RAL 95-010

On the Use of Element-by-Element Preconditioners to Solve Large Scale Partially Separable Optimization Problems[1]

Michel J. Daydé[2,3], Jean-Yves L'Excellent[2] and Nicholas I. M. Gould[3,4,5]

ABSTRACT
We study the solution of large-scale nonlinear optimization problems by methods which aim to exploit their inherent structure. In particular, we consider the all-pervasive property of partial separability, first studied by Griewank and Toint (1982*b*). A typical minimization method for nonlinear optimization problems approximately solves a sequence of simplified linearized subproblems. In this paper, we explore how partial separability may be exploited by iterative methods for solving these subproblems. We particularly address the issue of computing effective preconditioners for such iterative methods. Numerical experiments indicate the effectiveness of these preconditioners on large-scale examples.

Keywords: large-scale problems, unconstrained optimization, elememt-by-element preconditioners, conjugate-gradients.

AMS(MOS) subject classifications: 65F05, 65F10, 65F15, 65F50, 65K05, 90C30.

Also appeared as ENSEEIHT-IRIT report RT/APO/94/4.

[1] Travel was funded, in part, by the ALLIANCE program from the British Council.

[2] ENSEEIHT-IRIT, 2 rue Camichel, 31071 Toulouse Cedex, France.

[3] Parallel Algorithms Team, CERFACS, 42 Ave. G. Coriolis, 31057 Toulouse Cedex, France.

[4] Computing and Information Systems Department, Rutherford Appleton Laboratory, Chilton, Oxfordshire, OX11 0QX, England.

[5] Current reports available by anonymous ftp from joyous-gard.cc.rl.ac.uk (internet 130.246.9.91) in the directory "pub/reports".

Central Computing Department
Atlas Centre
Rutherford Appleton Laboratory
Oxon OX11 0QX

January 19, 1995.

Contents

1 Introduction

In this paper, we study algebraic aspects of the numerical solution of large-scale unconstrained optimization problems. To be specific, we suppose that we wish to minimize a partially separable objective function $f(\boldsymbol{x})$. The function f is said to be *partially separable* (see Griewank and Toint, 1982*a* and Griewank and Toint, 1982*c*) if

$$f(\boldsymbol{x}) = \sum_{i=1}^{p} f_i(\boldsymbol{x}), \tag{1.1}$$

where each *element* function f_i has a large invariant subspace. Typically, this occurs when $f_i(\boldsymbol{x})$ is only a function of a small subset of the variables $\boldsymbol{x}$, but may, of course, happen for other reasons (see, for example, Conn, Gould and Toint, 1990). The decomposition (1.1) is extremely general. Indeed, Griewank and Toint (1982*b*) show that any sufficiently differentiable function with a sparse Hessian matrix may be expressed in this form.

In this paper, we shall be concerned with those partially separable functions for which

$$f(\boldsymbol{x}) = \sum_{i=1}^{p} f_i(\boldsymbol{x}^i), \tag{1.2}$$

where

1. each set of *local* variables, $\boldsymbol{x}^i \in \Re^{n_i}$, is a subset of the *global* variables, $\boldsymbol{x} \in \Re^n$, and
2. $n_i \ll n$.

Thus, the Hessian matrix of each f_i is a low-rank, sparse matrix — typically it will differ from the zero matrix only in a full block in the rows and columns corresponding to the variables $\boldsymbol{x}^i$. The overall Hessian is thus the sum of extremely sparse matrices — the element Hessians — and is thus frequently itself also sparse.

In unconstrained optimization, one is normally concerned with obtaining an (approximate) solution, $\boldsymbol{d}$, to the Newton equations

$$\nabla_{xx} f(\boldsymbol{x}) \boldsymbol{d} = -\nabla_x f(\boldsymbol{x}) \tag{1.3}$$

If f has the form (1.2), these equations become

$$\left(\sum_{i=1}^{p} \nabla_{xx} f_i(\boldsymbol{x}^i) \right) \boldsymbol{d} = -\sum_{i=1}^{p} \nabla_x f_i(\boldsymbol{x}^i). \tag{1.4}$$

We are thus concerned with constructing efficient methods for solving systems of this form which exploit the algebraic structure as fully as possible.

Putting this is a more general context, we suppose that the real, symmetric, n by n matrix $\boldsymbol{H}$ may be expressed as

$$\boldsymbol{H} = \sum_{i=1}^{p} \boldsymbol{H}_i, \tag{1.5}$$

where the symmetric *elementary* matrix $\boldsymbol{H}_i$ only has non-zeros in n_i rows and columns. We consider solving the linear system

$$\boldsymbol{H}\boldsymbol{d} = \left(\sum_{i=1}^{p} \boldsymbol{H}_i \right) \boldsymbol{d} = -\boldsymbol{g} \tag{1.6}$$

where $\boldsymbol{H}$ is large and normally positive definite. Clearly, the system (1.4) is of this form. Similar linear systems arises when solving constrained optimization problems using augmented Lagrangian methods (see, for example, Hestenes, 1969, Powell, 1969 and Conn et al., 1990), and when using finite-element methods to solve elliptic partial differential equations (see, for instance, Zienkiewicz, 1977). Both direct and iterative methods may be appropriate for solving (1.6). Frontal or multifrontal direct (factorization) methods (see, for example, Irons, 1970, Duff and Reid, 1983 and Reid, 1984) are appropriate so long as there is room to store the fill-in which occurs during the matrix factorization. If this is not the case, one is forced to consider iterative methods. The symmetry and definiteness of $\boldsymbol{H}$ normally makes the preconditioned conjugate gradient method (see, for example, Golub and Van Loan, 1989) the method of choice. The difficulty is, of course, the choice of an effective preconditioner (see, Axelsson, 1976, for a discussion of general issues).

When designing an iterative solver for the solution of (1.6), certain features appear to be desirable or even crucial.

- Since the matrix (1.5) is initially unassembled, we do not especially want to assemble it.
- We would like to find a preconditioner that can be computed element-wise. It is also desirable not to have to assemble this preconditioner.
- The computations involved in forming and using the preconditioner should ideally be vectorizable and parallelizable, since we are interested in solving large problems. Note that in the conjugate gradients, the most consuming parts at each iteration are the matrix-vector product and the solve of the preconditioned system. As the matrix-vector product can be very well parallelized, it is thus crucial to parallelize the solve.
- As there is no absolute guarantee that the matrix (1.5) is positive definite, we would like to be able to detect when the matrix is indefinite. Thereafter, we might perturb the original matrix so that in all cases we compute the solution of safely positive definite systems.
- For optimization problems we must guarantee that the preconditioner is symmetric and positive definite.

The LANCELOT package for the solution of large-scale nonlinear optimization problems uses a variety of techniques to solve systems of the form (1.4). As well as direct methods, the package allows the use of conjugate gradients with various preconditioners. These include

- diagonal preconditioners,
- band preconditioners,
- incomplete Cholesky preconditioners,
- expanding band preconditioners, and
- full-matrix factorization preconditioners.

Further details are given in Conn, Gould and Toint, 1992.

We have decided to explore other alternatives, keeping in mind that the preconditioners should take advantage of the structure of the problems given by the partial separability and that we must be able to control their inertia. Thus, in this paper, we study the use of the following element-by-element preconditioners:

- The Element matrix Factorization (EMF) of Gustafsson and Lindskog (1986) based on a Cholesky factorization of each element;

- The Finite Element Preconditioner (FEP) of Kaasschieter (1989) and Wathen (1992);
- The one-pass (EBE) and two-pass (EBE2) element-by-element preconditioners of Hughes, Levit and Windget (1983) and Ortiz, Pinsky and Taylor (1983), initially described and used in the context of finite element techniques for partial differential equations; and
- The "Gauss Seidel" EBE preconditioner (GS EBE).

In Section 2, we describe the various element-by-element preconditioners used in the finite element solution of partial differential equations. In Section 3, we propose a generalization of the EBE preconditioner. In Section 4, we compare the numerical behaviour of these preconditioners with classical preconditioners on a set of numerical experiments. Finally, in Section 5, we make some remarks on the influence of the partitioning of the matrix (1.5) into elementary matrices.

We use the following notation: we let $\boldsymbol{I}$ denote the (appropriately dimensioned) identity matrix and we let $\Delta(\boldsymbol{A})$ be the diagonal matrix whose diagonals are the diagonals of $\boldsymbol{A}$.

2 Finite Element preconditioners

In this section, we review the range of element-by-element preconditioners which have been proposed for solving the linear systems that arise from finite-element solution of partial differential equations. We note that specific error analyses are possible for particular classes of model differential equations, but a more general analysis is most likely impossible. Thus, all of the preconditioners should be viewed as heuristics which aim to approximate (1.5) at low cost.

2.1 Connectivity matrices

As we are assuming that the element matrix $\boldsymbol{H}_i$ has non-zeros in just n_i rows, we may write

$$\boldsymbol{H}_i = \boldsymbol{C}_i^T \boldsymbol{H}_i^e \boldsymbol{C}_i, \tag{2.1}$$

where the rows of the n_i by n *connectivity* matrix, $\boldsymbol{C}_i$, are simply the rows of the n by n identity matrix corresponding to the variables used in the element, and $\boldsymbol{H}_i^e$ is a dense n_i by n_i symmetric matrix. In what follows, we may occasionally say that $\boldsymbol{H}_i$ is positive definite when strictly we mean that $\boldsymbol{H}_i^e$ is positive definite.

2.2 Element Matrix Factorization

First, we assume that the elementary matrices $\boldsymbol{H}_i^e$ are positive definite. The basic idea (see, Gustafsson and Lindskog, 1986) for forming the preconditioner is to factorize each elementary matrix into

$$\boldsymbol{H}_i^e = \boldsymbol{L}_i^e \boldsymbol{L}_i^{eT}, \tag{2.2}$$

where $\boldsymbol{L}_i^e$ is a lower triangular matrix. The preconditioner is then

$$\boldsymbol{P}_{EMF} = \left(\sum_{1=1}^{p} \boldsymbol{L}_i \right) \left(\sum_{1=1}^{p} \boldsymbol{L}_i \right)^T, \tag{2.3}$$

where $\boldsymbol{L}_i = \boldsymbol{C}_i^T \boldsymbol{L}_i^e$. Clearly $\sum_{1=1}^{p} \boldsymbol{L}_i$ is also lower triangular and thus (2.3) is easy to invert. More generally, letting $\boldsymbol{L}_i = \bar{\boldsymbol{L}}_i + \boldsymbol{D}_i$, where $\bar{\boldsymbol{L}}_i$ is the strictly lower triangular part of $\boldsymbol{L}_i$ and $\boldsymbol{D}_i = \Delta(\boldsymbol{L}_i)$, we might choose

$$\boldsymbol{P}_{EMF}(\theta) = \left((1+\theta)^{-1} \sum_{1=1}^{p} \bar{\boldsymbol{L}}_i + (1+\theta) \sum_{1=1}^{p} \boldsymbol{D}_i \right) \left((1+\theta)^{-1} \sum_{1=1}^{p} \bar{\boldsymbol{L}}_i + (1+\theta) \sum_{1=1}^{p} \boldsymbol{D}_i \right)^T, \tag{2.4}$$

where θ is a non-negative parameter. In our experiments for simplicity we choose $\theta = 0$, but other choices have been suggested by Gustafsson and Lindskog (1986) for finite-element applications.

2.3 Finite Element preconditioner

If $\boldsymbol{H}_i$ is positive semi-definite, Kaasschieter (1989) and Wathen (1992) have suggested modifying the previous preconditioner so that

$$\boldsymbol{H}_i = (\boldsymbol{D}_i + \bar{\boldsymbol{L}}_i)\boldsymbol{D}_i^+(\boldsymbol{D}_i + \bar{\boldsymbol{L}}_i^T), \tag{2.5}$$

where $\boldsymbol{D}_i^+$ is the pseudo-inverse of $\boldsymbol{D}_i$. Then, if $\sum_{i=1}^p \boldsymbol{D}_i$ has positive diagonal entries, the finite element preconditioner (FEP) is

$$\boldsymbol{P}_{FEP} = \left(\sum_{i=1}^p \boldsymbol{D}_i + \sum_{i=1}^p \bar{\boldsymbol{L}}_i\right)\left(\sum_{i=1}^p \boldsymbol{D}_i\right)^{-1}\left(\sum_{i=1}^p \boldsymbol{D}_i + \sum_{i=1}^p \bar{\boldsymbol{L}}_i^T\right). \tag{2.6}$$

2.4 The EBE preconditioner

Element-By-Element (EBE) preconditioners were introduced by Hughes et al. (1983) and Ortiz et al. (1983) and have been successfully applied in a number of applications in engineering and physics (see, for example, Hughes, Ferencz and Hallquits, 1987, and Erhel, Traynard and Vidrascu, 1991). A detailed analysis of this technique is given by Wathen (1989).

We now assume that $\boldsymbol{H}$ is positive definite. We may rewrite $\boldsymbol{H}$ as

$$\boldsymbol{H} = \sum_{i=1}^p \boldsymbol{M}_i + \sum_{i=1}^p (\boldsymbol{H}_i - \boldsymbol{M}_i) = \boldsymbol{M} + \sum_{i=1}^p (\boldsymbol{H}_i - \boldsymbol{M}_i), \tag{2.7}$$

where $\boldsymbol{M}_i = \Delta(\boldsymbol{H}_i)$ and $\boldsymbol{M} = \sum_{i=1}^p \boldsymbol{M}_i$. Now, let $\boldsymbol{M} = \boldsymbol{L}_M \boldsymbol{L}_M^T$ be the Cholesky factorization of $\boldsymbol{M}$; of course $\boldsymbol{L}_M$ is simply a diagonal matrix. Then,

$$\boldsymbol{H} = \boldsymbol{L}_M\left(\boldsymbol{I} + \sum_{i=1}^p \boldsymbol{L}_M^{-1}(\boldsymbol{H}_i - \boldsymbol{M}_i)\boldsymbol{L}_M^{-T}\right)\boldsymbol{L}_M^T = \boldsymbol{L}_M\left(\boldsymbol{I} + \sum_{i=1}^p \boldsymbol{E}_i\right)\boldsymbol{L}_M^T, \tag{2.8}$$

where we have defined $\boldsymbol{E}_i = \boldsymbol{L}_M^{-1}(\boldsymbol{H}_i - \boldsymbol{M}_i)\boldsymbol{L}_M^{-T}$. Consider the product $\prod_{i=1}^p(\boldsymbol{I} + \boldsymbol{E}_i)$. A simple calculation reveals that

$$\prod_{i=1}^p(\boldsymbol{I} + \boldsymbol{E}_i) \approx \boldsymbol{I} + \sum_{i=1}^p \boldsymbol{E}_i \tag{2.9}$$

where the error in the approximation may be expressed in terms of second and higher order products of the components $\boldsymbol{E}_i$ and $\boldsymbol{E}_j$ with $i \neq j$. Thus $\boldsymbol{I} + \sum_{i=1}^p \boldsymbol{E}_i$ will be well approximated by $\prod_{i=1}^p(\boldsymbol{I} + \boldsymbol{E}_i)$ if the the norms of the terms involving products of the $\boldsymbol{E}_i$ are small compared to 1. This may be true for various reasons:

- Individual $\boldsymbol{E}_i$ may be small or zero (likely to be true if $\boldsymbol{H}_i$ is very diagonal dominant);
- The product of the overlapping components $\boldsymbol{E}_i$ and $\boldsymbol{E}_j$ is small or zero.

Note that the order used for writing the product is not without importance. The preconditioner has to be symmetric as we intend to use conjugate gradient. The $\boldsymbol{E}_i$ are symmetric matrices but $\boldsymbol{E}_i\boldsymbol{E}_j$ is, in general, not. There are, of course, ways of symmetrizing the approximation (2.9), such as writing

$$\boldsymbol{I} + \sum_{i=1}^p \boldsymbol{E}_i \approx \left(\prod_{i=1}^p(\boldsymbol{I} + \tfrac{1}{2}\boldsymbol{E}_i)\right)\left(\prod_{i=p}^1(\boldsymbol{I} + \tfrac{1}{2}\boldsymbol{E}_i)\right) \tag{2.10}$$

but a discussion of probably the best such scheme is deferred until Section 2.5.

The first fundamental feature of the EBE preconditioner is that we replace $\boldsymbol{I} + \sum_{i=1}^{p} \boldsymbol{E}_i$ by $\prod_{i=1}^{p}(\boldsymbol{I} + \boldsymbol{E}_i)$ in (2.8) We then assume that $\boldsymbol{I} + \boldsymbol{E}_i$ is positive definite and has an LDL^T factorization

$$\boldsymbol{W}_i \stackrel{\text{def}}{=} \boldsymbol{I} + \boldsymbol{E}_i = \boldsymbol{L}_i \boldsymbol{D}_i \boldsymbol{L}_i^T; \tag{2.11}$$

the matrix $\boldsymbol{W}_i$ is known as the Winget decomposition of $\boldsymbol{H}_i$ (see Hughes et al., 1983). In this case, (2.8), (2.9) and (2.11) imply that

$$\boldsymbol{H} \approx \boldsymbol{L}_M \left(\prod_{i=1}^{p} \boldsymbol{L}_i \boldsymbol{D}_i \boldsymbol{L}_i^T \right) \boldsymbol{L}_M^T. \tag{2.12}$$

Unfortunately, (2.12) is not symmetric, and thus is not a satisfactory preconditioner. However, if we further assume that a rearrangement of the product

$$\prod_{i=1}^{p} \boldsymbol{L}_i \boldsymbol{D}_i \boldsymbol{L}_i^T \approx \left(\prod_{i=1}^{p} \boldsymbol{L}_i \right) \left(\prod_{i=1}^{p} \boldsymbol{D}_i \right) \left(\prod_{i=p}^{1} \boldsymbol{L}_i^T \right) \tag{2.13}$$

introduces little additional error, we obtain the matrix

$$\boldsymbol{P}_{EBE} = \boldsymbol{L}_M \left(\prod_{i=1}^{p} \boldsymbol{L}_i \right) \left(\prod_{i=1}^{p} \boldsymbol{D}_i \right) \left(\prod_{i=p}^{1} \boldsymbol{L}_i^T \right) \boldsymbol{L}_M^T \tag{2.14}$$

which may be used as a preconditioner for $\boldsymbol{H}$. Such a matrix is known as the *EBE* preconditioner. Note that the second approximation (2.13) is, as the previous one, exact if there is no overlap between the blocks and will be good under exactly the same circumstances as its predecessor.

Clearly, the efficiency of the EBE preconditioner depends on the the partitioning of the initial matrix and on the size of the off-diagonal elements of the elementary matrices. In order to solve efficiently the system of equations $\boldsymbol{P}_{EBE}\boldsymbol{x} = \boldsymbol{y}$, we exploit the decomposition (2.14).

We are free to order the elements in any way we choose and may thus encourage parallelism by consecutively ordering non-overlapping elements so that we can perform groups of forward and backsolve in parallel.

2.5 Two-pass EBE preconditioners

Here we consider another proposal in the same vein (see Hughes, 1987). We proceed, as in Section 2.4, to approximate the sum $\boldsymbol{I} + \sum_{i=1}^{p} \boldsymbol{E}_i$ by a product of invertible matrices. We dismissed using the approximation (2.12) as a preconditioner because of its non-symmetry. Instead, we combine (2.8) and the relationship

$$\boldsymbol{I} + \sum_{i=1}^{p} \boldsymbol{E}_i \approx \prod_{i=1}^{p} (\boldsymbol{I} + \tfrac{1}{2}\boldsymbol{E}_i) \prod_{i=p}^{1} (\boldsymbol{I} + \tfrac{1}{2}\boldsymbol{E}_i) \tag{2.15}$$

to give the preconditioner

$$\boldsymbol{P}_{EBE2} = \boldsymbol{L}_M \left(\prod_{i=1}^{p} (\boldsymbol{I} + \tfrac{1}{2}\boldsymbol{E}_i) \right) \left(\prod_{i=p}^{1} (\boldsymbol{I} + \tfrac{1}{2}\boldsymbol{E}_i) \right) \boldsymbol{L}_M^T. \tag{2.16}$$

Note that (2.16) is positive definite if and only if all the matrices $\boldsymbol{I} + \frac{1}{2}\boldsymbol{E}_i$ are nonsingular. To use this preconditioner, we merely require that each $\boldsymbol{I} + \frac{1}{2}\boldsymbol{E}_i$ is invertible, and to be able to solve systems of equations of the form $\boldsymbol{P}_{EBE2}\boldsymbol{x} = \boldsymbol{y}$ efficiently.

As before, we are free to order the elements in any way we choose and may thus encourage parallelism by consecutively ordering non-overlapping elements. We may also choose to obtain explicit inverses of the $\boldsymbol{I} + \frac{1}{2}\boldsymbol{E}_i$ to exploit vectorization in the forward and back substitutions.

The main problem with the approximation (2.15) is that the terms $\frac{1}{4}\boldsymbol{E}_i{}^2$, which result when expanding the product (2.16), are non-zero even if there is little overlap between distinct element Hessians $\boldsymbol{E}_i$ and $\boldsymbol{E}_j$ $(i \neq j)$. Furthermore, as a solve using EBE2 is roughly twice as expensive as one with EBE, in practice EBE2 is less efficient than EBE. But it can be interesting if we use inverse or approximate inverse elements or if we can find a way to subtract the terms $\frac{1}{4}\boldsymbol{E}_i{}^2$ in the solve.

2.6 The GS EBE preconditioner

The GS (Gauss-Seidel) EBE preconditioner is based on the same decomposition as the EBE preconditioner. But instead of using a Crout factorization, we instead use the decomposition

$$\boldsymbol{E}_i = \boldsymbol{L}_i + \boldsymbol{L}_i^T \tag{2.17}$$

where $\boldsymbol{L}_i$ is a strictly lower triangular matrix. The preconditioner is then

$$\boldsymbol{P}_{GS} = \boldsymbol{L}_M \prod_{i=1}^{p}(\boldsymbol{I} + \boldsymbol{L}_i) \prod_{i=p}^{1}(\boldsymbol{I} + \boldsymbol{L}_i^T)\boldsymbol{L}_M^T \tag{2.18}$$

The advantage of this preconditioner is obviously that it is very easy to construct. In fact, if the matrix is initially scaled, the preconditioner is not constructed and no more storage than diagonal is required. The principal drawback is that it is not exact, even when there is no overlap between element Hessians, because of the terms $\boldsymbol{L}_i\boldsymbol{L}_i^T$ which arise when approximating $\boldsymbol{I} + \sum_{i=1}^{p} \boldsymbol{E}_i$ by $\prod_{i=1}^{p}(\boldsymbol{I} + \boldsymbol{L}_i) \prod_{i=p}^{1}(\boldsymbol{I} + \boldsymbol{L}_i^T)$.

2.7 Definiteness of the preconditioner

The preconditioning matrix $\boldsymbol{P}$ should be positive definite. In the context of optimization problems, we have no guarantee that $\boldsymbol{H}$ and *a fortiori* $\boldsymbol{P}$ is positive definite. We describe here a simple strategy that guarantees that $\boldsymbol{P}$ is positive definite. Note that, if the elementary matrices $\boldsymbol{H}_i$ are positive definite, $\boldsymbol{H}$ will be positive definite. We also note that, when considering the EBE and EBE2 preconditioners, a sufficient condition for $\boldsymbol{P}$ to be positive definite is that all the $\boldsymbol{H}_i$, and hence the $\boldsymbol{I} + \boldsymbol{E}_i$, are — if $\boldsymbol{H}_i$ positive definite for all i, so *a fortiori* are $\boldsymbol{I} + \boldsymbol{E}_i$ and $\boldsymbol{I} + \frac{1}{2}\boldsymbol{E}_i$. However, this is not necessary since the $\boldsymbol{I} + \boldsymbol{E}_i$ and $\boldsymbol{P}$ may be positive definite even if some $\boldsymbol{H}_i$ are not. Griewank and Toint (1984) study conditions under which partially separable functions have convex decompositions, that is those functions whose Hessian matrix is the sum of positive semi-definite element Hessians.

The preconditioners we have considered in this section all depend upon the decomposition of a collection of matrices $\boldsymbol{W}_i$, where $\boldsymbol{W}_i = \boldsymbol{H}_i$, $\boldsymbol{I} + \boldsymbol{E}_i$ or $\boldsymbol{I} + \frac{1}{2}\boldsymbol{E}_i$ depending on the preconditioner. It thus seems natural to consider the use of a modified Cholesky factorization for these decompositions to guarantee that the preconditioner is positive definite. We use the modified Cholesky factorization proposed by Schnabel and Eskow (1991) that computes the Cholesky factorization of a matrix $\boldsymbol{W}_i$ if it is positive definite, or the Cholesky factorization of $\boldsymbol{W}_i + \boldsymbol{B}_i$, where $\boldsymbol{B}_i$ is a non negative diagonal matrix, otherwise. There is no need to know *a priori* if $\boldsymbol{W}_i$ is positive definite, and the matrix $\boldsymbol{B}_i$ is determined during the factorization process. This modified Cholesky factorization exhibits marginally better properties in terms of computational costs and upper bound on $||\boldsymbol{W}_i||_\infty$ than those described by Gill and Murray (1974) and Gill, Murray and Wright (1981).

The main drawback of such a strategy for forming the preconditioner, is that we may perturb $\boldsymbol{W}_i$ even if the initial matrix is positive definite. Before attempting to form the preconditioner, we should first amalgamate elementary matrices whose sparsity structure lies completely within that of another element. We might also consider amalgamating indefinite elements with positive definite ones if the composite element is positive definite, even if this means introducing structural zeros that is zeros within the element — the amalgamation of an indefinite element might, for example, be made with the elementary matrix having the largest intersection. While we have not developed a fail-safe scheme here, the simple strategy just presented appears to work well in practice.

3 Generalizing the EBE preconditioner

Let $\boldsymbol{H}$ satisfy (1.5). In Sections 2.4 and 2.5, we considered the classical and two-pass EBE preconditioners

$$\boldsymbol{P}_{EBE} = \boldsymbol{L}_M \left(\prod_{i=1}^{p} \boldsymbol{L}_i \right) \left(\prod_{i=1}^{p} \boldsymbol{D}_i \right) \left(\prod_{i=p}^{1} \boldsymbol{L}_i^T \right) \boldsymbol{L}_M^T \tag{3.1}$$

and

$$\boldsymbol{P}_{EBE2} = \boldsymbol{L}_M \left(\prod_{i=1}^{p} (\boldsymbol{I} + \tfrac{1}{2}\boldsymbol{E}_i) \right) \left(\prod_{i=p}^{1} (\boldsymbol{I} + \tfrac{1}{2}\boldsymbol{E}_i) \right) \boldsymbol{L}_M^T, \tag{3.2}$$

where $\boldsymbol{M} \stackrel{\text{def}}{=} \Delta(\boldsymbol{H}) = \boldsymbol{L}_M \boldsymbol{L}_M^T$, $\boldsymbol{M}_i = \Delta(\boldsymbol{H}_i)$, $\boldsymbol{E}_i \stackrel{\text{def}}{=} \boldsymbol{L}_M^{-1}(\boldsymbol{H}_i - \boldsymbol{M}_i)\boldsymbol{L}_M^{-T}$ and the Winget decomposition, $\boldsymbol{W}_i$, had a factorization

$$\boldsymbol{W}_i \stackrel{\text{def}}{=} \boldsymbol{I} + \boldsymbol{E}_i = \boldsymbol{L}_i \boldsymbol{D}_i \boldsymbol{L}_i^T. \tag{3.3}$$

In this section, we consider a generalization of $\boldsymbol{M}$ with the intention of including more than simply the diagonal of $\boldsymbol{H}$.

Let $\boldsymbol{M}_i^e$ be a symmetric n_i by n_i matrix, for each $1 \le i \le p$, such that the composite matrix,

$$\boldsymbol{M} = \sum_{i=1}^{p} \boldsymbol{M}_i, \tag{3.4}$$

is positive definite, and where

$$\boldsymbol{M}_i = \boldsymbol{C}_i^T \boldsymbol{M}_i^e \boldsymbol{C}_i. \tag{3.5}$$

Then we observe that the descriptions of the EBE and two-pass EBE preconditioners given in Sections 2.4 and 2.5 *do not depend* on $\boldsymbol{M}$ being the diagonal of $\boldsymbol{H}$. We may thus think of using (3.1) or (3.2) with a non-diagonal $\boldsymbol{M}$. However, a restriction on the choice of $\boldsymbol{M}$ has to be made, since we want to have the same structure for the matrices $\boldsymbol{H}_i$ and $\boldsymbol{E}_i$. A general matrix $\boldsymbol{M}$ — indeed even a band matrix — may introduce fill-in, complicate the required data structures and increase storage and computational overheads. For simplicity in this section, we only consider matrices $\boldsymbol{M}$ for which no fill-in occurs.

3.1 Overlapping blocks of the matrix

The ideal preconditioner, if it could be economically formed and factorized, would be to pick $\boldsymbol{M}_i = \boldsymbol{H}_i$, and hence $\boldsymbol{P} = \boldsymbol{M} = \boldsymbol{H}$. Unfortunately, this is frequently out of the question. We must, therefore, be content with an approximate preconditioner. We know, from Section 2.4 that the primary source of errors in the EBE preconditioners may be attributed to the overlap between the $\boldsymbol{E}_i$. It is thus of interest to mitigate this effect by aiming to include as many of the overlapping portions of the elements in $\boldsymbol{M}$ as is possible.

Example. To illustrate our ideas, we consider what happens in a simple structured problem with two elements.

Suppose $\boldsymbol{H} = \boldsymbol{H}_1 + \boldsymbol{H}_2$ In this case, the error, $\boldsymbol{E}$, due to the replacement of the sum by a product is

$$\boldsymbol{E} = \boldsymbol{L}_M(\boldsymbol{I} + \boldsymbol{E}_1)(\boldsymbol{I} + \boldsymbol{E}_2)\boldsymbol{L}_M^T - \boldsymbol{H} = (\boldsymbol{H}_1 - \boldsymbol{M}_1)\boldsymbol{M}^{-1}(\boldsymbol{H}_2 - \boldsymbol{M}_2). \tag{3.6}$$

It is useful to make this error as small as possible, since the additional error due to the rearrangement (2.13) depends on $\boldsymbol{E}$.

Now suppose $\boldsymbol{H}_1$ and $\boldsymbol{H}_2$ are

$$\boldsymbol{H}_1 = \begin{pmatrix} \boldsymbol{A}_1 & \boldsymbol{B}_1^T & 0 \\ \boldsymbol{B}_1 & \boldsymbol{C}_1 & 0 \\ 0 & 0 & 0 \end{pmatrix} \quad \text{and} \quad \boldsymbol{H}_2 = \begin{pmatrix} 0 & 0 & 0 \\ 0 & \boldsymbol{C}_2 & \boldsymbol{B}_2^T \\ 0 & \boldsymbol{B}_2 & \boldsymbol{A}_2 \end{pmatrix}. \tag{3.7}$$

Further, suppose we choose $\boldsymbol{M} = \boldsymbol{M}_1 + \boldsymbol{M}_2$, where

$$\boldsymbol{M}_1 = \begin{pmatrix} \boldsymbol{N}_1 & 0 & 0 \\ 0 & \boldsymbol{C}_1 & 0 \\ 0 & 0 & 0 \end{pmatrix} \quad \text{and} \quad \boldsymbol{M}_2 = \begin{pmatrix} 0 & 0 & 0 \\ 0 & \boldsymbol{C}_2 & 0 \\ 0 & 0 & \boldsymbol{N}_2 \end{pmatrix} \tag{3.8}$$

and $\boldsymbol{N}_1$ and $\boldsymbol{N}_2$ are any matrices of appropriate dimensions. Then, combining (3.7) and (3.8), we obtain

$$\boldsymbol{E} = \begin{pmatrix} 0 & 0 & \boldsymbol{B}_1^T(\boldsymbol{C}_1 + \boldsymbol{C}_2)^{-1}\boldsymbol{B}_2^T \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}. \tag{3.9}$$

If, on the other hand, $\boldsymbol{M} = \Delta(\boldsymbol{H})$, the error is

$$\boldsymbol{E} = \begin{pmatrix} 0 & \boldsymbol{B}_1^T\Delta(\boldsymbol{C}_1 + \boldsymbol{C}_2)^{-1}(\boldsymbol{C}_2 - \Delta(\boldsymbol{C}_2)) & \boldsymbol{B}_1^T\Delta(\boldsymbol{C}_1 + \boldsymbol{C}_2)^{-1}\boldsymbol{B}_2^T \\ 0 & (\boldsymbol{C}_1 - \Delta(\boldsymbol{C}_1))\Delta(\boldsymbol{C}_1 + \boldsymbol{C}_2)^{-1}(\boldsymbol{C}_2 - \Delta(\boldsymbol{C}_2)) & (\boldsymbol{C}_1 - \Delta(\boldsymbol{C}_1))\Delta(\boldsymbol{C}_1 + \boldsymbol{C}_2)^{-1}\boldsymbol{B}_2^T \\ 0 & 0 & 0 \end{pmatrix}. \tag{3.10}$$

Remarkably, the errors do not depend upon $\boldsymbol{N}_1$ or $\boldsymbol{N}_2$. While this example is rather simple, it indicates the potential for allowing non-diagonal contributions within $\boldsymbol{M}$. In general, it may prove advantageous for $\boldsymbol{M}$ to be block diagonal, so long as the factorization of $\boldsymbol{M}$ remains cheap.

3.2 Construction

Since we wish to avoid an increase in storage/calculation, we are quite limited in the choice of the diagonal blocks $\boldsymbol{M}_i$. For example, suppose $\boldsymbol{H} = \boldsymbol{H}_1 + \boldsymbol{H}_2 + \boldsymbol{H}_3$ where $\boldsymbol{H}_1$ involves variables x_1, x_2, x_3, x_4, $\boldsymbol{H}_2$ involves variables x_2, x_3, x_4, x_5, x_6 and $\boldsymbol{H}_3$ involves variables x_3, x_4, x_5, x_7 (see figure 3.1). Then the only block we can include in $\boldsymbol{M}$ is the one corresponding to the variables x_3 and x_4. For, taking the block involving x_2, x_3, x_4 would introduce fill-in when calculating $\boldsymbol{E}_3$ while taking that for x_3, x_4, x_5 would introduce fill-in when calculating $\boldsymbol{E}_1$. We note, however, that in this example, it would be useful to amalgamate $\boldsymbol{H}_1$ and $\boldsymbol{H}_2$ or $\boldsymbol{H}_2$ and $\boldsymbol{H}_3$.

With this in mind, we propose the following algorithm to construct a generalized EBE preconditioner:

1. Compute the list of all elements involving a given variable;

2. Group together all variables involved with the same set of elements;

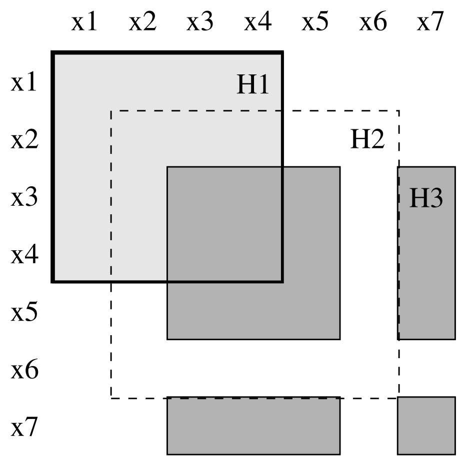

Figure 3.1: Example where the only block that will not introduce fill-in is x_3, x_4.

3. Assemble the blocks $\boldsymbol{M}_i$, each $\boldsymbol{M}_i$ corresponding to a group of variables computed in step 2;
4. Compute $\boldsymbol{M}_i = \boldsymbol{L}_{M_i}\boldsymbol{L}_{M_i}^T$;
5. Construct the $\boldsymbol{E}_i$ and calculate the modified Cholesky factorization of $\boldsymbol{I} + \boldsymbol{E}_i$.

3.3 Remarks

We note that the construction of Section 3.2 is rather restrictive. For example, if we consider a tridiagonal matrix found by summing elements with variables x_i and x_{i+1}, $1 \leq i \leq p$, our construction would prevent $\boldsymbol{M}$ from including more than 1 by 1 diagonal blocks.

We might hope that, by taking into account the zeros created when subtracting $\boldsymbol{M}_i$ from $\boldsymbol{H}_i$, we might be able to include more of $\boldsymbol{H}$ in $\boldsymbol{M}$ without introducing fill-in in $\boldsymbol{E}_i$. Whenever the initial matrix $\boldsymbol{M}$ introduced fill-in to the Winget decomposition, we have tried removing, one at a time, those elements of $\boldsymbol{M}$ which caused this fill-in. Unfortunately the tests we performed with this in mind indicated that such a strategy is rarely better than the simpler construction of Section 3.2.

Thus, so long as we prohibit fill-in within the Winget decomposition, we are quite restricted as to choice of $\boldsymbol{M}$. It would be of interest to allow limited fill-in within $\boldsymbol{E}_i$ and we intend to investigate this in the future.

3.4 Other possible generalizations

3.4.1 Higher order approximations to the summation

We indicated that a difficulty with EBE2 was due to the terms $\frac{1}{4}\boldsymbol{E}_i^2$ when using the approximation (2.15). To remove these terms, one may use the approximation

$$\boldsymbol{I} + \sum_{i=1}^{p} \boldsymbol{E}_i \approx \prod_{i=1}^{p}(\boldsymbol{I} + \tfrac{1}{2}\boldsymbol{E}_i - \tfrac{1}{8}\boldsymbol{E}_i^2) \prod_{i=p}^{1}(\boldsymbol{I} + \tfrac{1}{2}\boldsymbol{E}_i - \tfrac{1}{8}\boldsymbol{E}_i^2) \tag{3.11}$$

which is based on the second-order approximation $\boldsymbol{I} + \boldsymbol{E}_i = (\boldsymbol{I} + \frac{1}{2}\boldsymbol{E}_i - \frac{1}{8}\boldsymbol{E}_i^2)^2 + O(\boldsymbol{E}_i^3)$. Such an approximation is only really of use when the spectrum of $\boldsymbol{E}_i$ is within $[-1, 1]$, as otherwise the $O(\boldsymbol{E}_i^3)$ term may dominate. Higher-order approximations are also possible.

Unfortunately, such approximations do not appear to offer significant improvements over EBE2 in the tests we have performed. In view of their additional complexity, they will not be considered further in this paper.

3.4.2 Use of the square root of $I + E_i$

Another way of symmetrizing the preconditioner is based on using the square root of $\boldsymbol{I} + \boldsymbol{E}_i$ when it exists. The square root of a positive definite matrix $\boldsymbol{A}$ is defined to be the unique positive definite matrix $\boldsymbol{X}$ such that $\boldsymbol{X}^2 = \boldsymbol{A}$. Combining (2.8) and

$$\boldsymbol{I} + \sum_{i=1}^{p} \boldsymbol{E}_i \approx \left(\prod_{i=1}^{p} \sqrt{\boldsymbol{I} + \boldsymbol{E}_i} \right) \left(\prod_{i=p}^{1} \sqrt{\boldsymbol{I} + \boldsymbol{E}_i} \right), \tag{3.12}$$

we obtain the preconditioner

$$\boldsymbol{L}_M \left(\prod_{i=1}^{p} \sqrt{\boldsymbol{I} + \boldsymbol{E}_i} \right) \left(\prod_{i=p}^{1} \sqrt{\boldsymbol{I} + \boldsymbol{E}_i} \right) \boldsymbol{L}_M^T. \tag{3.13}$$

This is an exact preconditioner when there is no overlap and it does not need any additional approximation of the type (2.13). However, the cost of computing the matrix square root can sometimes be prohibitive, even for small elements. In tests, such a scheme appears to give a preconditioner whose effectiveness lies somewhere between EBE and EBE2, but at quite an additional cost. Again, we shall not consider this method further in this paper.

4 Experiments with Element-by-Element preconditioners

A number of the preconditioners described in Section 2 are known to work well in practice when applied to classes of problems arising from partial differential equations. In this section, we aim to investigate whether these preconditioners are effective in the more general context of systems which arise from partially separable optimization applications. Although our experiments are still preliminary, they do lead to interesting conclusions and are helpful in deciding future directions of research.

4.1 Tests on structured matrices with specified values

We first test our preconditioners on randomly generated matrices $\boldsymbol{H}$ whose structure reflects that which frequently arises in optimization applications. For such problems, the overlap appears only between two successive elements; the structure of the test matrices is the one defined in Figure 4.2.

Each matrix is the sum of randomly generated blocks; we merely specify the range of the spectrum within each block. We construct problems with p blocks, each of size k and each of which overlaps with the last r rows and columns of the previous block — the order of the whole matrix is thus $n = (k - r) * p + r$.

The results presented here arise from two types of matrices :

- Type I : the eigenvalues of each block are randomly generated in the range [0; 10000.0];
- Type II : there are three ranges of eigenvalues depending on the blocks : [0.01; 5.0], [10.0; 100.0] and [500.0; 10000.0].

In addition to the preconditioners discussed in Sections 2 and 3, we also consider the following:

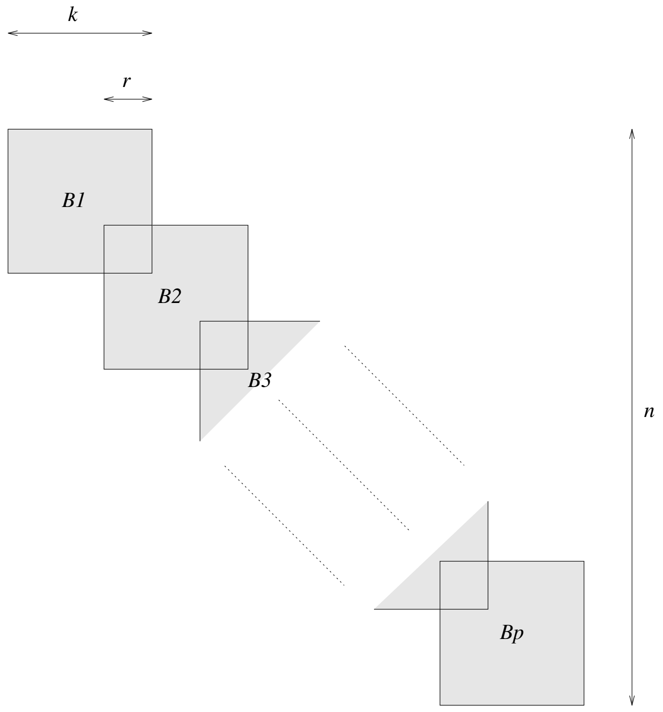

Figure 4.2: structure of the test matrices

Richardson : For this method there is no preconditioner. ($\boldsymbol{P} = \boldsymbol{I}$)

Diagonal : The diagonal of $\boldsymbol{H}$ is used : $\boldsymbol{P} = \Delta(\boldsymbol{H})$.

Block Jacobi : The preconditioner is formed from the diagonal blocks of the matrix. Here we have chosen to assemble the blocks where there is some overlap; elsewhere, the diagonal is taken. This preconditioner may be more efficient than diagonal when there is significant overlap. Of course, we could have chosen other block structures but this is a reasonable choice.

We ran the preconditioned conjugate gradient method (see, for example, Golub and Van Loan, 1989) to solve the system $\boldsymbol{Hx} = \boldsymbol{b}$, starting with the estimate $\boldsymbol{x} = 0$, and stopping as soon as $||\boldsymbol{Hx} - \boldsymbol{b}||_2 < 10^{-9}||\boldsymbol{b}||_2$. The right hand side $\boldsymbol{b}$ was either $(1, 1, ..., 1)^T$, or problem dependent. All of the experiments reported in this paper were performed in double precision on a single processor of an Alliant FX/80, with vectorization of the inner loops of the solves and the matrix-vector products.

Tables 4.1, 4.2 and 4.3 give some results of the preconditioned conjugate gradient with different preconditioners. On these tables, we observe that EBE is more efficient than diagonal so long as there is low overlap between elements.

In Figure 4.3, we illustrate the convergence behaviour for each of our preconditioners for a matrix of type II with 150 elements of size 12 with an overlap of 2 (third line of Table 4.3).

A good preconditioner for the conjugate gradient algorithm is supposed to cluster the eigenvalues. On matrices of type II, the element-by-element preconditioners are very efficient because of the initial intervals of eigenvalues. Frequency histograms of the eigenvalues are given Figure 4.4.

[1]For EMF and FEP, the times of execution are not reported because the triangular matrices were assembled as dense matrices and thus the implementation is suboptimal

[2]Block Jacobi was not tested for overlaps 8 and 9 because of double overlapping : the strategy used in the choice of the diagonal blocks cannot be applied

Over-lap	Order of H	Preconditioner							
		None	Diagonal	Block Jacobi	EBE	EBE2	GS EBE	EMF[1]	FEP[1]
0	600	180/4.4	160/4.0	160/4.0	1/0.24	48/5.2	64/4.9	1	1
1	581	159/3.9	127/3.2	127/3.8	12/1.2	37/4.0	48/3.7	20	12
2	562	144/3.6	110/2.8	110/3.4	12/1.1	33/3.6	47/3.6	21	15
3	543	133/3.2	98/2.5	97/3.1	13/1.2	30/3.3	39/3.0	23	16
4	524	118/2.9	84/2.1	83/2.7	14/1.3	26/2.8	33/2.5	23	16
5	505	112/2.7	75/1.9	75/2.5	13/1.2	24/2.6	30/2.3	24	17
6	486	107/2.6	69/1.7	68/2.3	13/1.2	21/2.3	27/2.0	26	18
7	467	85/2.1	58/1.5	57/2.0	11/0.99	19/2.1	24/1.8	27	20
8	448	79/1.9	54/1.4	52/1.9	11/0.98	17/1.9	22/1.6	26	18
9	429	73/1.8	50/1.3	48/1.8	11/0.96	16/1.7	20/1.5	26	18
10	410	70/1.7	47/1.2	45/1.7	11/0.95	16/1.7	19/1.4	26	19
11	391	66/1.6	43/1.1	41/1.6	11/0.94	15/1.6	18/1.3	25	20
12	372	61/1.5	38/0.99	37/1.5	10/0.85	13/1.4	16/1.2	25	18
13	353	60/1.5	36/0.94	33/1.4	10/0.84	13/1.4	16/1.2	24	18
14	334	52/1.3	34/0.89	30/1.2	10/0.83	12/1.3	14/1.0	24	19
15	315	46/1.2	31/0.83	27/1.1	11/0.89	12/1.2	14/1.0	22	18

Table 4.1: Number of iterations and time for convergence of the Preconditioned Conjugate Gradient Algorithm on a matrix of type I with 20 blocks of size 30.

Over-lap	Order of H	Preconditioner							
		None	Diagonal	Block Jacobi	EBE	EBE2	GS EBE	EMF[1]	FEP[1]
0	3000	322/26.0	323/27.5	323/27.5	1/0.81	97/43.6	125/39.7	1	1
1	2801	213/17.2	158/13.4	158/21.0	17/6.4	50/22.3	67/20.9	29	22
2	2602	145/11.6	98/8.3	97/14.0	18/6.6	32/14.2	39/12.0	31	23
3	2403	111/8.8	80/6.7	80/12.2	16/5.7	26/11.4	33/9.9	31	24
4	2204	106/8.4	66/5.5	66/10.5	14/4.8	23/9.9	28/8.2	30	24
5	2005	82/6.4	52/4.3	51/8.7	13/4.4	19/8.1	21/6.1	27	21
6	1806	73/5.7	43/3.5	40/7.2	12/3.9	16/6.7	18/5.1	28	22
7	1607	56/4.3	37/3.0	32/5.9	12/3.7	14/5.8	16/4.4	26	21
8	1408	39/3.0	32/2.6	[2]	11/3.3	13/5.3	14/3.8	26	19
9	1209	37/2.9	28/2.3	[2]	11/3.2	12/4.8	12/3.2	22	18

Table 4.2: Number of iterations and time for convergence of the Preconditioned Conjugate Gradient Algorithm on a matrix of type I with 200 blocks of size 15

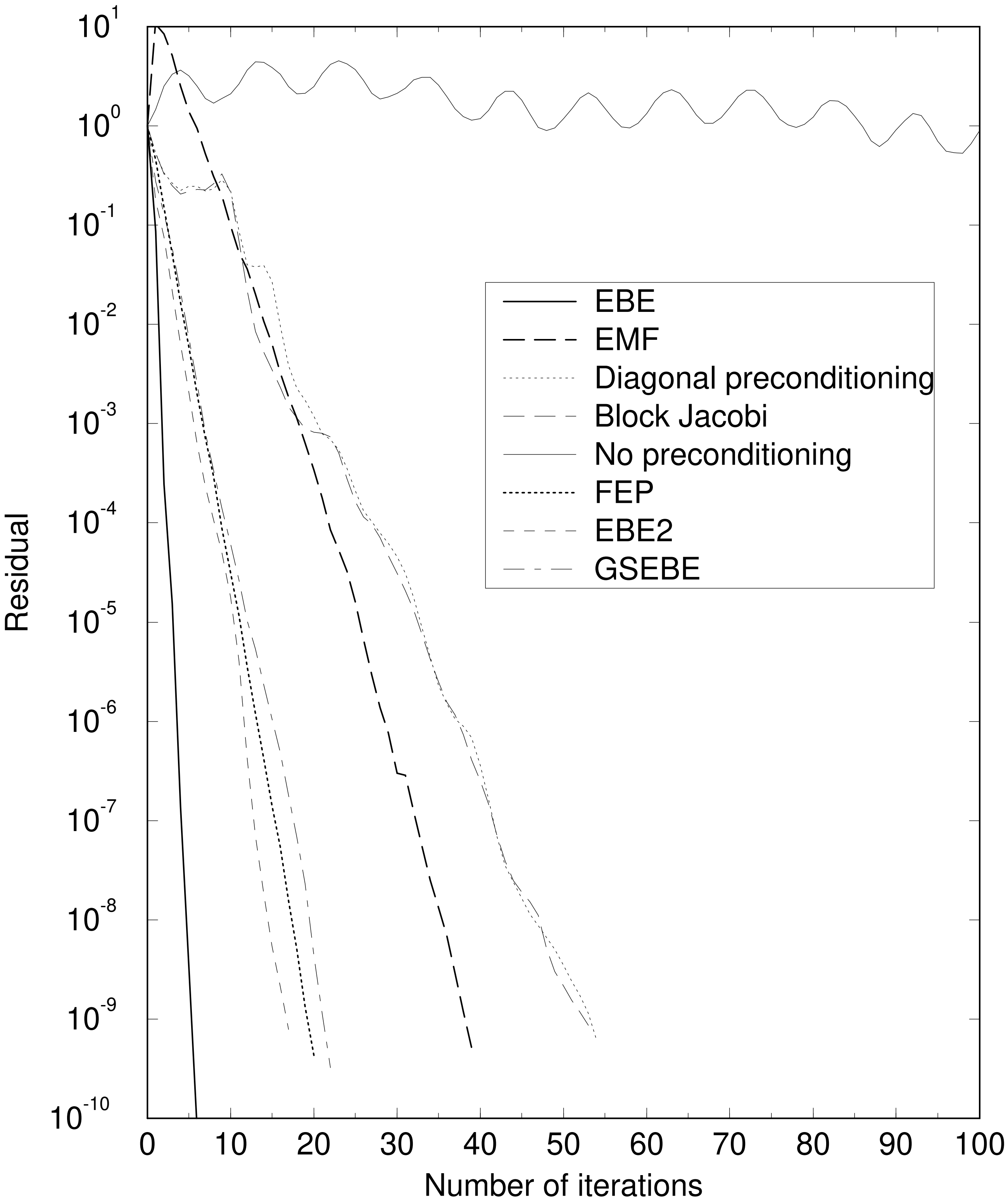

Figure 4.3: Convergence of the conjugate gradient for a matrix of type II with $P = 150$, $K = 12$ and $P = 2$

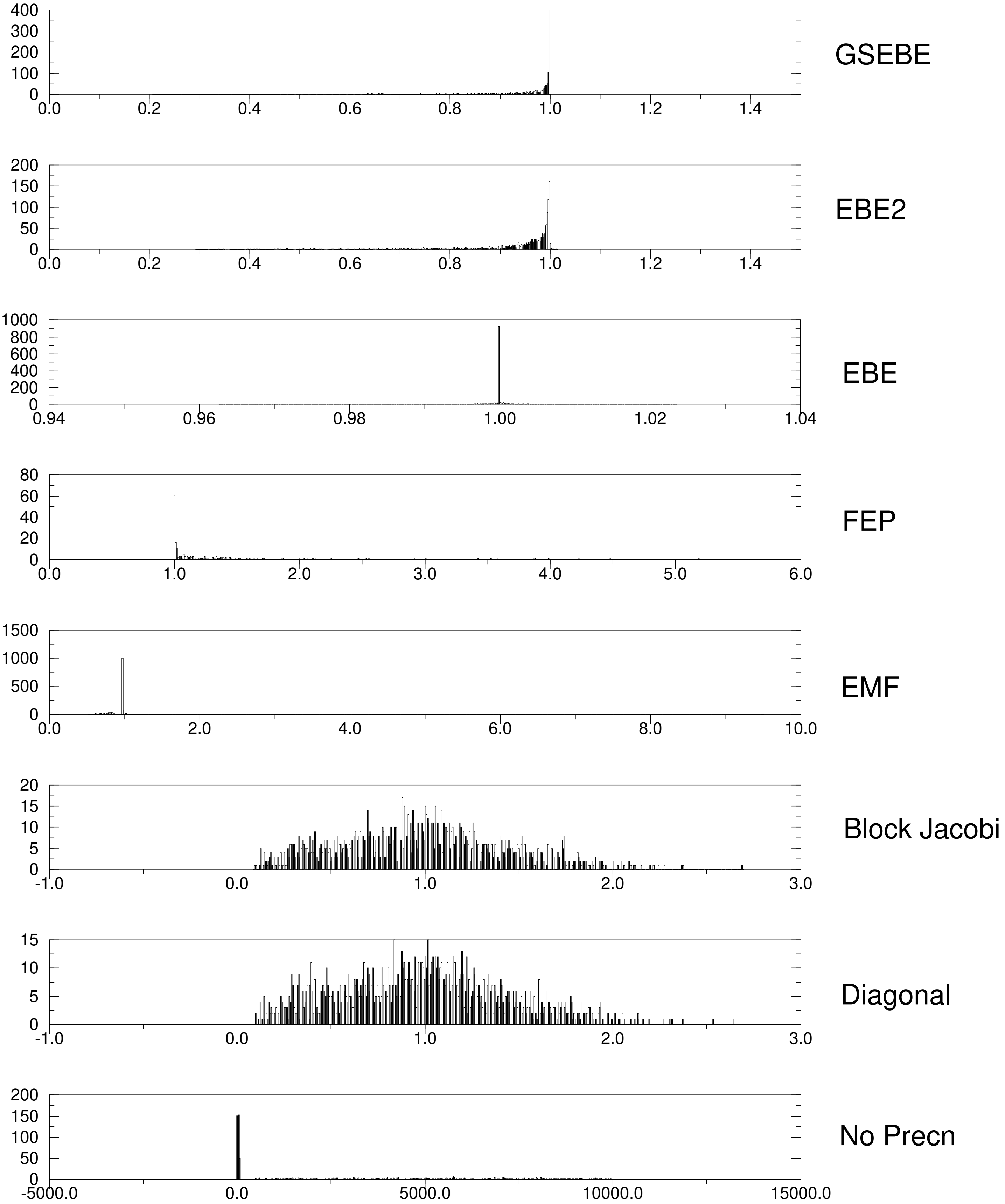

Figure 4.4: Eigenvalues of the preconditioned matrix for a matrix of type II with $P = 150$, $K = 12$ and $P = 2$

Over-lap	Order of $\boldsymbol{H}$	Preconditioner							
		None	Diagonal	Block Jacobi	EBE	EBE2	GS EBE	EMF[1]	FEP[1]
0	1800	3944/186	116/5.7	116/6.0	1/0.51	36/9.7	47/9.0	1	1
1	1651	2507/118	76/3.8	73/6.5	6/1.5	24/6.5	31/5.8	35	17
2	1502	1721/79.5	54/2.7	53/5.1	6/1.5	17/4.5	22/4.1	39	20
3	1353	1372/62.9	49/2.4	48/5.0	6/1.4	16/4.2	20/3.6	36	18
4	1204	1132/51.3	49/2.4	45/4.9	5/1.2	16/4.1	20/3.4	31	15
5	1055	822/36.7	49/2.3	42/4.9	5/1.1	16/4.0	20/3.3	26	13
6	996	389/17.3	48/2.3	35/4.3	5/1.1	16/3.9	20/3.2	13	6

Table 4.3: Results for a matrix of type II with 150 blocks of size 12

4.2 Tests on real matrices

EBE, EBE2, EMF and FEP have already proved to be competitive on finite element problems. FEP and EMF were studied by Kaasschieter (1989) while Gustafsson and Lindskog (1986) reported on EMF. EBE and EBE2 have been shown to be efficient for 2D and especially for 3D problems (see Hughes et al., 1987, Erhel et al., 1991, Wathen, 1989). In this section, we aim to consider matrices which arise from both PDE and optimization applications.

4.2.1 Set of test matrices

Problem name	Order	Number of elements	Min element size	Max element size	Mean element size	Degree of overlap	Condition number
CEGB2802	2694	108	42	60	58.7	2.4	2.5×10^2
CEGB2802	2694	108	42	60	58.7	2.4	5.7×10^4
MAN5976	5882	785	20	20	20.0	2.7	5.0×10^1
LOCK3491	3416	684	6	24	19.8	4.0	1.3×10^2
MAT32	57	157	1	3	2.2	6.0	1.3×10^1
MAT33	637	273	1	3	2.6	6.0	5.5×10^1
BIGGSB1	998	1001	0	2	2.0	2.0	4.0×10^5
TORSION1	3360	3792	1	5	4.4	5.0	6.7×10^0
NOBNDTOR	480	562	1	5	4.2	4.9	1.8×10^2
CBRATU3D	4934	4934	5	8	7.5	7.5	3.4×10^1
NET3	512	531	1	6	2.6	2.7	2.4×10^9

Table 4.4: Summary of the characteristics of each test problem

The matrices come either from the Harwell-Boeing collection (CEGB2802, MAN5976, LOCK3491), see Duff, Grimes and Lewis (1989), or are problems in SIF format from the CUTE collection, see Bongartz, Conn, Gould and Toint (1993). In the case of the Harwell-Boeing, we removed all rows and columns which corresponded to unused variables. Furthermore, we have used random numerical values because they were originally not present. Since the choice of these values influences the conditioning of the matrices, and since this conditioning is relevant to the performance of our solution techniques, we have created two instances of CEGB2802 with significantly differing spectra.

The patterns of CEGB2802 and LOCK3491 arise from structural engineering problems; MAN5976 comes from deformation problems; MAT32 and MAT33 are finite element matrices generated with the SPARSKIT software (see Saad (1990)), TORSION1 and NOBNDTOR are quadratic elastic torsion problems arising from as an obstacle problem on a square, NET3 is a very ill-conditioned example which arises from the optimization of a high-pressure gas network and CBRATU3D is obtained by discretizing a complex 3D PDE problem in a cubic region.

A summary of each problem characteristics is given Table 4.4. The degree of overlap is the average number of elements containing each variable, that is the sum of the element dimensions divided by the order of the matrix.

4.2.2 Results on test matrices

The results of the execution of the Preconditioned Conjugate Gradient Algorithm on our test problems are given in Tables 4.5 and 4.6. In this table, we present the time spent to calculate the preconditioner, the number of iterations required for convergence, the time for convergence, the time per iteration and the logarithm of the norm of the final residual obtained for different preconditioners.

We observe that, unlike the structured matrices of Section 4.1, element-by-element preconditioners do not appear to be significantly more effective than diagonal preconditioning on many of the current test examples. However we notice that, as we might hope, these preconditioners do appear to be more effective than their diagonal counterparts for the ill-conditioned problems. For example, the ratio EBE/diagonal for the Harwell-Boeing matrix CEGB2802 significantly improves as the conditioning changes.

4.3 Remarks on the storage and the vectorization

Thus far, we have reported on experiments where packed storage is used for element matrices, that is, where we store only the lower triangular part of each element. In Table 4.7, we compare the time per iteration for runs using the matrix CEGB2802 in which we consider both packed or full storage of the element matrices. We performed the tests on a variety of preconditioners and consider the performance both with and without vectorization. Only the inner loops were vectorized, with vectors of length no larger than 60.

It appears that the full storage is more efficient for vectorized conjugate gradients with no or diagonal preconditioning and less efficient for preconditioners involving triangular solves. This is due to the relative efficiency of the computational kernels used both for the matrix-vector products and the triangular solves. An interesting strategy – if extra workspace is available – would be to use full storage for the Hessian, because of the beneficial effect on matrix-vector products, and packed storage for the preconditioners, because of the advantages this gives for triangular solves.

The gain due to vectorization is between 2 and 5, depending on the preconditioner and the storage applied. We would expect to improve these ratios for example with longer vectors or more powerful vector computers.

Instead of using factorizations of the $\boldsymbol{W}_i$, it might be better to form explicit inverses, as the sequence of solves may then be replaced by a sequence of matrix-vector products. This has some advantage in terms of using tuned BLAS routines to form the products. For EBE2, this could be especially useful since two triangular solves would be replaced by only one matrix-vector product. Unfortunately, the elements in our examples are typically small and we did not observe any benefit from such a strategy.

Problem name	Precondi-tioner	Calculating the preconditioner	Number of iterations	Time for convergence	Time per iteration	Residual norm (log10)
CEGB2802	None	0	121	29.0	0.24	-9.1
	Diagonal	0.035	35	8.6	0.24	-9.0
$k = 2.5 \times 10^2$	EBE	6.4	12	7.7	0.59	-9.5
	EBE2	6.3	11	10.9	0.90	-9.3
	GS EBE	0.68	17	10.5	0.58	-9.4
CEGB2802	None	0	2040	481.5	0.24	-9.0
	Diagonal	0.035	661	158.4	0.24	-9.1
$k = 5.7 \times 10^4$	EBE	6.4	129	75.7	0.58	-9.0
	EBE2	6.3	145	131.0	0.90	-9.0
	GS EBE	0.68	362	211.2	0.58	-9.0
MAN5976	None	0	68	22.7	0.33	-9.1
	Diagonal	0.081	28	9.8	0.33	-9.1
$k = 5.0 \times 10^1$	EBE	5.2	10	10.1	0.91	-9.4
	EBE2	5.0	11	17.5	1.45	-9.6
	GS EBE	0.85	12	11.9	0.91	-9.3
LOCK3491	None	0	69	20.0	0.29	-9.1
	Diagonal	0.068	24	7.3	0.29	-9.4
$k = 1.3 \times 10^2$	EBE	4.7	9	7.9	0.78	-9.3
	EBE2	4.5	9	12.7	1.26	-9.3
	GS EBE	0.71	11	9.4	0.78	-9.3
MAT32	None	0	21	0.36	0.013	-9.2
	Diagonal	0.014	21	0.36	0.013	-9.2
$k = 1.3 \times 10^1$	EBE	0.060	14	0.71	0.043	-9.3
	EBE2	0.053	12	0.97	0.069	-9.1
	GS EBE	0.027	15	0.76	0.043	-9.8
MAT33	None	0	48	2.2	0.043	-9.1
	Diagonal	0.028	48	2.2	0.043	-9.1
$k = 5.5 \times 10^1$	EBE	0.22	28	4.9	0.17	-9.2
	EBE2	0.19	24	7.1	0.28	-9.0
	GS EBE	0.076	29	5.1	0.17	-9.2

Table 4.5: Results for our test problems (I). The times are in seconds

Problem name	Precondi-tioner	Calculating the preconditioner	Number of iterations	Time for convergence	Time per iteration	Residual norm (log10)
BIGGSB1	None	0	499	29.5	0.059	-9.3
	Diagonal	0.035	499	30.1	0.060	-9.3
$k = 4.04 \times 10^5$	EBE	0.29	276	67.3	0.24	-9.0
	EBE2	0.24	328	133.6	0.41	-9.0
	GS EBE	0.11	376	90.7	0.24	-9.0
TORSION1	None	0	24	8.5	0.34	-9.0
	Diagonal	0.14	25	8.9	0.34	-9.3
$k = 6.7 \times 10^0$	EBE	2.2	12	16.1	1.23	-9.1
	EBE2	1.9	11	24.9	2.07	-9.7
	GS EBE	0.62	12	16.0	1.22	-9.0
NOBNDTOR	None	0	68	3.5	0.050	-9.0
	Diagonal	0.028	68	3.6	0.051	-9.2
$k = 1.8 \times 10^2$	EBE	0.33	35	6.5	0.18	-9.2
	EBE2	0.30	33	10.3	0.30	-9.2
	GS EBE	0.096	37	6.8	0.18	-9.3
CBRATU3D	None	0	53	31.6	0.58	-9.1
	Diagonal	0.20	53	31.8	0.59	-9.1
$k = 3.4 \times 10^1$	EBE	5.4	24	49.4	1.97	-9.2
	EBE2	5.0	20	69.2	3.29	-9.2
	GS EBE	1.2	25	50.8	1.96	-9.2
NET3	None	0	3755	142.4	0.038	-9.0
	Diagonal	0.025	1561	59.9	0.038	-9.0
$k = 2.4 \times 10^9$	EBE	0.20	628	89.7	0.14	-9.1
	EBE2	0.18	559	133.2	0.24	-9.0
	GS EBE	0.072	742	105.5	0.14	-9.1

Table 4.6: Results for our test problems (II). The times are in seconds

Preconditioner	scalar mode		vector mode	
used	Packed	Full	Packed	Full
None	0.613	1.02	0.235	0.214
Diagonal	0.618	1.03	0.237	0.217
EBE	1.52	1.98	0.582	0.594
EBE2	2.40	2.92	0.897	0.970
GS EBE	1.52	1.97	0.582	0.591

Table 4.7: Time per iteration in CEGB2802

4.4 Testing the generalized EBE preconditioner

To date, the generalized EBE preconditioner has only been tested on matrices with regular patterns like in Figure 4.2. We do not believe that additional tests are necessary at this stage, as we are able to predict the likely overall behaviour of the generalized EBE preconditioner from the tests we report here.

		Preconditioner			
Overlap	Order	None	Diagonal	EBE	GEN EBE
0	500	102/1.7	116/2.0	1/0.20	1/0.20
1	451	111/1.8	100/1.6	19/1.3	19/2.6
2	402	102/1.7	86/1.4	20/1.3	18/2.5
3	353	96/1.5	70/1.2	20/1.2	17/2.3
4	304	84/1.3	59/0.97	17/0.99	15/2.1
5	255	59/0.96	50/0.83	18/0.99	12/1.7

Table 4.8: Comparison of EBE and GEN EBE on a well conditioned problem (number of iterations / time to converge)

Table 4.8 gives a comparison between EBE and GEN EBE for a problem whose structure is described in Figure 4.2 and for which the logarithms of the eigenvalues in each element are randomly chosen between -1 and 1. There are 50 elements, each of dimension 10. We allow the overlap to vary and report on the effect of this in the table. For these matrices, we observe that GEN EBE appears to be more effective as the overlap increases.

		Preconditioner			
Overlap	Order	None	Diagonal	EBE	GEN EBE
0	500	>5000 / >76.5	>5000 / >79.3	1/0.20	1/0.20
1	451	>5000 / >76.5	4272/67.1	253/15.2	253/32.7
2	402	4306/65.0	2617/40.5	381/21.6	421/53.4
3	353	2584/38.6	1823/27.9	359/19.4	414/53.2
4	304	1390/20.5	998/15.1	223/11.5	246/30.2
5	255	664/9.9	552/8.2	171/8.3	167/20.5

Table 4.9: Number of iterations and time for convergence of EBE and GEN EBE on a badly conditioned problem (number of iterations/time to converge)

In Table 4.9, we illustrate the same effects on a worse conditioned example. The structure is as before, but the logarithms of the eigenvalues in each element are randomly chosen between -3 and 3. The results here are much less clear; no overall trend is observed.

Finally, in Table 4.10, we report on an experiment in which the matrix is the same as that examined in Table 4.8 except that all the entries in positions where two elements overlap have been multiplied by 10^5. GEN EBE is much more efficient than EBE on these problems, but we should note that they have been especially constructed to exhibit good behaviour of the former method and that, alas, we are unlikely to observe such a difference in practice.

We observe that GEN EBE is not very effective according to Tables 4.8 and 4.9. Except in the extreme case of Table 4.10, EBE is always better when CPU times are compared. On well-conditioned problems, we observe a reasonable improvement in the number of iterations required,

Overlap	Order	Preconditioner			
		None	Diagonal	EBE	GEN EBE
0	500	102/1.7	116/2.0	1/0.20	1/0.20
1	451	853/13.1	88/1.5	2/0.25	2/0.46
2	402	1058/16.1	68/1.14	9/0.64	2/0.45
3	353	913/13.7	46/0.79	11/0.72	2/0.45
4	304	766/11.4	34/0.60	12/0.74	2/0.44
5	255	44/0.74	31/0.55	13/0.74	1/0.32

Table 4.10: Number of iterations and time for convergence of EBE and GEN EBE on a special problem

but this is insignificant when CPU times are compared. Indeed, the time per iteration can be much larger for GEN EBE than for EBE, especially if there is much overlapping. Thus, the results are not encouraging. This is undoubtedly partially because, even if the $\boldsymbol{H}_i - \boldsymbol{M}_i$ are small compared to $\boldsymbol{H}_i$, the matrices $\boldsymbol{E}_i = \boldsymbol{L}_M^{-1}(\boldsymbol{H}_i - \boldsymbol{M}_i)\boldsymbol{L}_M^{-T}$ are not necessarily small. Furthermore the norms $\|\boldsymbol{L}_M \boldsymbol{E}_i \boldsymbol{E}_j \boldsymbol{L}_M^T\| = \|(\boldsymbol{H}_i - \boldsymbol{M}_i)\boldsymbol{M}^{-1}(\boldsymbol{H}_j - \boldsymbol{M}_j)\|$ may be significantly bigger than is usually observed for classical EBE, especially on badly conditioned problems. It would be interesting to derive different ways of decreasing the errors introduced in the approximate factorizations, perhaps taking into account the values of the elemental Hessians and not just their patterns.

4.5 Conclusions on the use of element-by-element preconditioners

The results of the previous sections indicate that element-by-element preconditioners are effective, in terms of the numbers of iterations required and the clustering of eigenvalues of the preconditioned Hessian, particularly if the overlap between blocks is small. EBE seems to be the best of our block preconditioners and it does not require any assembly of the matrix. EMF and FEP do not require an assembly of the matrix either but the resulting triangular incomplete factors need to be partially assembled, which can make each solve rather costly.

A disadvantage of EBE2 and GS EBE is that the terms $\frac{1}{4}\boldsymbol{E}_i^2$ and $\boldsymbol{L}_i \boldsymbol{W}^{-1} \boldsymbol{L}_i^T$ may give rise to poor approximations even when there is little overlap between elements. If there is significant overlap, the efficiency of EBE2 and GS EBE is close to that observed for EBE.

Usually, the number of iterations required by EBE is smaller than that for EBE2 which is, in turn, smaller that that for GS EBE — the only case where this is not so is for problems with very large overlap. As GS EBE is the easiest preconditioner to construct, it may be beneficial to use GS EBE when we do not need much accuracy in the solution of our system, as, for example, is common in the early stages of optimization calculations. However in more general cases, we prefer EBE.

We have not found it useful to replace the triangular solves by products of inverses. To obtain some benefit from this, we would need larger elements so as to exploit better vectorization/parallelization.

A difficulty for general matrices is that we have no *a priori* information on the sizes of the elements. On finite-element problems, typically all elements have the same size. We may thus *colour* the elements — that is, partition the complete set of elements into subsets, or colours, of independent (non-overlapping) elements — to encourage both vectorization and parallelization. In our case, vectorization is restricted to the treatment of each individual element, so is not usually very effective. A potentially better strategy would be to vectorize at a coarser level, treating blocks of the same size within each colour together.

In our experiments, except for ill-conditioned problems, EBE is not significantly more efficient than diagonal preconditioning. We believe that this is for three reasons. The first is because of the structure of the elements. When there is lower overlap, EBE appears much more efficient than diagonal preconditioning (see section 4.1). Amalgamating elements may reduce the number of iterations by decreasing the degree of overlap in the new partition. The second is that vectorization here is not as efficient as it could be. If we knew *a priori* that all blocs have the same size, it would be possible to vectorize the solve more efficiently, as was reported in previous experiments by, for example, Erhel et al. (1991). The third is that no colouring, parallelization and specific code optimizations have been yet been carried out.

Finally, we believe that further study, on real problems with less regular matrix structures, will inevitably lead to a better understanding of the classes of problems for which each of the preconditioners we have considered is particularly appropriate.

5 Remarks on the importance of element regrouping

Let $f(\boldsymbol{x})$ be of the form (1.2). Clearly, the decomposition (1.2) may not be unique, and different decompositions may significantly affect the performance of the preconditioners considered in this paper. In some of the experiments reported in the previous chapter, the limitation in performance of the conjugate gradient method, both for vectorization and parallelization, is due to a non-optimal choice of the decomposition. Indeed, frequently the local variable set for one element may be completely contained within another and it would pay to merge the two elements into a single super-element or *group*. More generally, the local variable sets for two elements may significantly overlap and again it may be advantageous to merge the elements into a single group.

Conn, Gould and Toint (1993) have considered this problem from the point of view of improving the matrix-vector products at the heart of many iterative methods for minimizing partially separable functions. In this section our scope is larger in that we hope to produce more effective preconditioners by amalgamating similar elements. We need to take into account the following goals:

- Amalgamation should aim to merge elements which involve many common variables; this improves the behaviour of the element-by-element preconditioners and decreases the amount of work;
- In the case of generalized EBE, amalgamation may allow bigger blocks $\boldsymbol{M}_i$ in the matrix $\boldsymbol{M}$;
- If we wish to vectorize outside the elements (longer vectors than in the case of vectorization inside the treatment of each element), amalgamation should aim to keep many elements of the same size. Each routine would then be applied to vectors of elemental matrices instead of elemental matrices themselves and this should make vectorization very efficient. For the moment, we do not take this point into account because we do not try to vectorize outside the elements.

It is clear that determining an optimal partitioning of the elements into groups may be costly, or even impossible. Frequently, the construction of an initial decomposition of f into elements by a user depends more on considerations on the easiness of expressing the function and its derivatives, rather than considerations of computational performance. It is our experience with many of the test examples in the CUTE package (see, Bongartz et al., 1993), for example, that the overlap between elements is high (and even that some elements are entirely subsumed by other elements). Therefore, we regard it to be crucial for performance to determine a heuristic to partition the original sets of elements into computationally attractive disjoint groups.

5.1 Goals of the regrouping technique

Let $\boldsymbol{x} = (x_1, x_2, \cdots, x_n)^T$, $\boldsymbol{x}^i$ be the set of local variables involved in the elementary function f_i, for $1 \leq i \leq p$ and $\mathcal{V}_i$ be the set of indices of $\boldsymbol{x}^i$. For example, if

$$f(x_1, x_2, x_3) = f_1(x_1, x_2) + f_2(x_2, x_3),$$

we have

$$\boldsymbol{x}^1 = \{x_1, x_2\}, \boldsymbol{x}^2 = \{x_2, x_3\}, \mathcal{V}_1 = \{1, 2\} \quad \text{and} \quad \mathcal{V}_2 = \{2, 3\}.$$

Our aim is to partition the index set of elements, $\mathcal{P} \stackrel{\text{def}}{=} \{1, \ldots p\}$, into a collection of *groups* $\{\mathcal{G}_k\}$ such that

1. $\bigcup \mathcal{G}_k = \mathcal{P}$,
2. $\mathcal{G}_k \bigcap \mathcal{G}_l = \emptyset$ for $k \neq l$.
3. If elements i and $j \in \mathcal{G}_k$, either $\mathcal{V}_i \setminus \mathcal{V}_j$ or $\mathcal{V}_j \setminus \mathcal{V}_i$ is small.
4. If element $i \in \mathcal{G}_k$ and element $j \in \mathcal{G}_l$, $(k \neq l)$, both $\mathcal{V}_i \setminus \mathcal{V}_j$ and $\mathcal{V}_j \setminus \mathcal{V}_i$ are large.

Thus, each group aims to collect elements whose overlap is large, while keeping the overlap between groups small.

Once the groups have been determined, all of the elements indexed by a single group will be summed to form a *super-element*. Thereafter, the algorithms described in Sections 2 – 4 of this paper should be applied to the sum of super-elements,

$$f(\boldsymbol{x}) = \sum_k s_k(\boldsymbol{x}), \quad \text{where} \quad s_k = \sum_{i \in \mathcal{G}_k} f_i(\boldsymbol{x}^i). \tag{5.1}$$

5.2 Possible amalgamation algorithms

A natural approach, frequently used in sparse linear algebra, (see Duff and Reid, 1983, and Amestoy, 1991) is to amalgamate two elements into a group if their overlap is large. The main difficulty is to define a suitable heuristic to control the amalgamation process.

We start by assigning the index of each element to its own group. We call such a group an *elementary* group and denote the group as $\mathcal{G}_i = \{i\}$, $1 \leq i \leq p$. The algorithm proceeds by merging groups until a satisfactory partitioning has been determined. Once a group comprises two or more elements, it will be called an *amalgamated* group. By convention, if we merge groups $\mathcal{G}_i$ and $\mathcal{G}_j$, with $i < j$, we replace $\mathcal{G}_i$ by $\mathcal{G}_i \bigcup \mathcal{G}_j$ and delete $\mathcal{G}_j$. We denote the set of indices of variables used by elements in group $\mathcal{G}_k$ by $\mathcal{V}_k$.

A simple amalgamation algorithm is as follows. The *fill-in* between groups i and j is defined to be twice the product of the cardinalities of the sets $\mathcal{V}_i \setminus \mathcal{V}_j$ and $\mathcal{V}_j \setminus \mathcal{V}_i$ — this is in fact the number of extra (zero) entries introduced if the two group Hessians are amalgamated. Of course, this fill-in should be limited to reasonable value, so we let h_{max} be the maximum fill-in value allowed when amalgamating two groups. Then we amalgamate pairs of groups for which the fill-in is smallest, until any remaining amalgamation would produce a fill-in value larger than h_{max}. Ties are broken arbitrarily.

Another approach, suggested by Conn et al. (1993) considers the number of flops gained in the matrix-vector product. A list of elements is built for each variable and two elements are amalgamated if it leads to decrease the number of floating point operations in the matrix-vector product.

In this paper, we have chosen to concentrate on an algorithm that aims to decrease the time spent in matrix-vector products and triangular solves on the target architecture. Our overall goal is to reduce the cost of a (preconditioned) conjugate gradient iteration. As the dominant cost of such an iteration is in solving a system involving the preconditioner and in forming a matrix-vector product, it is of interest to reduce these costs.

5.3 An amalgamation algorithm

As a precursor to the main amalgamation algorithm, we apply the following scheme to merge elements which are completely subsumed within others and to construct an *amalgamation* graph. The amalgamation graph has as its nodes the groups and has arcs between the nodes for which there is some benefit from amalgamation. The arcs have associated weights, $b(\mathcal{G}_i, \mathcal{G}_j)$, which indicate the possible benefit to be obtained by merging nodes.

Initial phase (suppress complete inclusions and construct the amalgamation graph)
For each pair of groups $\mathcal{G}_i$ and $\mathcal{G}_j$, $i < j$, such that $\mathcal{V}_i \cap \mathcal{V}_j$ is not zero,
If $\mathcal{V}_i \setminus \mathcal{V}_j$ or $\mathcal{V}_j \setminus \mathcal{V}_i$ is zero,
    Amalgamate these groups.
Else
    Compute the benefit $b(\mathcal{G}_i, \mathcal{G}_j)$
    Add the arc $(\mathcal{G}_i,\ \mathcal{G}_j)$ with weight $b(\mathcal{G}_i, \mathcal{G}_j)$ to the graph
End if

We then apply the main amalgamation algorithm.

Main amalgamation algorithm (simplified)
While there exists an arc for which b is positive
    Find the arc $\mathcal{G}_i,\ \mathcal{G}_j,\ b(\mathcal{G}_i, \mathcal{G}_j)$ of the graph for which b is maximum
    Amalgamate the groups $\mathcal{G}_i$ and $\mathcal{G}_j$
    Update all the arcs of the graph incident on $\mathcal{G}_i$ or $\mathcal{G}_j$
End while
End of the algorithm. The new partitioning into groups has been obtained.

5.4 Computing the benefit

The choice of amalgamating two elements is based on the estimated gain of CPU time or *benefit* of this amalgamation which is defined as

$$b(\mathcal{G}_i, \mathcal{G}_j) = t(card(\mathcal{V}_i)) + t(card(\mathcal{V}_j)) - t(card(\mathcal{V}_i \bigcup \mathcal{V}_j)), \tag{5.2}$$

where $t(i)$ is the estimated time to treat an element of order i.

The most costly operations in a preconditioned conjugate gradient iteration are

- p elemental size matrix-vector products, and
- q elemental size triangular solves,

where p is the number of elements, and q is

- zero for no or diagonal preconditioning,
- two times p for EBE and GS EBE preconditioning, and
- four times p for EBE2 preconditioning.

In our current experiments, we shall only consider diagonal and EBE preconditioning, as these proved to be the most effective of the preconditioners that we investigated in Section 4. Thus we have two different possibilities:

Minimize the time spent in matrix-vector products. In what follows, this variant will be called **amalg1**. It should be optimal for conjugate gradients without preconditioner or with diagonal preconditioning. In this case, $t(i)$ is an estimate of the time spent to perform a matrix-vector product of order i on the considered architecture. We consider the case where the element matrix is symmetric, stored in packed symmetric storage and the accesses to the vectors are indirect.

Optimize the ordering for use of the EBE (or GS EBE) preconditioner. This variant is called **amalg2**. The time estimate, $t(i)$, is now the time spent in a matrix-vector product plus twice the time of a triangular solve with indirect accesses to the right-hand-side and the solution.

Note that the time spent in constructing the preconditioner is not taken in account in these costs, as this is an overhead for the whole conjugate gradient process, not just for an individual iteration. The time spent in diagonal products — the matrix-vector products for the diagonal preconditioner or the three products associated with the EBE preconditioner — is not taken into account either, but such products are independent of the ordering.

The times of matrix-vector products and triangular solves for all realistic values of i and computed once and stored in a pair of data files. The values $t(i)$ are read from these files as required and stored in a real array. The benefit $b(\mathcal{G}_i, \mathcal{G}_j)$ can then easily be computed from (5.2). Throughout the algorithm, estimates of the total benefit along with the total time for matrix-vector products and triangular solves before amalgamation are recorded.

5.5 Preliminary tests

We report the results of first running the amalgamation algorithm and then using diagonal and EBE preconditioned-conjugate gradients to solve the linear system in question. Our implementation is principally in Fortran but the amalgamation graph manipulation is coded in C. All the times reported in this section are in seconds.

It is clear from Table 5.11 that amalgamation can be very effective. There are often large gains in convergence times, both for diagonal and EBE preconditioning. As we would have expected, it appears that, in most cases, the diagonal preconditioned method is faster with **amalg1** and the EBE is faster with **amalg2**. For some other problems, such as CEGB2802, however, amalgamation is not so useful as the function is already well decomposed. For some problems, **amalg2** is less effective than **amalg1** with an EBE preconditioner, which shows the limits of our heuristic. Such limitations likely arise because of the indirect addressing of data and the fact that we do not know *a priori* how the data is accessed in the memory and hence the experiments do not fit exactly with the estimates. The second reason is that the eigenvalue-clustering quality of the preconditioner may happen to be worse with **amalg2** than **amalg1**. This happens, for example, for the test problem MAT33.

While amalgamating does not normally effect the quality of the diagonal preconditioner (i.e., the number of iterations), it improves significantly the numerical behaviour of the EBE-preconditioned conjugate-gradients method. However, when considering, for example, the test problem NET3, we observe a variation of the number of iterations with diagonal preconditioning. This is entirely because the order of floating-point operations is altered by the amalgamation, and different rounding properties come into effect (see Higham, 1993) when computing the diagonal preconditioner and matrix-vector products. When considering the EBE preconditioned method, we observe that the convergence time is about 50 times faster with **amalg2** than with the original matrix. This may be attributed to the decrease ($628 \rightarrow 145$) in the number of iterations due to a better preconditioner and to the reduction in time (0.17 s $\rightarrow$ 0.034 s) per iteration.

	Amalgamation time		Number of elements	Average element size	Diagonal			EBE		
	symb	num			Number of its	Prec. constr	Conv time	Number of its	Prec. constr	Conv time
NET3			538	2.58	1561	0.023	57.3	628	0.20	89.7
NET3.amalg1	1.2	0.1	48	12.6	1582	0.011	23.2	171	0.14	6.3
NET3.amalg2	1.2	0.1	42	14.1	1670	0.011	24.6	145	0.15	5.4
BIGGSB1			1001	1.99	499	0.035	30.1	276	0.29	67.3
BIGGSB1.amalg1	1.37	0.18	64	16.6	499	0.013	12.3	104	0.21	6.4
BIGGSB1.amalg2	1.37	0.18	64	16.6	499	0.013	12.3	104	0.21	6.4
TORSION1			3792	4.42	25	0.14	8.9	12	2.2	16.0
TORSION1.amalg1	20.5	1.7	262	22.7	25	0.036	3.67	10	1.4	4.0
TORSION1.amalg2	21.4	1.8	198	28.0	25	0.033	3.68	10	1.5	3.9
MAT32			157	2.18	21	0.014	0.36	14	0.061	0.71
MAT32.amalg1	0.27	0.030	4	16.5	21	0.0084	0.11	10	0.030	0.095
MAT32.amalg2	0.27	0.032	3	21.3	21	0.0085	0.12	10	0.033	0.096
MAT33			637	2.57	48	0.028	2.2	28	0.22	4.9
MAT33.amalg1	2.09	0.15	19	20.2	48	0.0098	0.55	21	0.11	0.55
MAT33.amalg2	1.98	0.16	18	21.3	48	0.0097	0.55	24	0.11	0.62
CEGB2802 $k = 2.5 \times 10^2$			108	58.7	35	0.035	8.5	12	6.4	7.7
CEGB2802.amalg1	0.7	2.0	106	59.4	35	0.033	8.5	12	6.4	7.6
CEGB2802.amalg2	0.7	2.0	102	60.8	35	0.033	8.6	12	6.4	7.6
CEGB2802 $k = 5.7 \times 10^4$			108	58.7	661	0.035	156.7	129	6.4	75.7
CEGB2802.amalg1	0.7	2.0	106	59.4	660	0.034	155.5	111	6.4	65.1
CEGB2802.amalg2	0.7	2.0	102	60.8	659	0.031	156.5	111	6.5	64.9
CBRATU3D			4394	7.54	53	0.20	31.8	24	5.5	48.8
CBRATU3D.amalg1	53.7	3.8	851	25.1	53	0.097	25.6	21	7.3	28.4
CBRATU3D.amalg2	57.9	4.0	764	27.1	53	0.093	25.8	21	7.6	28.3
NOBNDTOR			562	4.16	68	0.028	3.6	35	0.33	6.5
NOBNDTOR.amalg1	2.3	0.2	40	21.8	68	0.012	1.44	29	0.22	1.55
NOBNDTOR.amalg2	2.4	0.2	31	26.3	68	0.014	1.47	30	0.22	1.57

Table 5.11: Comparison of the preconditioned conjugated gradients applied to the original matrices and the matrices with amalgamation strategies 1 and 2 on 1 processor of the Alliant FX/80. Symb and num refer to the time taken to perform the amalgamation and set up data structures for the resulting factors, and to the time required to insert the numerical values into the resulting factors, respectively.

As far as we are concerned, the most important thing to note is that in certain cases, with the best amalgamation strategy, EBE preconditioning is much more efficient than diagonal preconditioning (e.g., 5.4 seconds versus 23.2 seconds for NET3).

5.6 Cost of the algorithm

The times spent in the amalgamation procedure are reported in the second and third columns of Table 5.11. The second column gives the times required to perform the amalgamation and set up data structures for the resulting factors, while the third column shows the additional time required to insert the numerical values into the resulting factors. The algorithm is sequential and has not yet been optimized.

Currently, we consider the reported times for the amalgamation procedure to be significant. Taking these times into account, it would appear that applying the amalgamation algorithm is not always beneficial, particularly if we wish to solve a single linear system and if the number of groups is large (see CBRATU3D). If, however, we have to solve a large number of problems with the same matrix, or which have the same elemental structure — such as might occur in a nonlinear optimization or PDE application — amalgamating the elements is essential.

5.7 Conclusions on the use of element amalgamation

In our examples, we have observed that most of the benefit of amalgamations is due to the Main Amalgamation Algorithm. However it seems natural to start the algorithm with an initial phase in which complete inclusions are suppressed, because it reduces the number of elements without introducing fill-in and saves obviously space and floating-point operations.

Obviously, the implementation of our algorithm is not optimal, but the results obtained are very promising. The cost of the amalgamation procedure is currently rather costly, but it is hoped that good heuristics will decrease this preprocessing cost with roughly the same effect. As we may have to solve many systems with the same structure in the course of a nonlinear optimization calculation, a good preprocessing step may pay handsome dividends in the longer run.

The resulting numerical quality of the preconditioner seems to be difficult to appreciate in general. All that we may hope is that decreasing the degree of overlap may also decrease the number of iterations. But we note that even if the number of iterations following the amalgamation process is not significantly reduced, the times for the calculation of the preconditioner and for obtaining the solution often decrease.

The ratio of EBE to diagonal preconditioner solution times decreases as the amalgamation is applied, both because typically fewer iterations are required following the amalgamation and because an increase in element sizes encourages efficient vectorization for EBE and matrix-vector products — diagonal preconditioning is already completely vectorizable.

6 Final Comments

We have shown that element-by-element preconditioners may be extremely effective for systems of equations which arise in partially separable nonlinear optimization applications. Furthermore, they seem to offer great possibilities of vectorization/parallelization on multi-processor architectures. The next step will be to include such a scheme within a nonlinear optimizer.

It is clear that preprocessing should be applied to any large-scale optimization problem, and that in our case it is important to amalgamate elements and find a suitable colouring for later calculations. This is a difficult task as many criteria need to be taken into account. This preprocessing may well be quite costly, but we expect there to be longer-term payoffs.

There seems to be three main approaches to fully exploiting parallelism:

- Try to keep the elements to be roughly the same size, with the aim of both vectorizing and parallelizing over the elements within a colour;
- Vectorize the computations within each element, while handling the elements within each colour in parallel;
- Apply graph partitionning techniques to decrease overlap between elements and exploit sparsity within large elements.

In either case we will need an outer sequential loop over the colours. In the first case there is both a vector and a parallel loop over the elements — we need many of the elements to be of the same size in each colour as the same calculations will be performed on each element). In the second case there is a parallel loop over the elements and the treatment of each element is vectorized — of course, the length of vectors is limited by the element sizes. In the third case there is a parallel loop over the elements and the vectorization is limited by the sparsity of the elements.

If we intend to solve large problems with small elements, the first approach is certainly the best and was successfully applied to finite element problems. Even if there are some elements with smaller size, their structure should be relaxed. If the elements are large enough with a range of different sizes, we should opt for the second. The third approach may be very interesting for the point of view of numerical quality of the preconditioners. In that case, frontal approaches could be used to compute the preconditioners and different graph partitioning techniques need to be studied both for numerical efficiency of the preconditioners and parallel implementation. However, the granularity may become quite large and a compromise should be found between

- use large elements with low overlap and
- keep enough elements for an efficient parallelization and avoid increasing too much the work at each iteration.

The third approach might be the best for very large ill-conditioned problems.

Anyway, the three implementations are of interest and maybe a dynamic choice is possible. We shall report on this in a future paper.

Finally we aim to implement these preconditioning techniques in the LANCELOT package, profiting from the preprocessing to optimize other parts of the algorithm such as the matrix-vector products and linear solvers.

7 Acknowledgement

The authors are extremely grateful to Iain Duff for his careful reading of a previous draft of this report.

References

P. Amestoy. Factorization of large sparse matrices based on a multifrontal approach in a multiprocessor environment. PhD Thesis TH/PA/91/2, CERFACS, Toulouse, France, 1991.

O. Axelsson. Solution of linear systems of equations: Iterative methods. *In* V. A. Baker, ed., 'Sparse Matrix Techniques (Copenhagen 1976)', Springer Verlag, Berlin, 1976.

I. Bongartz, A. R. Conn, N. I. M. Gould, and Ph. L. Toint. CUTE: Constrained and Unconstrained Testing Environment. Technical Report TR/PA/93/10, CERFACS, Toulouse, France, 1993. To appear in *ACM Transactions on Mathematical Software*.

A. R. Conn, N. I. M. Gould, and Ph. L. Toint. An introduction to the structure of large scale nonlinear optimization problems and the LANCELOT project. *In* R. Glowinski and A. Lichnewsky, eds, 'Computing Methods in Applied Sciences and Engineering', pp. 42–54, SIAM, Philadelphia, USA, 1990.

A. R. Conn, N. I. M. Gould, and Ph. L. Toint. Large-scale nonlinear constrained optimization. *In* J. R. E. O'Malley, ed., 'Proceedings of the Second International Conference on Industrial and Applied Mathematics', pp. 51–70, SIAM, Philadelphia, USA, 1992.

A. R. Conn, N. I. M. Gould, and Ph. L. Toint. Improving the decomposition of partially separable functions in the context of large-scale optimization: a first approach. Technical Report TR/PA/93/16, CERFACS, Toulouse, France, 1993.

I. D. Duff, R. G. Grimes, and J. G. Lewis. Sparse matrix test problems. *ACM Transactions on Mathematical Software*, **15**, 1–14, 1989.

I. S. Duff and J. K. Reid. The multifrontal solution of indefinite sparse symmetric linear equations. *ACM Transactions on Mathematical Software*, **9**(3), 302–325, 1983.

J. Erhel, A. Traynard, and M. Vidrascu. An element-by-element preconditioned conjugate gradient method implemented on a vector computer. *Parallel Computing*, **17**, 1051–1065, 1991.

P. E. Gill and W. Murray. Newton-type methods for linearly constrained optimization. *In* P. E. Gill and W. Murray, eds, 'Numerical Methods for Constrained Optimization', Academic Press, London, 1974.

P. E. Gill, W. Murray, and M. H. Wright. *Practical Optimization*. Academic Press, London and New York, 1981.

G. H. Golub and C. F. Van Loan. *Matrix Computations*. Johns Hopkins University Press, Baltimore, second edn, 1989.

A. Griewank and Ph. L. Toint. Local convergence analysis for partitioned quasi-Newton updates. *Numerische Mathematik*, **39**, 429–448, 1982*a*.

A. Griewank and Ph. L. Toint. On the unconstrained optimization of partially separable functions. *In* M. J. D. Powell, ed., 'Nonlinear Optimization 1981', pp. 301–312, Academic Press, London and New York, 1982*b*.

A. Griewank and Ph. L. Toint. Partitioned variable metric updates for large structured optimization problems. *Numerische Mathematik*, **39**, 429–448, 1982*c*.

A. Griewank and Ph. L. Toint. On the existence of convex decomposition of partially separable functions. *Mathematical Programming*, **24**, 25–49, 1984.

I. Gustafsson and G. Lindskog. A preconditioning technique based on element matrix factorization. *Computational Methods in Applied Mechanics*, **55**, 201–220, 1986.

M. R. Hestenes. Multiplier and gradient methods. *Journal of Optimization Theory and Applications*, **4**, 303–320, 1969.

N. J. Higham. The accuracy of floating-point summation. *SIAM Journal on Scientific and Statistical Computing*, **14**, 783–799, 1993.

T. J. R. Hughes. *The Finite-Element Method: Linear Static and Dynamic Finite-Element Analysis*. Prentice Hall, Englewood Cliffs, New Jersey, USA, 1987.

T. J. R. Hughes, R. M. Ferencz, and J. O. Hallquits. Large-scale vectorized implicit calculations in solid mechanics on a CRAY X-MP/48 utilizing EBE preconditioned conjugate gradients. *Computational Methods in Applied Mechanics and Engineering*, **61**, 215–248, 1987.

T. J. R Hughes, I. Levit, and J. Windget. An element-by-element solution algorithm for problems of structural and solid mechanics. *Computational Methods in Applied Mechanics and Engineering*, **36**, 241–254, 1983.

B. M. Irons. A frontal solution program for finite-element analysis. *International Journal on Numerical Methods in Engineering*, **2**, 5–32, 1970.

E. F. Kaasschieter. A general finite element preconditioning for the conjugate gradient method. *BIT*, **29**, 824–849, 1989.

M. Ortiz, P. M. Pinsky, and R. L. Taylor. Unconditionally stable element-by-element algorithms for dynamic problems. *Computational Methods in Applied Mechanics and Engineering*, **36**, 223–239, 1983.

M. J. D. Powell. A method for nonlinear constraints in minimization problems. *In* R. Fletcher, ed., 'Optimization', pp. 283–298, Academic Press, London and New York, 1969.

J. K. Reid. Treesolve, a Fortran package for solving large sets of linear finite element equations. Technical Report HL 84/796, AERE Harwell Laboratory, Harwell, UK, 1984.

Y. Saad. Krylov subspace methods: theory, algorithms and applications. *In* 'Computational Methods in Applied Science and Engineering, INRIA 29 January - 2 February', 1990.

R. B. Schnabel and E. Eskow. A new modified Cholesky factorization. *SIAM Journal on Scientific and Statistical Computing*, **11**, 1136–1158, 1991.

A. J. Wathen. An analysis of some element-by-element techniques. *Computational Methods in Applied Mechanics and Engineering*, **74**, 271–287, 1989.

A. J. Wathen. Singular element preconditioning for the finite element method. *In* R. Beauwens and P. de. Groen, eds, 'Iterative methods in linear algebra', pp. 531–540, Elsevier/North Holland, Amsterdam, 1992.

O. C. Zienkiewicz. *The Finite Element Method*. McGraw-Hill, London, 1977.