to the fact the Hessian matrix is nearly singular. As regards Figure 4.5, it can be observed that the eigenvalues remain similarly distributed after preconditioning; in this case, the deterioration of the performance is justified by a significative increase of the condition number of the preconditioned Hessian matrix with respect to the unpreconditioned one.

The latter case considered represents an example for which the Hessian matrix is such as to give rise to the definition of a very poor preconditioner. This relies, in general, on the fact that the diagonal elements σ_j in (3.1) might be a poor approximation of the ℓ_1–norm of the j-th column vector of the Hessian matrix. When this occurs, the equilibration strategy based on columns scaling of DSPREC may be not successful in the sense that the Hessian matrix could remain badly scaled or even the conditioning could be worsened after preconditioning. This seems to be one of the main reasons of the poor behaviour of DSPREC observed is some cases.

Another interesting analysis of the results can be carried on by considering the cumulative results, that is the total number of iterations, function evaluations, inner iterations and CPU time needed to solve all the problems considered in the Tables 4.2 and 4.3. These cumulative results are reported in Table 4.4. They confirm the effectiveness of DSPREC; in fact, the method which uses the diagonal preconditioning strategy performs the best in terms of all the criteria considered. In order to complete the numerical comparison among the three algorithms, in Table 4.5 we report the number of times each algorithm performs the best in term of number of iterations, function evaluations, inner iterations and CPU time (in the comparison of very low CPU times—lower than one second—the results of two runs are reputed equal if they differ less than half a second; in the other cases, they are reputed equal if they differ by at most of 5%). This table confirms that, in most cases, the diagonal preconditioning strategy proposed in this paper produces the best results with respect to the unpreconditioned method and the automatic

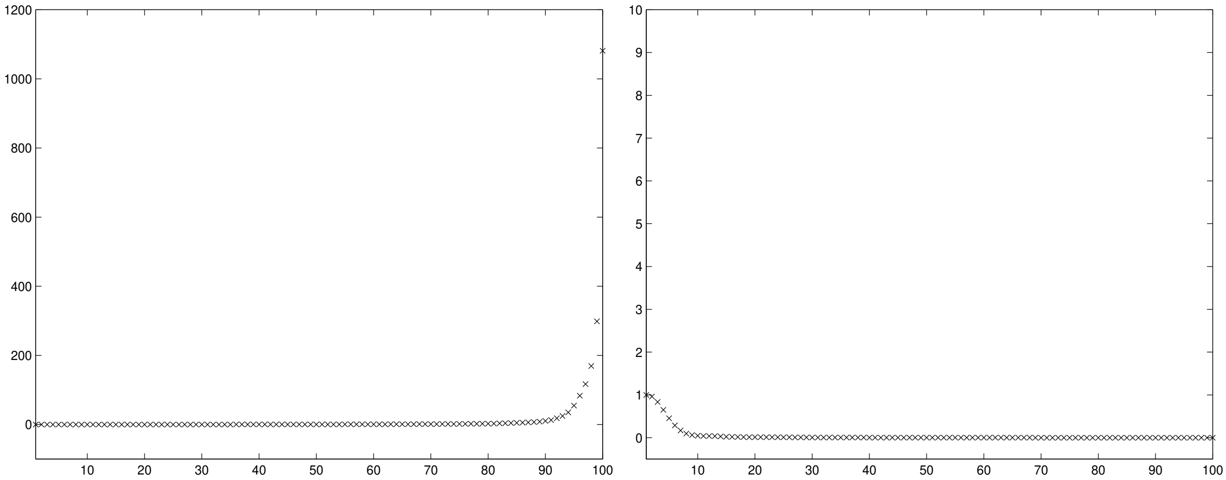

Figure 4.4: Distribution of the eigenvalues of the Hessian matrix H (on the left) and the preconditioned matrix $\widetilde{M}^{-1/2} H \widetilde{M}^{-1/2}$ (on the right) for the problem NCB20B $n = 100$

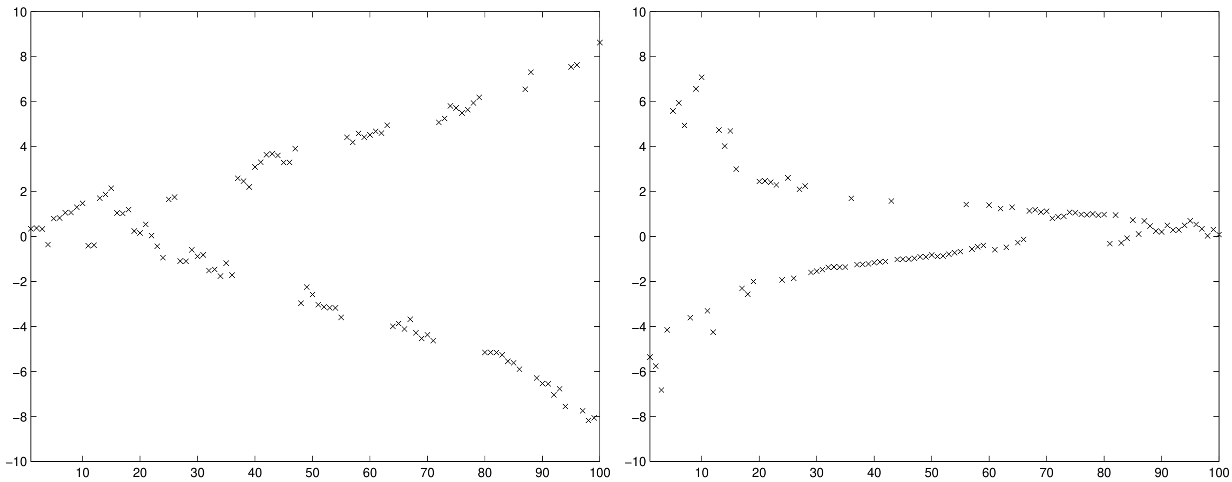

Figure 4.5: Distribution of the eigenvalues of the Hessian matrix H (on the left) and the preconditioned matrix $\widetilde{M}^{-1/2} H \widetilde{M}^{-1/2}$ (on the right) for the problem SPMSRTLS $n = 100$

	Unpreconditioned CG	PREQN	DSPREC
it/ng	6617	4832	4016
nf	10260	10712	7491
CG-it	182027	154257	114833
time	1437.95	1550.73	1279.48

Table 4.4: Cumulative results for unpreconditioned CG, PREQN and diagonal preconditioned CG

preconditioning PREQN.

	Unpreconditioned CG	PREQN	DSPREC
it/ng	24	29	55
nf	25	30	54
CG-it	18	29	54
time	16	13	33

Table 4.5: Number of times each algorithm performs the best

To conclude the numerical investigations, it is interesting to compare the behaviour of DSPREC with other two diagonal preconditioners which can be considered "ideal", in the sense they satisfy the strong theoretical properties reported in the previous section: the *Jacobi preconditioner* and the *exact diagonal column scaling* in the ℓ_1–norm. Actually, for large scale problems, they are impracticable since to build them the actual elements of the Hessian matrix must be known. However, this comparison is very interesting to know how close DSPREC is to an "ideal" preconditioner. Note that a huge computational effort is usually required to construct the preconditioner which carries out the exact diagonal column scaling, and therefore in this comparison we concentrate on the number of CG iterations needed for solving each problem without considering the CPU time. In Table 4.6 and Table 4.7 these results are reported for all the problems where the three algorithms converge to the same point, excluding those ones with 10000 variables for which the computation of the exact ℓ_1–norm of the columns of the Hessian matrix is impracticable. As it can be observed from these tables, in many cases DSPREC behaves exactly like the exact column scaling or it is very close to it, and this is at the basis of the good behaviour of DSPREC on those problems. On the overall these results reveals that the use of the diagonal of the Hessian as preconditioner would be very effective in terms of number of CG inner iterations, whenever this diagonal were available with a low computational cost; as regards DSPREC, these results point out how it closely resembles the exact diagonal column scaling, leading to a great efficiency due to the fact that DSPREC can be built very inexpensively.

5. Concluding remarks

In this paper the problem of defining preconditioning strategies for truncated Newton methods has been considered. In particular, dynamical preconditioners which use only information on the Hessian matrix obtained by the product of the Hessian matrix times a vector are investigated. Within this framework, a new preconditioning strategy is proposed based on a dynamic scaling of the Hessian matrix, aiming at obtaining a column equilibrated matrix in ℓ_1–norm.

The diagonal preconditioning strategy proposed has been embedded within a linesearch–based truncated Newton method and an extensive numerical investigation has been performed on a large set of large scale test problems. The obtained results have been compared with those obtained by means of the same method which does not use any preconditioner and by a preconditioned method which uses the automatic preconditioner based on limited quasi–Newton updating (PREQN) proposed in [23]. This comparison evidenced that the truncated Newton method which uses this new diagonal preconditioning strategy is very effective in the solution of most problems considered, performing better than the unpreconditioned method and the one which uses the automatic quasi–Newton preconditioner PREQN. It is also evident that both the preconditioners are not always beneficial; hence, once more, the experiments performed in this work pointed out the difficulty to define a "general purpose" preconditioner. Further numerical

		Jacobi preconditioner			*Exact diagonal column scaling*			DSPREC		
	n	it/ng	nf	CG-it	it/ng	nf	CG-it	it/ng	nf	CG-it
ARWHEAD	1000	7	7	7	7	7	7	7	7	7
BDQRTIC	1000	10	10	18	10	10	13	10	10	13
BRYBND	1000	10	10	32	11	11	32	11	11	69
COSINE	1000	7	8	9	7	8	9	9	13	12
CURLY10	1000	17	26	7455	17	22	7821	18	25	8295
CURLY20	1000	18	35	6678	19	32	8467	18	29	6903
CURLY30	1000	18	33	6663	18	33	5830	18	37	6317
DIXMAANA	1500	8	8	8	8	8	8	8	8	8
DIXMAANA	3000	8	8	8	8	8	8	8	8	8
DIXMAANB	1500	8	8	8	8	8	8	8	8	8
DIXMAANB	3000	8	8	8	8	8	8	8	8	8
DIXMAANC	1500	10	10	12	9	9	9	9	9	9
DIXMAANC	3000	10	10	12	9	9	9	9	9	9
DIXMAAND	1500	11	11	13	10	10	10	10	10	10
DIXMAAND	3000	11	11	13	10	10	10	10	10	10
DIXMAANE	1500	8	8	8	9	9	9	9	9	9
DIXMAANE	3000	8	8	8	9	9	9	9	9	9
DIXMAANF	1500	9	9	9	14	14	24	14	14	24
DIXMAANF	3000	9	9	9	14	14	23	14	14	23
DIXMAANG	1500	10	10	12	14	14	23	14	14	23
DIXMAANG	3000	10	10	10	17	18	39	16	16	34
DIXMAANH	1500	11	11	14	16	21	34	16	21	34
DIXMAANH	3000	11	11	14	16	16	32	16	16	32
DIXMAANI	1500	8	8	8	9	9	9	9	9	9
DIXMAANI	3000	8	8	8	9	9	9	9	9	9
DIXMAANJ	1500	10	10	11	16	16	25	16	16	25
DIXMAANJ	3000	10	10	10	16	16	24	16	16	24
DIXMAANK	1500	10	10	10	16	16	24	16	16	24
DIXMAANK	3000	10	10	10	16	16	23	16	16	23
DIXMAANL	1500	11	11	11	17	17	26	17	17	26
DIXMAANL	3000	11	11	11	17	17	26	17	17	26
DQDRTIC	1000	2	2	2	2	2	2	2	2	2
DQRTIC	1000	23	23	23	23	23	23	23	23	23
EDENSCH	1000	13	13	19	13	13	21	13	13	19

Table 4.6: Results for Jacobi preconditioner, exact diagonal column scaling, DSPREC – Part A

	n	*Jacobi preconditioner* it/ng	nf	CG-it	*Exact diagonal column scaling* it/ng	nf	CG-it	DSPREC it/ng	nf	CG-it
EIGENALS	930	31	45	143	37	52	129	38	52	138
ENGVAL1	1000	9	9	13	9	9	13	9	9	13
FLETCBV2	1000	1	1	1	1	1	1	1	1	1
FLETCHCR	1000	1477	1683	23808	1479	1689	23795	1475	1681	25207
FMINSURF	1024	1412	2915	2520	41	378	2557	26	167	5547
FREUROTH	1000	10	15	18	11	16	17	13	18	24
GENHUMPS	1000	306	716	585	1142	1429	2306	615	1696	1873
GENROSE	1000	711	2350	6262	669	1853	6035	560	1589	7528
LIARWHD	1000	12	12	21	11	11	18	12	12	20
MOREBV	1000	2	2	188	2	2	186	2	2	185
NCB20B	1000	23	90	3165	18	66	3338	18	70	3263
NONDIA	1000	776	807	1542	36	58	50	36	58	50
NONDQUAR	1000	41	91	2903	33	66	2877	38	78	3814
PENALTY1	1000	42	44	70	50	57	323	49	55	324
POWELLSG	1000	19	19	63	19	19	60	19	19	68
POWER	1000	31	37	36	31	37	225	31	37	406
QUARTC	1000	23	23	23	23	23	23	23	23	23
SCHMVETT	1000	7	7	26	8	8	42	7	7	29
SINQUAD	1000	111	168	298	393	427	1175	89	157	221
SPARSINE	1000	15	18	2433	12	12	1465	5	5	5
SPARSQUR	1000	20	20	50	20	20	20	20	20	20
SPMSRTLS	1000	17	35	132	11	12	119	18	58	727
SROSENBR	1000	9	10	11	9	9	11	9	9	11
TESTQUAD	1000	2	2	2	2	2	2	2	2	2
TOINTGSS	1000	7	7	10	7	7	10	3	3	3
TQUARTIC	1000	15	20	21	15	23	21	10	11	14
TRIDIA	1000	11	11	46	11	11	46	9	9	47
VARDIM	1000	28	101	1340	19	151	184	19	151	179
VAREIGVL	1000	15	17	628	15	15	736	27	73	4849
WOODS	1000	48	58	155	83	104	302	82	100	298

Table 4.7: Results for Jacobi preconditioner, exact diagonal column scaling, DSPREC – Part B

experiences showed that the behaviour of the new diagonal preconditioner is very closely related to the "ideal" exact diagonal column scaling.

Even if no final conclusion can be drawn, on the whole, the new preconditioning strategy proposed in this paper seems to be very efficient and we believe that it could be successfully used for defining efficient truncated Newton methods for the solution of large scale unconstrained problems. Moreover, some points are still worthwhile investigating: to devise adaptive rules to decide when the preconditioner should be used based on information gained from the CG iterations, getting inspiration from [37]; to exploit the information obtained from the sequence of the matrix vector products involving the Hessian computed at each CG iteration to improve the preconditioner, like, e.g. in [24]; to combine the use of this diagonal preconditioner with other preconditioners, or to use it to initialize other preconditioners.

Finally, we think that the definition of an effective preconditioning strategy is strictly connected with the choice of an efficient truncation criterion for the inner iterations which is still a challenging topic for the research in the truncated Newton methods.

Appendix

In this Appendix we report the complete numerical results obtained by the linesearch based truncated Newton method which uses

- the unpreconditioned CG (Tables A.1 and A.2),
- the preconditioner PREQN [23] (Tables A.3 and A.4)
- the diagonal preconditioning strategy proposed in this paper (Tables A.5 and A.6).

All the results are reported in terms of number of outer iterations (it) —which is the same as the number of gradient evaluation (ng)— number of function evaluations (nf), total number of inner iterations (CG-it) and CPU time (time); moreover the optimal function value obtained is reported ($f^\star$). Finally, for the methods which uses a preconditioner, the additional time due to preconditioning, i.e. the total time needed to computed the preconditioned residual at each inner iteration is reported.

		it/ng	nf	CG-it	$f^\star$	time
ARWHEAD	1000	6	6	6	1.690288D-10	0.46
ARWHEAD	10000	6	6	6	0.000000D+00	0.97
BDQRTIC	1000	13	13	59	3.983818D+03	0.26
BDQRTIC	10000	14	32	64	4.003431D+04	5.31
BROYDN7D	1000	76	232	550	3.746532D+02	1.42
BROYDN7D	10000	590	2053	4480	3.714410D+03	250.90
BRYBND	1000	11	11	73	1.816541D-13	0.39
BRYBND	10000	17	24	168	4.353566D-12	11.33
CHAINWOO	1000	139	470	7667	6.362471D+01	11.74
CHAINWOO	10000	–	–	–	–	> 900
COSINE	1000	7	8	9	-9.990000D+02	0.07
COSINE	10000	7	8	7	-9.999000D+03	0.87
CRAGGLVY	1000	15	15	102	3.364231D+02	0.32
CRAGGLVY	10000	17	17	133	3.377956D+03	7.94
CURLY10	1000	17	31	8008	-1.003163D+05	6.74
CURLY10	10000	–	–	–	–	> 900
CURLY20	1000	17	31	6783	-1.003163D+05	20.38
CURLY20	10000	–	–	–	–	> 900
CURLY30	1000	19	36	7126	-1.003163D+05	35.25
CURLY30	10000	–	–	–	–	> 900
DIXMAANA	1500	7	7	8	1.000000D+00	0.12
DIXMAANA	3000	8	8	9	1.000000D+00	0.34
DIXMAANB	1500	8	8	8	1.000000D+00	0.14
DIXMAANB	3000	8	8	8	1.000000D+00	0.34
DIXMAANC	1500	9	9	9	1.000000D+00	0.15
DIXMAANC	3000	9	9	9	1.000000D+00	0.38
DIXMAAND	1500	11	11	13	1.000000D+00	0.20
DIXMAAND	3000	11	11	13	1.000000D+00	0.48
DIXMAANE	1500	10	10	188	1.000000D+00	0.55
DIXMAANE	3000	11	11	427	1.000000D+00	3.04
DIXMAANF	1500	15	19	406	1.000000D+00	1.40
DIXMAANF	3000	15	20	600	1.000000D+00	4.69
DIXMAANG	1500	15	20	450	1.000000D+00	1.57
DIXMAANG	3000	16	21	370	1.000000D+00	3.15
DIXMAANH	1500	19	27	423	1.000000D+00	1.57
DIXMAANH	3000	21	33	637	1.000000D+00	5.14
DIXMAANI	1500	11	11	3255	1.000000D+00	7.73
DIXMAANI	3000	11	11	6218	1.000000D+00	37.81
DIXMAANJ	1500	26	58	4069	1.000000D+00	12.60
DIXMAANJ	3000	35	114	8621	1.000000D+00	58.41
DIXMAANK	1500	20	43	1865	1.000000D+00	5.94
DIXMAANK	3000	48	185	8359	1.000000D+00	57.70
DIXMAANL	1500	21	42	3721	1.000000D+00	11.58
DIXMAANL	3000	42	152	5505	1.000000D+00	38.57
DQDRTIC	1000	7	7	14	2.103314D-36	0.06
DQDRTIC	10000	8	8	16	3.894166D-36	1.39
DQRTIC	1000	23	23	22	1.184145D-01	0.07
DQRTIC	10000	27	27	26	1.814550D+01	1.84
EDENSCH	1000	14	15	29	6.003285D+03	0.21
EDENSCH	10000	17	17	40	6.000328D+04	4.46
EIGENALS	930	39	57	811	6.170301D-06	21.81
EIGENBLS	930	170	327	23371	6.788592D-06	711.00
ENGVAL1	1000	10	10	25	1.108195D+03	0.11
ENGVAL1	10000	10	10	20	1.109926D+04	2.13
FLETCBV2	1000	1	1	1	-5.013384D-01	0.01
FLETCBV2	10000	1	1	1	-5.001341D-01	0.08
FLETCBV3	1000	277	277	281	-1.256304D+05	3.47
FLETCBV3	10000	> 3000	–	–	–	–

Table A.1: Results for unpreconditioned CG – part A

		it/ng	nf	CG-it	$f^\star$	time
FLETCHCR	1000	1484	1699	24522	1.855475D-11	32.15
FLETCHCR	10000	–	–	–	–	> 900
FMINSURF	1024	39	243	6854	1.000000D+00	10.12
FMINSURF	5625	62	601	33799	1.000000D+00	641.33
FREUROTH	1000	12	17	30	1.214697D+05	0.19
FREUROTH	10000	11	16	23	1.216521D+06	2.76
GENHUMPS	1000	2154	2703	5657	2.165901D-10	24.67
GENHUMPS	10000	> 3000	–	–	–	–
GENROSE	1000	550	1713	9388	1.000000D+00	12.69
GENROSE	10000	–	–	–	–	> 900
LIARWHD	1000	16	16	23	1.347339D-11	0.18
LIARWHD	10000	17	19	23	3.673645D-09	2.91
MOREBV	1000	2	2	185	2.784355D-09	0.12
MOREBV	10000	2	2	1200	2.402995D-12	32.33
MSQRTALS	1024	26	69	5122	6.687495D-08	148.33
MSQRTBLS	1024	30	100	6415	4.590134D-12	184.08
NCB20	1010	56	103	510	9.146927D+02	13.73
NCB20B	1000	18	55	975	1.676011D+03	18.82
NONCVXUN	1000	148	410	8231	2.341600D+03	10.47
NONCVXUN	10000	–	–	–	–	> 900
NONCVXU2	1000	184	567	2799	2.316926D+03	4.54
NONCVXU2	10000	–	–	–	–	> 900
NONDIA	1000	7	7	8	5.328525D-12	0.06
NONDIA	10000	5	5	5	5.777204D-10	0.71
NONDQUAR	1000	61	170	10793	7.797202D-07	5.38
NONDQUAR	10000	46	105	3222	1.406633D-05	74.90
PENALTY1	1000	41	43	52	9.686175D-03	0.23
PENALTY1	10000	47	49	56	9.900151D-02	5.75
POWELLSG	1000	20	20	66	2.308946D-08	0.10
POWELLSG	10000	21	21	74	4.560881D-08	2.77
POWER	1000	32	32	937	6.719134D-10	0.57
POWER	10000	38	38	2608	7.236558D-10	49.88
QUARTC	1000	23	23	22	1.184145D-01	0.07
QUARTC	10000	27	27	26	1.814550D+01	1.86
SCHMVETT	1000	7	7	36	-2.994000D+03	0.17
SCHMVETT	10000	8	8	46	-2.999400D+04	3.28
SINQUAD	1000	> 3000	–	–	–	–
SINQUAD	10000	> 3000	–	–	–	–
SPARSINE	1000	17	21	3729	2.120104D-11	9.10
SPARSINE	10000	–	–	–	–	> 900
SPARSQUR	1000	20	20	36	1.277448D-08	0.28
SPARSQUR	10000	23	23	38	1.004828D-08	9.08
SPMSRTLS	1000	12	16	143	1.775335D-10	0.59
SPMSRTLS	10000	18	41	303	5.384487D-08	17.53
SROSENBR	1000	8	8	9	3.831667D-09	0.04
SROSENBR	10000	8	8	9	3.831667D-08	0.79
TESTQUAD	1000	14	14	1188	7.998922D-18	0.62
TOINTGSS	1000	5	5	9	1.001002D+01	0.05
TOINTGSS	10000	4	4	4	1.000100D+01	0.47
TQUARTIC	1000	464	469	921	5.441991D-08	3.23
TQUARTIC	10000	8	12	9	7.172640D-05	1.00
TRIDIA	1000	12	12	674	1.166811D-16	0.35
TRIDIA	10000	13	13	1910	1.780044D-14	37.93
VARDIM	1000	19	177	18	1.163681D-25	0.11
VARDIM	10000	–	–	–	–	> 900
VAREIGVL	1000	16	18	1501	6.995798D-10	4.08
VAREIGVL	10000	16	16	1212	2.921130D-09	63.30
WOODS	1000	296	305	1159	4.389618D-14	2.38
WOODS	10000	283	296	1108	6.200442D-09	53.19

Table A.2: Results for unpreconditioned CG – Part B

		it/ng	nf	CG-it	$f^\star$	add. time	time
ARWHEAD	1000	6	6	6	1.690288D-10	0.01	0.08
ARWHEAD	10000	6	6	6	0.000000D+00	0.02	1.11
BDQRTIC	1000	14	14	59	3.983818D+03	0.04	0.37
BDQRTIC	10000	14	32	53	4.003431D+04	0.47	5.42
BROYDN7D	1000	80	266	750	3.851963D+02	0.67	3.00
BROYDN7D	10000	646	2518	6525	3.801124D+03	99.80	434.13
BRYBND	1000	11	11	71	1.505758D-16	0.04	0.51
BRYBND	10000	16	20	141	1.968983D-16	1.84	11.33
CHAINWOO	1000	138	408	7396	6.362471D+01	6.45	23.71
CHAINWOO	10000	–	–	–	–	–	> 900
COSINE	1000	7	8	9	-9.990000D+02	0.02	0.05
COSINE	10000	7	8	7	-9.999000D+03	0.02	0.89
CRAGGLVY	1000	16	16	92	3.364231D+02	0.09	0.45
CRAGGLVY	10000	16	16	75	3.377956D+03	0.85	6.25
CURLY10	1000	14	28	4806	-1.003163D+05	3.48	10.84
CURLY10	10000	–	–	–	–	–	> 900
CURLY20	1000	16	26	5798	-1.003163D+05	4.30	24.86
CURLY20	10000	–	–	–	–	–	> 900
CURLY30	1000	18	37	5955	-1.003163D+05	4.38	34.95
CURLY30	10000	–	–	–	–	–	> 900
DIXMAANA	1500	8	8	11	1.000000D+00	0.01	0.14
DIXMAANA	3000	8	8	10	1.000000D+00	0.00	0.31
DIXMAANB	1500	8	8	8	1.000000D+00	0.00	0.14
DIXMAANB	3000	8	8	8	1.000000D+00	0.00	0.30
DIXMAANC	1500	9	9	9	1.000000D+00	0.02	0.16
DIXMAANC	3000	9	9	9	1.000000D+00	0.02	0.34
DIXMAAND	1500	11	11	12	1.000000D+00	0.01	0.20
DIXMAAND	3000	11	11	12	1.000000D+00	0.01	0.48
DIXMAANE	1500	11	11	285	1.000000D+00	0.51	1.48
DIXMAANE	3000	11	11	345	1.000000D+00	1.63	3.84
DIXMAANF	1500	18	29	482	1.000000D+00	0.66	2.64
DIXMAANF	3000	17	20	626	1.000000D+00	2.78	7.18
DIXMAANG	1500	16	23	439	1.000000D+00	0.66	2.36
DIXMAANG	3000	18	34	464	1.000000D+00	2.13	5.59
DIXMAANH	1500	18	21	426	1.000000D+00	0.59	2.32
DIXMAANH	3000	25	50	886	1.000000D+00	2.75	9.91
DIXMAANI	1500	11	11	2988	1.000000D+00	5.02	13.68
DIXMAANI	3000	11	11	5799	1.000000D+00	22.05	57.38
DIXMAANJ	1500	37	132	7028	1.000000D+00	11.11	34.55
DIXMAANJ	3000	30	83	3787	1.000000D+00	14.10	40.38
DIXMAANK	1500	39	121	3986	1.000000D+00	6.00	19.96
DIXMAANK	3000	38	111	9679	1.000000D+00	37.73	101.94
DIXMAANL	1500	26	50	2988	1.000000D+00	2.93	13.38
DIXMAANL	3000	62	234	8883	1.000000D+00	32.73	94.93
DQDRTIC	1000	6	6	9	1.148421D-26	0.02	0.07
DQDRTIC	10000	7	7	12	5.053195D-29	0.05	1.24
DQRTIC	1000	23	23	22	1.184145D-01	0.02	0.08
DQRTIC	10000	27	27	26	1.814550D+01	0.03	1.86
EDENSCH	1000	15	17	33	6.003285D+03	0.02	0.33
EDENSCH	10000	15	16	31	6.000328D+04	0.16	3.79
EIGENALS	930	32	38	688	5.439341D-09	0.39	18.99
EIGENBLS	930	–	–	–	–	–	> 900
ENGVAL1	1000	10	10	19	1.108195D+03	0.00	0.18
ENGVAL1	10000	10	10	16	1.109926D+04	0.06	2.03
FLETCBV2	1000	1	1	1	-5.013384D-01	0.00	0.01
FLETCBV2	10000	1	1	1	-5.001341D-01	0.00	0.08
FLETCBV3	1000	240	240	246	-4.997924D+04	0.11	3.01
FLETCBV3	10000	> 3000	–	–	–	–	–

Table A.3: Results for PREQN - Part A

		it/ng	nf	CG-it	f^*	add. time	time
FLETCHCR	1000	1477	1683	18937	3.175229D-13	20.62	60.57
FLETCHCR	10000	–	–	–	–	–	> 900
FMINSURF	1024	34	223	6162	1.000000D+00	4.38	18.58
FMINSURF	5625	37	285	21253	1.000000D+00	120.70	514.60
FREUROTH	1000	12	17	25	1.214697D+05	0.00	0.26
FREUROTH	10000	11	16	18	1.216521D+06	0.09	2.68
GENHUMPS	1000	1162	3204	3686	4.701924D-11	1.97	18.58
GENHUMPS	10000	> 3000	–	–	–	–	–
GENROSE	1000	638	2407	8289	1.000000D+00	5.38	22.48
GENROSE	10000	–	–	–	–	–	> 900
LIARWHD	1000	15	15	24	2.580933D-22	0.00	0.20
LIARWHD	10000	14	14	22	2.301803D-16	0.14	2.47
MOREBV	1000	2	2	185	2.784355D-09	0.02	0.15
MOREBV	10000	2	2	1200	2.402995D-12	3.16	34.83
MSQRTALS	1024	29	71	8924	9.220249D-13	6.65	260.70
MSQRTBLS	1024	27	74	4601	2.072442D-12	3.34	135.85
NCB20	1010	602	5486	1136	9.220346D+02	0.57	100.54
NCB20B	1000	21	69	1538	1.676011D+03	1.07	30.07
NONCVXUN	1000	108	356	5415	2.321925D+03	2.52	12.31
NONCVXUN	10000	–	–	–	–	–	> 900
NONCVXU2	1000	106	335	1832	2.316867D+03	1.36	5.16
NONCVXU2	10000	387	1297	9554	2.316820D+04	117.70	425.77
NONDIA	1000	7	7	9	3.770613D-22	0.01	0.08
NONDIA	10000	5	5	5	5.756219D-10	0.01	0.66
NONDQUAR	1000	68	203	12669	5.725976D-07	8.20	16.94
NONDQUAR	10000	60	154	3743	7.411955D-06	50.32	134.79
PENALTY1	1000	41	43	52	9.686175D-03	0.02	0.25
PENALTY1	10000	47	49	56	9.900151D-02	0.09	5.70
POWELLSG	1000	20	20	52	2.661025D-08	0.00	0.13
POWELLSG	10000	21	21	52	5.288469D-08	0.41	2.72
POWER	1000	32	32	296	8.390397D-10	0.24	0.51
POWER	10000	38	38	1002	8.562759D-10	12.45	32.93
QUARTC	1000	23	23	22	1.184145D-01	0.01	0.12
QUARTC	10000	27	27	26	1.814550D+01	0.05	1.91
SCHMVETT	1000	7	7	29	-2.994000D+03	0.02	0.19
SCHMVETT	10000	7	7	26	-2.999400D+04	0.20	2.62
SINQUAD	1000	208	1936	297	6.945483D-09	0.16	3.44
SINQUAD	10000	110	252	266	4.666074D-07	1.68	29.07
SPARSINE	1000	36	205	3234	6.770181D-13	2.53	12.70
SPARSINE	10000	–	–	–	–	–	> 900
SPARSQUR	1000	20	20	27	1.277426D-08	0.01	0.29
SPARSQUR	10000	23	23	30	1.004822D-08	0.14	8.59
SPMSRTLS	1000	12	15	133	6.401210D-10	0.12	0.75
SPMSRTLS	10000	15	32	319	5.693204D-12	4.33	21.78
SROSENBR	1000	8	8	10	6.487883D-11	0.01	0.05
SROSENBR	10000	8	8	10	6.487883D-10	0.04	0.85
TESTQUAD	1000	14	14	1454	1.607105D-17	0.78	1.66
TOINTGSS	1000	5	5	9	1.001002D+01	0.00	0.06
TOINTGSS	10000	4	4	4	1.000100D+01	0.00	0.59
TQUARTIC	1000	16	35	26	3.070101D-18	0.01	0.13
TQUARTIC	10000	9	13	11	2.408842D-13	0.04	1.18
TRIDIA	1000	12	12	488	6.306145D-17	0.24	0.60
TRIDIA	10000	13	13	1479	1.849585D-14	19.95	48.06
VARDIM	1000	19	177	18	1.163681D-25	0.01	0.11
VARDIM	10000	–	–	–	–	–	> 900
VAREIGVL	1000	32	98	363	8.786995D-10	0.34	2.11
VAREIGVL	10000	16	16	924	2.023605D-09	11.95	60.78
WOODS	1000	47	79	136	6.194550D-17	0.02	0.45
WOODS	10000	46	75	134	2.051012D-09	1.21	9.41

Table A.4: Results for PREQN – Part B

		it/ng	nf	CG-it	$f^\star$	add. time	time
ARWHEAD	1000	7	7	7	0.000000D+00	0.04	0.09
ARWHEAD	10000	7	7	7	0.000000D+00	0.52	1.42
BDQRTIC	1000	10	10	13	3.983818D+03	0.04	0.16
BDQRTIC	10000	10	10	12	4.003431D+04	0.75	2.35
BROYDN7D	1000	93	285	799	3.584696D+02	0.56	2.08
BROYDN7D	10000	641	1980	5469	3.666719D+03	59.01	314.44
BRYBND	1000	11	11	69	2.161374D-15	0.14	0.45
BRYBND	10000	12	12	78	1.124742D-16	1.25	6.34
CHAINWOO	1000	175	647	18592	1.000000D+00	3.17	28.94
CHAINWOO	10000	–	–	–	–	–	> 900
COSINE	1000	9	13	12	-9.990000D+02	0.01	0.07
COSINE	10000	9	13	12	-9.999000D+03	0.48	1.36
CRAGGLVY	1000	20	21	149	3.364231D+02	0.13	0.48
CRAGGLVY	10000	20	21	168	3.377956D+03	1.87	10.29
CURLY10	1000	18	25	8295	-1.003163D+05	1.02	8.51
CURLY10	10000	–	–	–	–	–	> 900
CURLY20	1000	18	29	6903	-1.003163D+05	1.02	22.06
CURLY20	10000	–	–	–	–	–	> 900
CURLY30	1000	18	37	6317	-1.003163D+05	1.20	32.60
CURLY30	10000	–	–	–	–	–	> 900
DIXMAANA	1500	8	8	8	1.000000D+00	0.05	0.15
DIXMAANA	3000	8	8	8	1.000000D+00	0.12	0.33
DIXMAANB	1500	8	8	8	1.000000D+00	0.05	0.17
DIXMAANB	3000	8	8	8	1.000000D+00	0.12	0.36
DIXMAANC	1500	9	9	9	1.000000D+00	0.06	0.17
DIXMAANC	3000	9	9	9	1.000000D+00	0.12	0.40
DIXMAAND	1500	10	10	10	1.000000D+00	0.08	0.20
DIXMAAND	3000	10	10	10	1.000000D+00	0.23	0.52
DIXMAANE	1500	9	9	9	1.000000D+00	0.06	0.16
DIXMAANE	3000	9	9	9	1.000000D+00	0.13	0.37
DIXMAANF	1500	14	14	24	1.000000D+00	0.11	0.30
DIXMAANF	3000	14	14	23	1.000000D+00	0.27	0.70
DIXMAANG	1500	14	14	23	1.000000D+00	0.12	0.30
DIXMAANG	3000	16	16	34	1.000000D+00	0.28	0.87
DIXMAANH	1500	16	21	34	1.000000D+00	0.12	0.38
DIXMAANH	3000	16	16	32	1.000000D+00	0.27	0.89
DIXMAANI	1500	9	9	9	1.000000D+00	0.05	0.16
DIXMAANI	3000	9	9	9	1.000000D+00	0.15	0.39
DIXMAANJ	1500	16	16	25	1.000000D+00	0.10	0.33
DIXMAANJ	3000	16	16	24	1.000000D+00	0.26	0.82
DIXMAANK	1500	16	16	24	1.000000D+00	0.16	0.34
DIXMAANK	3000	16	16	23	1.000000D+00	0.31	0.80
DIXMAANL	1500	17	17	26	1.000000D+00	0.20	0.40
DIXMAANL	3000	17	17	26	1.000000D+00	0.34	0.86
DQDRTIC	1000	2	2	2	0.000000D+00	0.01	0.01
DQDRTIC	10000	2	2	2	0.000000D+00	0.05	0.23
DQRTIC	1000	23	23	23	6.334892D-02	0.02	0.08
DQRTIC	10000	27	27	27	9.709868D+00	0.63	2.19
EDENSCH	1000	13	13	19	6.003285D+03	0.10	0.20
EDENSCH	10000	13	13	19	6.000328D+04	1.06	3.24
EIGENALS	930	38	52	138	3.695505D-06	2.77	8.75
EIGENBLS	930	–	–	–	–	–	> 900
ENGVAL1	1000	9	9	13	1.108195D+03	0.02	0.10
ENGVAL1	10000	9	9	13	1.109926D+04	0.66	1.95
FLETCBV2	1000	1	1	1	-5.013384D-01	0.00	0.00
FLETCBV2	10000	1	1	1	-5.001341D-01	0.00	0.07
FLETCBV3	1000	43	43	54	-2.053162D+04	0.22	0.72
FLETCBV3	10000	241	241	360	-8.889024D+08	17.88	52.34

Table A.5: Result for diagonal preconditioning – Part A

		it/ng	nf	CG-it	$f^\star$	add. time	time
FLETCHCR	1000	1475	1681	25207	1.156621D-11	7.71	36.67
FLETCHCR	10000	–	–	–	–	–	> 900
FMINSURF	1024	26	167	5547	1.000000D+00	0.63	9.03
FMINSURF	5625	41	362	27395	1.000000D+00	33.95	538.76
FREUROTH	1000	13	18	24	1.214697D+05	0.04	0.26
FREUROTH	10000	13	18	24	1.216521D+06	1.30	3.75
GENHUMPS	1000	615	1696	1873	4.611225D-12	2.65	8.38
GENHUMPS	10000	558	1422	1344	1.061312D-14	36.21	122.92
GENROSE	1000	560	1589	7528	1.000000D+00	2.92	12.52
GENROSE	10000	–	–	–	–	–	> 900
LIARWHD	1000	12	12	20	5.263256D-24	0.05	0.13
LIARWHD	10000	13	13	17	5.767645D-18	0.78	2.44
MOREBV	1000	2	2	185	2.833140D-09	0.02	0.15
MOREBV	10000	2	2	1200	2.402999D-12	2.15	33.32
MSQRTALS	1024	–	–	–	–	–	> 900
MSQRTBLS	1024	–	–	–	–	–	> 900
NCB20	1010	610	5096	1502	9.183763D+02	32.23	114.38
NCB20B	1000	18	70	3263	1.676011D+03	1.47	59.19
NONCVXUN	1000	317	1235	4503	2.335523D+03	1.37	8.00
NONCVXUN	10000	> 3000	–	–	–	–	–
NONCVXU2	1000	344	1417	2575	2.317411D+03	1.03	5.95
NONCVXU2	10000	> 3000	–	–	–	–	–
NONDIA	1000	36	58	50	9.898969D-01	0.17	0.40
NONDIA	10000	66	138	95	9.898969D-01	3.90	12.70
NONDQUAR	1000	38	78	3814	1.950578D-06	0.45	2.82
NONDQUAR	10000	25	34	801	1.000643D-05	2.27	20.77
PENALTY1	1000	49	55	324	9.686176D-03	0.09	0.51
PENALTY1	10000	52	54	494	9.900151D-02	3.18	18.00
POWELLSG	1000	19	19	68	3.700970D-08	0.02	0.11
POWELLSG	10000	20	20	75	7.310397D-08	0.66	3.01
POWER	1000	31	37	406	1.938946D-10	0.06	0.33
POWER	10000	36	64	121	5.357454D-11	1.13	4.87
QUARTC	1000	23	23	22	6.334892D-02	0.02	0.09
QUARTC	10000	27	27	26	9.709868D+00	0.57	2.12
SCHMVETT	1000	7	7	29	-2.994000D+03	0.02	0.17
SCHMVETT	10000	8	9	32	-2.999400D+04	0.76	2.98
SINQUAD	1000	89	157	221	9.696874D-08	0.49	1.32
SINQUAD	10000	1813	1879	5498	5.691653D-06	142.11	506.29
SPARSINE	1000	5	5	5	1.925053D-19	0.02	0.07
SPARSINE	10000	5	5	5	1.923320D-17	0.70	1.87
SPARSQUR	1000	20	20	20	5.828555D-09	0.06	0.27
SPARSQUR	10000	23	23	23	4.488231D-09	3.59	9.47
SPMSRTLS	1000	18	58	727	1.892078D-09	0.34	2.59
SPMSRTLS	10000	51	276	7436	7.180795D-09	17.81	360.34
SROSENBR	1000	9	9	11	7.511199D-13	0.00	0.05
SROSENBR	10000	9	9	11	7.511199D-12	0.28	1.02
TESTQUAD	1000	2	2	2	6.625563D-25	0.00	0.01
TOINTGSS	1000	3	3	3	1.001002D+01	0.00	0.04
TOINTGSS	10000	3	3	3	1.000100D+01	0.09	0.35
TQUARTIC	1000	10	11	14	2.377308D-18	0.02	0.07
TQUARTIC	10000	7	11	9	5.094582D-16	0.41	1.03
TRIDIA	1000	9	9	47	2.945676D-18	0.02	0.10
TRIDIA	10000	9	9	47	2.734246D-18	0.27	1.70
VARDIM	1000	19	151	179	2.102924D-11	0.03	0.20
VARDIM	10000	233	5013	926	1.929873D-12	8.68	77.64
VAREIGVL	1000	27	73	4849	2.949849D-11	0.77	13.37
VAREIGVL	10000	–	–	–	–	–	> 900
WOODS	1000	82	100	298	9.043476D-11	0.15	0.72
WOODS	10000	83	100	302	1.375305D-18	4.24	17.72

Table A.6: Results for diagonal preconditioning – Part B

Acknowledgments

The author wish to thank Stefano Lucidi for his many constructive comments and suggestions. Thanks also to Giovanni Fasano for helpful discussions and for carefully reading a preliminary version of this paper.

References

[1] B. Averick, R. Carter, J. Moré, and G. Xue, "The MINPACK-2 test problems collection," preprint MCS-P153-0692, Argonne National Laboratoty, Argonne, IL, 1992.

[2] O. Axelsson, *Iterative solution methods.* Cambridge: Cambridge University Press, 1994.

[3] J. Barlow and G. Toraldo, "The effect of diagonal scaling on projected gradient methods for bound constrained quadratic programming problems," *Optimization Methods and Software*, vol. 5, pp. 235–245, 1995.

[4] R. Barret, M. Berry, T. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. Van der Vorst, *Templates for the solution of linear systems: building blocks for iterative methods.* Philadelphia, PA: SIAM, 1994.

[5] F. Bauer, "Optimally scaled matrices," *Numerische Mathematik*, vol. 5, pp. 73–87, 1963.

[6] I. Bongartz, A. Conn, N. I. M. Gould, and P. L. Toint, "CUTE: Constrained and unconstrained testing environment," *ACM Transaction on Mathematical Software*, vol. 21, pp. 123–160, 1995.

[7] A. Bouaricha, J. Moré, and Z. Wu, "Preconditioning Newton's method," Technical Report CRPC–TR98762, Center for Research on Parallel Computing, Rice University, Houston, TX, 1998.

[8] J. Bunch, "Equilibration of symmetric matrices in the max-norm," *Journal of ACM*, vol. 18, pp. 566–572, 1971.

[9] M. J. Daydé, J.-Y. L'Excellent, and N. I. M. Gould, "Element–by–element preconditioners for large–scale partially separable optimization problems," *SIAM Journal on Scientific Computing*, vol. 18, pp. 1767–1787, 1997.

[10] R. Dembo, S. Eisenstat, and T. Steihaug, "Inexact Newton methods," *SIAM Journal on Numerical Analysis*, vol. 19, pp. 400–408, 1982.

[11] R. Dembo and T. Steihaug, "Truncated-Newton algorithms for large-scale unconstrained optimization," *Mathematical Programming*, vol. 26, pp. 190–212, 1983.

[12] I. S. Duff, A. M. Erisman, and J. K. Reid, *Direct methods for sparse matrices.* London: Oxford University Press, 1986.

[13] G. Golub and C. Van Loan, *Matrix Computations.* Baltimore: The John Hopkins Press, 1996. Third edition.

[14] A. Greenbaum, *Iterative methods for solving linear systems.* Philadelphia, PA: SIAM, 1997.

[15] A. Griewank, "On automatic differentiation," in *Mathematical programming: Recent Developments and Applications* (M. Iri and K. Tanake, eds.), pp. 83–108, Kluwer Academic Publishers, 1989.

[16] A. Griewank and P. L. Toint, "Partitioned variable metric updates for large structured optimization problems," *Numerische Mathematik*, vol. 39, pp. 119–137, 1982.

[17] N. Higham, *Accuracy and stability of numerical algorithms.* Philadelphia, PA: SIAM, 2002. Second edition.

[18] C. T. Kelley, *Iterative methods for linear and nonlinear equations.* Philadelphia, PA: SIAM, 1995.

[19] C.-J. Lin and J. Moré, "Incomplete Cholesky factorization with limited memory," *SIAM Journal on Scientific Computing*, vol. 21, pp. 24–45, 1999.

[20] D. Liu and J. Nocedal, "On the limited memory BFGS method for large scale optimization," *Mathematical Programming*, vol. 45, pp. 503–528, 1989.

[21] G. Meurant, *Computer solution of large linear systems.* Amsterdam: North Holland, Elsevier, 1999.

[22] J. L. Morales and J. Nocedal, "Algorithm PREQN: Fortran 77 subroutine for preconditioning the conjugate gradient method," *ACM Transaction on Mathematical Software*, vol. 27, pp. 83–91, 2001.

[23] J. Morales and J. Nocedal, "Automatic preconditioning by limited memory quasi–Newton updating," *SIAM Journal on Optimization*, vol. 10, pp. 1079–1096, 2000.

[24] S. Nash, "Preconditioning of truncated-Newton methods," *SIAM Journal on Scientific and Statistical Computing*, vol. 6, pp. 599–616, 1985.

[25] S. Nash, "A survey of truncated-Newton methods," *Journal of Computational and Applied Mathematics*, vol. 124, pp. 45–59, 2000.

[26] S. Nash and J. Nocedal, "A numerical study of the limited memory BFGS method and the truncated-Newton method for large scale optimization," *SIAM Journal on Optimization*, vol. 1, pp. 358–372, 1991.

[27] S. Nash and A. Sofer, "Assessing a search direction within a truncated-Newton method," *Operations Research Letter*, vol. 9, pp. 219–221, 1990.

[28] J. Nocedal, "Large scale unconstrained optimization," in *The state of the art in Numerical Analysis* (A. Watson and I. Duff, eds.), (Oxford), pp. 311–338, Oxford University Press, 1997.

[29] D. O'Leary, "A discrete Newton algorithm for minimizing a function of many variables," *Mathematical Programming*, vol. 23, pp. 20–33, 1982.

[30] B. Parlett and T. Landis, "Methods for scaling to double stochastic form," *Linear Algebra and Applications*, vol. 48, pp. 53–79, 1982.

[31] M. Roma, "Large scale unconstrained optimization," in *Encyclopedia of Optimization* (C. Floudas and P. Pardalos, eds.), vol. III, pp. 143–150, Kluwer Academic Publishers, 2001.

[32] D. Ruiz, "A scaling algorithm to equilibrate both rows and columns norms in matrices," Techical Report RAL–TR–2001–034, Computational Sciences and Engineering Department, Rutherford Appleton Laboratory, Oxon, UK, 2001. Submitted to *Linear Algebra and Applications*.

[33] T. Schlick, "Modified Cholesky factorization for sparse preconditioners," *SIAM Journal on Scientific Computing*, vol. 14, pp. 424–445, 1993.

[34] M. H. Schneider and S. Zenios, "A comparative study of algorithms for matrix balancing," *Operations Research*, vol. 38, pp. 439–455, 1990.

[35] A. Van der Sluis, "Condition number and equilibration of matrices," *Numerische Mathematik*, vol. 14, pp. 14–23, 1969.

[36] H. Van Der Vorst and K. Dekker, "Conjugate gradient type methods and preconditioning," *Journal of Computational and Applied Mathematics*, vol. 24, pp. 73–87, 1988.

[37] W. Wang and D. O'Leary, "Adaptative use of iterative methods in predictor–corrector interior point methods for linear programming," *Numerical Algorithms*, vol. 25, pp. 387–406, 2000.