PERGAMON

Computers and Structures 69 (1998) 655–670

Computers & Structures

A comparison of iterative multi-level finite element solvers

C. E. Jouglard[a], A. L. G. A. Coutinho[b, *]

[a]Laboratorio de Mecánica Computacional, Departamento de Física, Facultad de Ingeniería, Universidad de Buenos Aires, Paseo Colón 850, 1063, Buenos Aires, Argentina

[b]Department of Civil Engineering, COPPE/Federal University of Rio de Janeiro, P.O.Box 68506, Río de Janeiro, RJ 21945-970, Brasil

Received 8 January 1997

Abstract

A comparison is made of two iterative algorithms: Preconditioned Conjugate Gradients (PCG) and Multigrid methods (MG), applying them to a series of test problems of plane elasticity. These problems are discretized by multilevel finite element meshes, that is, a coarse mesh whose elements are successively refined to obtain a fine mesh. In particular, uniform refinement was adopted in conjunction with triangular finite element discretizations, to obtain the hierarchy of meshes needed by the multilevel algorithms. A numerical analysis is made of convergence criteria based on the energy variation of the incremental correction to the solution through the iterative process, which seems to be a more convenient choice to the usual criteria based on the norm of the residual. Performance comparisons are made using diagonal and hierarchical preconditioners, and in all the examples tested the hierarchical PCG is found to be faster than the multigrid solvers. © 1998 Elsevier Science Ltd. All rights reserved.

1. Introduction

Iterative solvers were used by the first pioneers of the finite element (FE) method in the early sixties, but they were soon discarded when it was found that for certain problems the number of operations needed to converge were far beyond the predicted theoretical limits. Iterative solvers were then abandoned and the more effective direct methods based on triangular factorization of the matrix were adopted by structural analysts as their method of choice.

The increasing computing power reached with vector and parallel computers has renewed the interest in iterative solvers, which can exploit the potential of these new architectures more efficiently than with direct solvers, particularly for very large problems (more than 100,000 elements), where a direct method requires huge storage demands which is often the limiting factor to the size of problems that can be solved. On the other hand, one of the most desirable advantages of iterative methods for solving large-scale linear FE equations is the low storage requirements. The coefficient matrix is preserved in its original form and is treated as a linear operator for computing matrix–vector products only, which can be carried out in an element-by-element parallel fashion.

The performance of iterative methods, that is, the number of iterations needed to converge, is strongly influenced by the numerical values of the data of the problem to be solved. An important goal is thus to develop robust iterative methods that perform well for a wide range of data. Among the most efficient iterative solvers we have the *Preconditioned Conjugate Gradient* (*PCG*) [1], whose performance depends on the type of preconditioner employed. Another family of efficient iterative algorithms are the *Multigrid methods* (*MG*) [2, 3, 10, 11] which require a hierarchy of discretizations, that is, several discretizations of increasing density for the same problem must be available. The main purpose of multigrid methods is to pro-

* Corresponding author.

vide an iterative algorithm where the amount of computational work is proportional to the number of unknowns of the finer grid.

If we substitute the standard nodal bases by hierarchical bases to generate the finite element matrices when a mesh corresponding to a given level is refined, the resulting discretization preserves all the information of the previous levels. When multigrid methods are utilized with hierarchical bases, we have the *Hierarchical Basis Multigrid (HBM) method* [7]. The mathematical analysis of the HBM method [7, 12, 13] shows that it shares the same basic properties of the standard multigrid method, but under less stringent conditions.

The multilevel hierarchical preconditioners are robust preconditioners based on some approximation of the hierarchical matrices as a preconditioner for PCG. This preconditioning technique has its basic theory described by Bank and Yserentant [12], Yserentant [14] for the case of symmetric positive definite matrix systems, and was already tested for 3D elastostatic analysis using FE by Coutinho *et al.* [15] and for 2D elastostatic analysis using boundary elements by Barra *et al.* [16], presenting a very good performance when compared with standard preconditioners, such as nodal-block diagonal and the element-by-element Gauss–Seidel preconditioners.

The objective of this work is to describe and compare the performance of PCG and MG solvers on plane elastostatics problems. Two preconditioners were used in this study, a simple diagonal preconditioner and the more complex hierarchical preconditioner. Therefore, this paper is organized as follows. Section 2 describes the preconditioned conjugate gradient algorithm. Section 3 describes the multigrid algorithm. Section 4 describes the hierarchical preconditioner employed in this study. The results of the numerical tests are presented in Section 5. Finally, in Section 6 the main conclusions and related discussion are summarized.

2. The Preconditioned Conjugate Gradient Method

The conjugate gradient method was first introduced by Hestenes and Stiefel [4] in 1952 but it was not until 1970 that the method was presented as an iterative method for the solution of large sparse systems of linear equations [5].

We consider the linear system of equations

$$\mathbf{Kx} = \mathbf{f}, \tag{1}$$

where $\mathbf{K}$ is a $n\ x\ n$ positive definite symmetric matrix.

Solving this linear equation is equivalent to obtain the minimum of the associated function

$$\phi(\mathbf{x}) = 1/2\mathbf{x}^{\mathrm{t}}\mathbf{Kx} - \mathbf{x}^{\mathrm{t}}\mathbf{f}. \tag{2}$$

In the gradient methods, a new approximation $\mathbf{x}^{(k+1)}$ is obtained from $\mathbf{x}^{(k)}$ by the equation

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha^{(k)}\mathbf{p}^{(k)}, \tag{3}$$

which defines a line passing by the point $\mathbf{x}^{(k)}$ in the direction $\mathbf{p}^{(k)}$. After substituting in Eq. (2) we have

$$\begin{aligned}\phi(\mathbf{x}^{(k+1)}) =& 1/2(\alpha^{(k)})^2(\mathbf{p}^{(k)})^{\mathrm{t}}\mathbf{Kp}^{(k)} - \alpha^{(k)}(\mathbf{p}^{(k)})^{\mathrm{t}}\mathbf{r}^{(k)} \\ &- 1/2(\mathbf{x}^{(k)})^{\mathrm{t}}\mathbf{r}^{(k)}, \mathbf{r}^{(k)} = \mathbf{f} - \mathbf{Kx}^{(k)},\end{aligned} \tag{4}$$

where $\mathbf{r}^{(k)}$ is the *residual* vector. Minimizing the function f with respect to a$^{(k)}$, we obtain

$$\alpha^{(k)} = (\mathbf{p}^{(k)})^{\mathrm{t}}\mathbf{r}^{(k)}/(\mathbf{p}^{(k)})^{\mathrm{t}}\mathbf{Kp}^{(k)}. \tag{5}$$

The different gradient methods differ in the direction $\mathbf{p}^{(k)}$ chosen. In the *steepest descent method* the direction $\mathbf{p}^{(k)}$ is chosen as the direction of maximum gradient of the function ϕ at point $\mathbf{x}^{(k)}$. It can be easily proved that this direction is proportional to the residual vector $\mathbf{r}^{(k)}$.

In the *conjugate gradient method* the directions $\mathbf{p}^{(k)}$ are chosen to better represent the directions of steepest descent at point $\mathbf{x}^{(k)}$, but with the additional constraint of being mutually conjugated. Here, the term conjugated means orthogonality with respect to matrix $\mathbf{K}$, that is

$$(\mathbf{p}^{(i)})^{\mathrm{t}}\mathbf{Kp}^{(j)} = 0, \quad \forall i \neq j. \tag{6}$$

The vectors $\mathbf{p}^{(k+1)}$ can be expressed as:

$$\mathbf{p}^{(k+1)} = \mathbf{r}^{(k)} + \beta^{(k)}\mathbf{p}^{(k)}, \tag{7}$$

where the constant β_k is determined by the $\mathbf{K}$-conjugacy property of the direction $\mathbf{p}^{(k)}$.

It can be proved by induction that in this method the next orthogonality conditions are satisfied:

$$\begin{aligned}(\mathbf{r}^{(i)})^{\mathrm{t}}\mathbf{p}^{(j)} =& 0, \quad \forall i > j; \\ (\mathbf{r}^{(i)})^{\mathrm{t}}\mathbf{r}^{(j)} =& 0, \quad \forall i \neq j.\end{aligned} \tag{8}$$

Owing to these orthogonality conditions the correct solution would be theoretically obtained after n steps, where n is the number of equations. However, it must be noted that convergence is rarely attained in exactly n steps. More or less steps are needed. Therefore, the conjugate gradient method must be considered as an iterative method.

Preconditioning is introduced to alleviate difficulties associated with the slow rate of convergence of the conjugate gradient method. Instead of solving the original system, one solves

$$(\mathbf{KB}^{-1})\mathbf{y} = \mathbf{f}, \tag{9}$$

$$\mathbf{x} = \mathbf{B}^{-1}\mathbf{y}$$

The matrix **B** is referred to as the preconditioning matrix. The number of iterations required to solve Eq. (9) depends on the condition number, κ, of its coefficient matrix$\kappa(\mathbf{KB}^{-1})$, which is the ratio of the largest eigenvalue of the eigenproblem $(\mathbf{K} - \lambda\ \mathbf{B})\mathbf{z} = 0$ to the smallest.

Theoretical considerations suggest that at the end of each of the first few iterations the residual norm is reduced by a factor of $\sqrt{[\kappa\ (\mathbf{KB}^{-1}) - 1]}/\sqrt{[\kappa(\mathbf{KB}^{-1}) + 1]}$. Note that when κ is unity a single iteration is sufficient to solve the equation. Of course, κ is one only when $\mathbf{K} = \mathbf{B}$. However this provides us with a guideline for choosing **B**. **B** must be chosen such that one can easily compute the solution of a linear system of equations with **B** as its matrix coefficient while at the same time it is as close to **K** as possible. These are contradictory requirements which makes the problem of finding good preconditioners a challenging one. For a well chosen **B** only a few iterations are required to reduce the residual norm to the desired level.

One of the most simple preconditioners is the *diagonal* preconditioner. In this case the matrix **B** is a diagonal matrix whose diagonal coefficients are those of the system matrix **K**.

2.1. Convergence criteria

The most simple measure of convergence is given by the quotient

$$\|\mathbf{r}^{(k)}\|/\|\mathbf{f}\| < \epsilon, \tag{10}$$

where $\|\cdot\|$ is the Euclidean norm and ϵ is a user-supplied tolerance. Using this approach, a solution is often possible to within machine accuracy.

For engineering purposes, the objective of a numerical analysis is to obtain a discretized solution of partial differential equations within a discretization error. Usually the discretization error is measured in the energy norm $\|\cdot\|_K$ defined for an arbitrary error vector **e** as

$$\|\mathbf{e}\|_K = (\mathbf{e}^t\mathbf{K}\mathbf{e})^{1/2} \tag{11}$$

The error vector **e** represents the difference between the discrete solution obtained from the discretized problem and the exact solution of the continuous problem. Since the exact solution of the continuous problem is generally not available, several estimates are used in practice giving an estimated error **e**. These estimates are based on the solution of the system of Eq. (1), and generally an approximate solution of these equations suffices to provide an adequate estimation of the discretization error. Then an approximated solution $\mathbf{x}^{(k)}$ is considered acceptable if it satisfies

$$\|\tilde{\mathbf{e}}\|_K < \epsilon_e\|\mathbf{x}^{(k)}\|_K, \tag{12}$$

where ϵ_e is the discretization error tolerance in the energy norm.

In the gradient methods we are looking for a minimum of the total potential energy ϕ, Eq. (2), then it makes sense to monitor the energy variation along the iterative process and terminate the iterations when the solution has reached a stabilized value in the energy norm. That is, if $\mathbf{x}^{(k)}$ and $\mathbf{x}^{(k+1)}$ are two successive solutions of the iterative process, we monitor whether or not the next inequality is satisfied in each step

$$\|\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}\|_K < \epsilon_U\|\mathbf{x}^{(k+1)}\|_K, \tag{13}$$

where ϵ_U is the numerical error tolerance in the energy norm.

Strictly speaking, a small value of the left-hand side of the above inequality, which represents the energy variation between two successive steps, does not mean that the right hand side has reached a stabilized value, since the variation of the left-hand side is not monotonic with the iterations, but for practical purposes, and particularly in connection with adaptive analysis, its use appears justified, as showed by tests made by the author [9]. Also, it must be noted that the numerical error tolerance ϵ_U cannot be greater than the discretization error tolerance ϵ_e, since the precision of the numerical solution affects the precision of the error estimator, and, in general, a much smaller value must be imposed to ϵ_U to obtain acceptable results.

If we define U_{cur} as the energy of the current computed solution in step k, and U_{err} as the energy of the difference between two successive steps, that is:

$$U_{cur} = (\mathbf{x}^{(k+1)})^t\mathbf{K}\mathbf{x}^{(k+1)} \tag{14}$$

$$U_{err} = (\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)})^t\mathbf{K}(\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}), \tag{15}$$

then the inequality in Eq. (12) can be expressed as:

$$U_{err} < \epsilon_U^2 U_{cur}. \tag{16}$$

It must be noted that the quantities involved in the above inequality are inexpensively obtained during each iterative step of the preconditioned conjugate gradient method, requiring only an additional dot product per step. If we define $\mathbf{f}_i^{(k+1)}$ as the vector of internal forces of the current solution $\mathbf{x}^{(k+1)}$ at step k, that is:

$$\mathbf{f}_i^{(k+1)} = \mathbf{K}\mathbf{x}^{(k+1)}, \tag{17}$$

```
Given:        $\mathbf{x}^{(0)}, k = 0$
Initialize:   $\mathbf{f}_i^{(0)} = \mathbf{K}\,\mathbf{x}^{(0)}$
              $\mathbf{r}^{(0)} = \mathbf{f} - \mathbf{f}_i^{(0)}$
              $\mathbf{z}^{(0)} = \mathbf{B}^{-1}\,\mathbf{r}^{(0)}$
              $\mathbf{p}^{(0)} = \mathbf{z}^{(0)}$
              $\mathrm{aux2} = (\mathbf{r}^{(0)}, \mathbf{z}^{(0)})$
Iterate over: $\mathbf{v}^{(k)} = \mathbf{K}\,\mathbf{p}^{(k)}$
              $\mathrm{aux1} = (\mathbf{p}^{(k)}, \mathbf{v}^{(k)})$
              $\alpha^{(k)} = \frac{\mathrm{aux2}}{\mathrm{aux1}}$
              $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha^{(k)}\mathbf{p}^{(k)}$
              $\mathbf{f}_i^{(k+1)} = \mathbf{f}_i^{(k)} + \alpha^{(k)}\mathbf{v}^{(k)}$
              convergence ckeck:
                $U_{cur} = (\mathbf{x}^{(k+1)}, \mathbf{f}_i^{(k+1)})$
                $U_{err} = (\alpha^{(k)})^2\,\mathrm{aux1}$
                if $(U_{err} < \varepsilon_U^2\, U_{cur})$ stop
              $\mathbf{r}^{(k+1)} = \mathbf{r}^{(k)} - \alpha^{(k)}\,\mathbf{v}^{(k)}$
              $\mathbf{z}^{(k+1)} = \mathbf{B}^{-1}\,\mathbf{r}^{(k+1)}$
              $\mathrm{aux3} = (\mathbf{r}^{(k+1)}, \mathbf{z}^{(k+1)})$
              $\beta^{(k)} = \frac{\mathrm{aux3}}{\mathrm{aux2}}$
              $\mathbf{p}^{(k+1)} = \mathbf{z}^{(k+1)} + \beta^{(k)}\,\mathbf{p}^{(k)}$
              $\mathrm{aux2} = \mathrm{aux3}$
              $k = k + 1$
```

Fig. 1. Preconditioned Conjugate Gradient method with energy monitoring.

and noting that the current solution $\mathbf{x}^{(k+1)}$ can be expressed as:

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha^{(k)}\mathbf{p}^{(k)}, \tag{18}$$

then the energies U_{cur} and U_{err} can be written as:

$$U_{\mathrm{cur}} = (\mathbf{x}^{(k+1)})^{\mathrm{t}}\mathbf{f}_i^{(k+1)} \tag{19}$$

$$U_{\mathrm{err}} = (\alpha^{(k)})^2(\mathbf{p}^{(k)})^{\mathrm{t}}\mathbf{K}\mathbf{p}^{(k)}, \tag{20}$$

where the vector $\mathbf{p}^{(k)}$ is the conjugated direction obtained in step k of the conjugate gradient method. The modified algorithm is shown in Fig. 1.

3. Multigrid Methods

Multigrid methods are fast iterative algorithms employed to obtain the solution of discrete boundary value problems. In this paper, we are concerned with the finite element solution of structural mechanics problems, represented by the linear matrix Eq. (1). This basic equation appears in the majority of engineering analyses, whether they be static, dynamic, linear or non-linear.

Consider an approximate solution $\bar{\mathbf{x}}$ to the exact solution $\mathbf{x}$ of Eq. (1), there are two important measures of $\bar{\mathbf{x}}$ as an approximation to $\mathbf{x}$. One is the *error* and is given by

$$\Delta\mathbf{x} = \mathbf{x} - \bar{\mathbf{x}}. \tag{21}$$

Another computable measure of how well $\bar{\mathbf{x}}$ approximates the exact solution $\mathbf{x}$ is the *residual*, given by

$$\mathbf{r} = \mathbf{f} - \mathbf{K}\bar{\mathbf{x}}. \tag{22}$$

If we rearrange its definition as

$$\mathbf{K}\bar{\mathbf{x}} = \mathbf{f} - \mathbf{r} \tag{23}$$

and then subtract this expression from Eq. (1), we find an important relation between the error and the residual:

$$\mathbf{K}\Delta\mathbf{x} = \mathbf{r}, \tag{24}$$

which is called the *residual equation*. This equation plays a vital role in multigrid methods, and it says that the error $\Delta\mathbf{x}$ satisfies the same set of equations as the unknown $\mathbf{x}$ when $\mathbf{f}$ is replaced by the residual $\mathbf{r}$.

Just as we can expand fairly arbitrary functions using a set of eigenfunctions, it is possible to expand

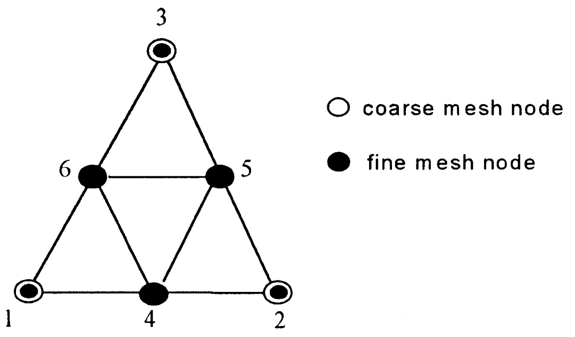

Fig. 2. Coarsening for the linear triangle element.

arbitrary vectors in terms of the eigenvectors of the matrix **K**. In particular, consider the expansion of the error vector, it can be proved [6] that many standard iterative methods, such as Jacobi or Gauss–Seidel methods, are very effective at eliminating the high-frequency or oscillatory components of the error, while leaving the low-frequency or smooth components relatively unchanged. One way to improve a relaxation scheme, at least in its early stages, is to use a good initial guess. A well-known technique for obtaining an improved initial guess is to perform some preliminary iterations on a coarse grid and then use the resulting approximation as an initial guess on the original fine grid. Relaxation on a coarser grid is less expensive, since there are fewer unknowns to be updated.

Recalling that most basic relaxation schemes suffer in the presence of smooth components of the error, we now ask what these smooth components look like on a coarser grid. And the answer is that a smooth mode of the fine grid looks more oscillatory on the coarse grid. Noting that the *k*th mode of both grids are approximations to the *k*th mode of the continuous model, it is evident that for the coarse grid this mode is closer to the high-frequency end of its representable spectrum than for the fine grid. This suggests that when relaxation begins to stall on the fine mesh, signaling the predominance of smooth error modes, it is advisable to move to a coarser grid, on which those smooth error modes appear more oscillatory and relaxation will be more effective. Then we can improve the approximation $\bar{\mathbf{r}}$ by solving the residual equation on a coarser mesh. Denoting quantities on the coarse mesh by the subscript "c" we have:

$$\mathbf{K}_c \Delta \mathbf{x}_c = \mathbf{r}_c, \tag{25}$$

where $\mathbf{K}_c$ is the stiffness matrix of the coarse mesh and $\mathbf{r}_c$ is the residual of the fine mesh approximation transferred to the coarse grid by an operation called *restriction*. The restriction operation depends on the geometrical relations between the coarse and fine meshes, and the type of finite element employed. One way to define a coarse mesh is by eliminating alternate nodes from the fine mesh by applying constraints to the fine mesh degrees of freedom. Fig. 2 shows a fine mesh of four triangles which are coarsened to form one triangle in the coarse mesh.

In this case, the constraint equations take the form:

$$\mathbf{x}_f^1 = \mathbf{x}_c^1, \tag{26}$$

$$\mathbf{x}_f^2 = \mathbf{x}_c^2, \tag{27}$$

$$\mathbf{x}_f^3 = \mathbf{x}_c^3, \tag{28}$$

$$\mathbf{x}_f^4 = (\mathbf{x}_c^1 + \mathbf{x}_c^2)/2, \tag{29}$$

$$\mathbf{x}_f^5 = (\mathbf{x}_c^2 + \mathbf{x}_c^3)/2, \tag{30}$$

$$\mathbf{x}_f^6 = (\mathbf{x}_c^3 + \mathbf{x}_c^1)/2, \tag{31}$$

where subscript "f" refers to the fine mesh. These equations define the displacements of the coarse mesh at the locations of the nodes of the fine mesh. It must be noted that the constraints of Eqs. (29)–(31) assumes that nodes 4, 5 and 6 are midside nodes of the parent triangle of the coarse mesh, if this is not the case, the coefficients of these equations must be slightly different. In matrix form these relations can be expressed as:

$$\Delta \mathbf{x}_f = \mathbf{T} \Delta \mathbf{x}_c, \tag{32}$$

where **T** is the interpolation matrix.

To solve Eq. (25) we must know the residual forces on the coarse mesh transferred from the fine mesh. Since forces on the fine mesh must do the same work when they are restricted to the coarse mesh, it can be shown (see Appendix A) that

$$\mathbf{r}_c = \mathbf{T}^T \mathbf{r}_f, \tag{33}$$

where $\mathbf{T}^T$ is the *restriction* operator and is simply the transpose of the *interpolation* operator **T**. The coarse mesh stiffness matrix $\mathbf{K}_c$ can then be computed as

$$\mathbf{K}_c = \mathbf{T}^T \mathbf{K}_f \mathbf{T}, \tag{34}$$

and solving Eq. (25) we obtain the coarse mesh correction $\Delta \mathbf{x}_c$. Then it is possible to obtain an improved solution on the fine mesh as:

$$\hat{\mathbf{x}}_f = \bar{\mathbf{x}}_f + \Delta \mathbf{x}_f, \tag{35}$$

where $\Delta \mathbf{x}_f$ is the fine mesh correction obtained from the coarse grid correction $\Delta \mathbf{x}_c$ by the *interpolation* operation of Eq. (32).

The above ideas constitute the basis for the multigrid method, where cycles of relaxation on the fine and coarse meshes are combined with intergrid transfers to reduce the high- and low-frequency components of the error of an approximate solution.

3.1. Basic multigrid algorithm

To illustrate the method, consider a two-mesh arrangement. Our objective is to solve Eq. (1) on a fine grid:

$$\mathbf{K}_f \mathbf{x}_f = \mathbf{f}_f. \tag{36}$$

The *k*th cycle of the two level multigrid algorithm consists of the following steps:

- Starting with the current approximation $\mathbf{x}_f^{(k)}$, perform v_1 cycles of an appropriate relaxation method on the fine mesh to obtain a new approximate solution $\mathbf{x}_f^{(k)}$.

- Compute the residual $\mathbf{r}_f^{(k)}$ associated with $\mathbf{x}_f^{(k)}$ on the fine mesh:

$$\bar{\mathbf{r}}_f^{(k)} = \mathbf{f} - \mathbf{K}_f\bar{\mathbf{x}}_f^{(k)}. \tag{37}$$

- Restrict the fine mesh residual onto the coarse mesh using the fine-to-coarse mesh restriction operator $\mathbf{T}^T$ to obtain the coarse mesh residual:

$$\bar{\mathbf{r}}_c^{(k)} = \mathbf{T}^T\bar{\mathbf{r}}_f^{(k)}. \tag{38}$$

- Solve the coarse mesh residual equation to produce the coarse mesh correction:

$$\mathbf{K}_c\Delta\bar{\mathbf{x}}_c^{(k)} = \bar{\mathbf{r}}_c^{(k)}. \tag{39}$$

  If the number of unknowns is sufficiently small, we can use a direct solver in this step.
- Interpolate the coarse mesh correction to the fine mesh using the coarse-to-fine mesh interpolation operator $\mathbf{T}^T$ to obtain the fine mesh correction:

$$\Delta\bar{\mathbf{x}}_f^{(k)} = \mathbf{T}^T\Delta\bar{\mathbf{x}}_c^{(k)}. \tag{40}$$

- Compute the new approximate solution on the fine mesh:

$$\hat{\mathbf{x}}_f^{(k)} = \bar{\mathbf{x}}_f^{(k)} + \Delta\bar{\mathbf{x}}_f^{(k)}. \tag{41}$$

- Finally, perform ν_2 cycles of relaxation starting with $\hat{\mathbf{x}}_f^{(k)}$ to produce the new fine mesh solution $\mathbf{x}_f^{(k+1)}$.

This cycle is repeated m times until a convergence criterion is satisfied on the fine mesh, such as the residual criteria, that is:

$$\frac{\|\bar{\mathbf{r}}_f^{(m)}\|}{\|\mathbf{f}_f\|} \leq \epsilon, \tag{42}$$

where $\|\cdot\|$ denotes the Euclidean norm of a vector, and ϵ is a user-supplied tolerance. In the present implementation we use another convergence criteria based on the energy norm of the increment shown in Section 2.1, that is the multigrid cycles are repeated until the next inequality will be satisfied in the fine mesh:

$$\|\mathbf{x}_f^{(k+1)} - \mathbf{x}_f^{(k)}\|_K < \epsilon_U\|\mathbf{x}_f^{(k+1)}\|_K. \tag{43}$$

The satisfaction of this inequality is verified in each iteration on the fine mesh, and the multigrid cycle is stopped at the iteration where the inequality was satisfied.

The basic two-grid method can be extended to produce a true multigrid method by introducing still coarser meshes, and recursively using one or more cycles of the two grid method to obtain approximate solutions to the coarse mesh correction in Eq. (25). Fig. 3 shows a graphical representation of the multigrid method.

4. Multilevel Preconditioners

The main objective of a preconditioner is to improve the rate of convergence of a basic relaxation algorithm. Some of the most efficient preconditioners are the multi-level preconditioners, which give a proportional relation between the computational effort of the relaxations and the number of equations of the system to be solved [14]. To use this type of preconditioner it is necessary to generate a sequence of refinements (uniform or adaptive) of an initial coarse mesh, and if instead of the standard shape functions used to generate the finite element matrices we use a hierarchical base, then we can retain the information of the previous meshes, as is shown in the next section.

4.1. Hierarchical shape functions

In its usual definition, finite element shape functions are associated with nodal values of the approximation. This identification has the merit of assigning a "physical" meaning to the amplitudes of the shape functions. There is, however, a disadvantage in this "standard" definition of the finite element shape functions. If we considered the succession of meshes generated by a uniform refinement, each level of approximation results in an entire re-evaluation of the equation set, that is:

$$\mathbf{K}_{00}^{(0)}\mathbf{x}_0 = \mathbf{f}_0^{(0)}, \tag{44}$$

$$\begin{bmatrix} \mathbf{K}_{00}^{(1)} & \mathbf{K}_{01}^{(1)} \\ \mathbf{K}_{10}^{(1)} & \mathbf{K}_{11}^{(1)} \end{bmatrix} \begin{Bmatrix} \mathbf{x}_0 \\ \mathbf{x}_1 \end{Bmatrix} = \begin{Bmatrix} \mathbf{f}_0^{(1)} \\ \mathbf{f}_1^{(1)} \end{Bmatrix}, \tag{45}$$

$$\begin{bmatrix} \mathbf{K}_{00}^{(2)} & \mathbf{K}_{01}^{(2)} & \mathbf{K}_{02}^{(2)} \\ \mathbf{K}_{10}^{(2)} & \mathbf{K}_{11}^{(2)} & \mathbf{K}_{12}^{(2)} \\ \mathbf{K}_{20}^{(2)} & \mathbf{K}_{21}^{(2)} & \mathbf{K}_{22}^{(2)} \end{bmatrix} \begin{Bmatrix} \mathbf{x}_0 \\ \mathbf{x}_1 \\ \mathbf{x}_2 \end{Bmatrix} = \begin{Bmatrix} \mathbf{f}_0^{(2)} \\ \mathbf{f}_1^{(2)} \\ \mathbf{f}_2^{(2)} \end{Bmatrix}, \tag{46}$$

where the index between parenthesis indicates the level where each matrix or vector were formed.

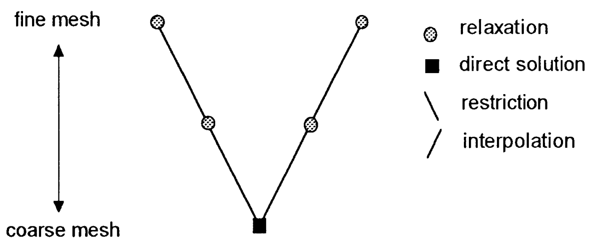

Fig. 3. Graphical representation of the Multigrid Method.

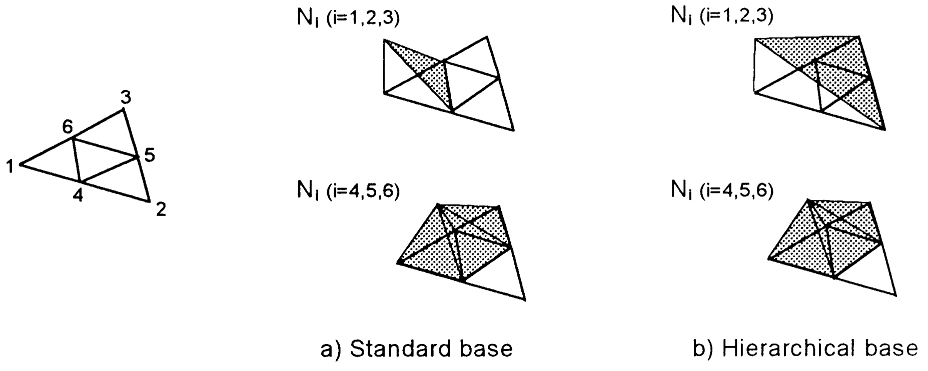

Fig. 4. Standard and hierarchical shape functions on a subdivided triangle.

However, it is possible to use "hierarchical" shape functions whose amplitudes are not directly associated with nodal values. An important aspect of the hierarchical basis is that when a level is refined, the new discretization preserves information from previous discretizations. The use of hierarchical basis results in the following sequence of approximations:

$$\hat{\mathbf{K}}_{00}^{(0)}\hat{\mathbf{x}}_0 = \hat{\mathbf{f}}_0^{(0)}, \tag{47}$$

$$\left[\begin{array}{c|c} \hat{\mathbf{K}}_{00}^{(0)} & \hat{\mathbf{K}}_{01}^{(1)} \\ \hline \hat{\mathbf{K}}_{10}^{(1)} & \hat{\mathbf{K}}_{11}^{(1)} \end{array}\right] \left\{\begin{array}{c} \hat{\mathbf{x}}_0 \\ \hat{\mathbf{x}}_1 \end{array}\right\} = \left\{\begin{array}{c} \hat{\mathbf{f}}_0^{(0)} \\ \hline \hat{\mathbf{f}}_1^{(1)} \end{array}\right\}, \tag{48}$$

$$\left[\begin{array}{cc|c} \hat{\mathbf{K}}_{00}^{(0)} & \hat{\mathbf{K}}_{01}^{(1)} & \hat{\mathbf{K}}_{02}^{(2)} \\ \hat{\mathbf{K}}_{10}^{(1)} & \hat{\mathbf{K}}_{11}^{(1)} & \hat{\mathbf{K}}_{12}^{(2)} \\ \hline \hat{\mathbf{K}}_{20}^{(2)} & \hat{\mathbf{K}}_{21}^{(2)} & \hat{\mathbf{K}}_{22}^{(2)} \end{array}\right] \left\{\begin{array}{c} \hat{\mathbf{x}}_0 \\ \hat{\mathbf{x}}_1 \\ \hat{\mathbf{x}}_2 \end{array}\right\} = \left\{\begin{array}{c} \hat{\mathbf{f}}_0^{(0)} \\ \hat{\mathbf{f}}_1^{(1)} \\ \hat{\mathbf{f}}_2^{(2)} \end{array}\right\}. \tag{49}$$

In each level above it can be seen that, as the approximation is refined, the matrices and vectors generated in the previous levels reoccur and need not be recomputed.

Fig. 4 shows a standard nodal base and a hierarchical base for a regular subdivided triangular element.

In Fig. 5 we can see the variation along a side of a function ϕ approximated hierarchically over a subdivided element.

The value ϕ_4 of the function ϕ at the midside node is composed of two values: the hierarchical part $\hat{\phi}_4$

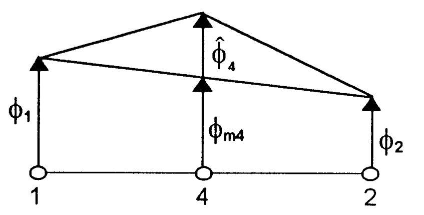

Fig. 5. Variation along a side of a linear hierarchical approximation.

and the linear approximation ϕ_{m4} at the midside node, which is the average of the total values at the end nodes ϕ_1 and ϕ_2, that is

$$\phi_{m4} = (\phi_1 + \phi_2)/2, \tag{50}$$

$$\phi_4 = \hat{\phi}_4 + \phi_{m4}. \tag{51}$$

These relations between variables of the standard and hierarchical basis can be expressed in matrix form as:

$$\mathbf{x} = \mathbf{S}\hat{\mathbf{x}}. \tag{52}$$

The non-singular transformation matrix $\mathbf{S}$ can be partitioned as:

$$\mathbf{S} = \begin{bmatrix} \mathbf{I}_0 & \mathbf{0} \\ \mathbf{W} & \mathbf{I}_1 \end{bmatrix}, \tag{53}$$

where $\mathbf{I}_0$ and $\mathbf{I}_1$ are identity matrices which correspond to the initial and superior levels, respectively, and the submatrix $\mathbf{W}$ contains weights which represent relations of the type of Eq. (50) for each degree of freedom of the finer meshes, that is

$$\mathbf{S} = \mathbf{S}_n\mathbf{S}_{n-1}\mathbf{S}_{n-2}\ldots\mathbf{S}_1, \tag{54}$$

where the indices 1 to n represent the degrees of freedom of the variables of the finer meshes, ordered sequentially from the first refined mesh to the last. Then, to obtain the nodal base representation of a displacement vector expressed in hierarchical coordinates, we must apply successive transformations $\mathbf{S}_i$ from the first refined level to the last:

$$\mathbf{x} = \mathbf{S}\hat{\mathbf{x}} = \mathbf{S}_n\mathbf{S}_{n-1}\mathbf{S}_{n-2}\ldots\mathbf{S}_1\hat{\mathbf{x}}. \tag{55}$$

The transpose matrix $\mathbf{S}^{\mathrm{T}}$ is

$$\mathbf{S}^{\mathrm{T}} = \mathbf{S}_1^{\mathrm{T}}\mathbf{S}_2^{\mathrm{T}}\mathbf{S}_3^{\mathrm{T}}\ldots\mathbf{S}_n^{\mathrm{T}}; \tag{56}$$

then, to obtain the hierarchical representation $\hat{\mathbf{f}}$ of a force vector $\mathbf{f}$ expressed in terms of nodal standard variables, we must apply successive transformations $\mathbf{S}_i^{\mathrm{T}}$

from the last refined level to the first:

$$\hat{\mathbf{f}} = \mathbf{S}^T\mathbf{f} = \mathbf{S}_1^T\mathbf{S}_2^T\mathbf{S}_3^T \ldots \mathbf{S}_n^T\mathbf{f}. \tag{57}$$

Then, to obtain the solution of Eq. (1) we pre-multiply this equation by $\mathbf{S}^T$ obtaining

$$\mathbf{S}^T\mathbf{K}\,\mathbf{x} = \mathbf{S}^T\,\mathbf{K}\,\mathbf{S}\,\hat{\mathbf{x}} = \mathbf{S}^T\,\mathbf{f} \tag{58}$$

which can be expressed as

$$\hat{\mathbf{K}}\,\hat{\mathbf{x}} = \hat{\mathbf{f}} \tag{59}$$

Where the matrix $\hat{\mathbf{K}}$ has the form showed by the system matrix of Eq. (39), that is

$$\hat{\mathbf{K}} = \begin{bmatrix} \hat{\mathbf{K}}_{00}^{(0)} & \hat{\mathbf{K}}_{01}^{(1)} & \hat{\mathbf{K}}_{02}^{(2)} \\ \hat{\mathbf{K}}_{10}^{(1)} & \hat{\mathbf{K}}_{11}^{(1)} & \hat{\mathbf{K}}_{12}^{(2)} \\ \hat{\mathbf{K}}_{20}^{(2)} & \hat{\mathbf{K}}_{21}^{(2)} & \hat{\mathbf{K}}_{22}^{(2)} \end{bmatrix} \tag{60}$$

and due to the couplings of the hierarchical basis $\hat{\mathbf{K}}$ is in general a much more dense matrix than the original matrix $\mathbf{K}$. It must be noted that since the variables of the lower level are not altered we have

$$\hat{\mathbf{K}}_{00}^{(0)} = \mathbf{K}_{00}^{(0)} \tag{61}$$

An important advantage of the hierarchical basis is that the matrices obtained at each stage of refinement are better conditioned than that obtained with the standard nodal basis [7].

4.2. A hierarchical multilevel preconditioner

We can define a hierarchical preconditioner matrix $\hat{\mathbf{B}}$ as

$$\hat{\mathbf{B}} = \begin{bmatrix} (\mathbf{LDL}^\mathbf{T})_{00}^{(0)} & 0 & 0 \\ 0 & \hat{\mathbf{D}}_{11}^{(1)} & 0 \\ 0 & 0 & \hat{\mathbf{D}}_{22}^{(2)} \end{bmatrix}, \tag{62}$$

where $(\mathbf{LDL}^\mathrm{T})_{00}^{(0)}$ represents the Crout factorization of the matrix $\mathbf{K}_{00}^{(0)}$, and $\hat{\mathbf{D}}_{11}^{(1)}$ and $\hat{\mathbf{D}}_{22}^{(2)}$ are the diagonal elements of the matrices $\hat{\mathbf{K}}_{11}^{(1)}$, $\hat{\mathbf{K}}_{22}^{(2)}$, respectively.

Then the solution of the auxiliar system $\mathbf{Bz} = \mathbf{r}$ of the preconditioned iterative methods can be made following the next steps:

1. Transformation of vector $\mathbf{r}$ from nodal to hierarchical base:

$$\hat{\mathbf{r}} = \mathbf{S}^T\mathbf{r} = \mathbf{S}^\mathrm{T}\begin{Bmatrix} \mathbf{r}_0 \\ \mathbf{r}_1 \end{Bmatrix}. \tag{63}$$

2. Solve the coarse mesh system:

$$\mathbf{K}_{00}^{(0)}\hat{\mathbf{z}}_0 = (\mathbf{LDL}^\mathbf{T})_{00}^{(0)}\hat{\mathbf{z}}_0 = \hat{\mathbf{r}}_0 \equiv \mathbf{r}_0. \tag{64}$$

3. Solve the diagonal system of the finer meshes:

$$\hat{\mathbf{z}}_1 = (\hat{\mathbf{D}}_{11}^{(1)})^{-1}\hat{\mathbf{r}}_1, \tag{65}$$

$$\hat{z}_2 = (\hat{\mathbf{D}}_{22}^{(2)})^{-1}\hat{\mathbf{r}}_2. \tag{66}$$

4. Transformation of vector $\mathbf{z}$ from hierarchical to nodal base:

$$\mathbf{z} = \mathbf{S}\hat{\mathbf{z}} = \mathbf{S}\begin{Bmatrix} \hat{\mathbf{z}}_0 \\ \hat{\mathbf{z}}_1 \end{Bmatrix}. \tag{67}$$

It must be noted that with this strategy there is no need to compute the matrices in the hierarchical basis, only vector transformations are employed.

5. Numerical Examples

We use the next convention for the different solvers employed

PCG-H	:conjugate gradients with hierarchical preconditioner
PCG-D	:conjugate gradients with diagonal preconditioner
MG-H	:multigrid with hierarchical PCG for relaxation
MG-D	:multigrid with diagonal PCG for relaxation

In both multigrid solvers we perform five iterations in each relaxation step, that is, we made $\nu_1 = 5$, $\nu_2 = 5$, and the convergence check is made in each iteration on the finer mesh. In all solvers we use the energy criteria for convergence with a tolerance $\epsilon_\mathrm{U} = 0.001$ (0.1% in energy norm).

It must be noted that the stiffness matrix $\boldsymbol{K}$ is only used as a linear operator for vector–matrix products. Noting that $\boldsymbol{K}$ is an assembly of element matrices the product $\boldsymbol{Kx}$ can be written as:

$$\mathbf{Kx} = \sum_e \mathrm{K}^e\mathrm{x}^e, \tag{68}$$

where the superscript "e" denotes element quantities, and summation is carried over all the elements of the mesh. There are basically two approaches for the computation of the summation. One is the *element-by-element* method, in this approach the individual element matrices are calculated once and stored. The product $\mathbf{K}^\mathrm{e}\ \mathbf{x}^\mathrm{e}$ is then directly computed. Another approach is

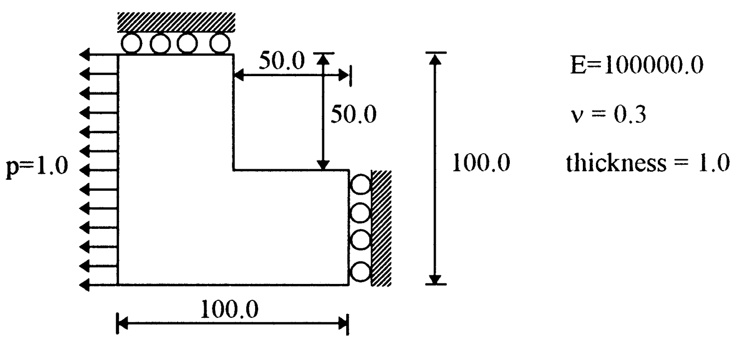

Fig. 6. L-shaped domain problem.

the *matrix-free* method, in this case the individual element matrices are not explicitly formed and stored, instead the product $\mathbf{K}^e\ \mathbf{x}^e$ is computed every time that it is needed.

In this implementation the matrix-free approach was adopted due to the limited memory of the platform employed in the computations, an IBM®-PC compatible (DX4-100 mhz, 8 Mb Ram). It must be noted that numerical experiments of Parsons [17] considering an eight brick node isoparametric element indicates that the matrix-free computations require not only minimum storage but also few operations in the element-by-element method.

In all the examples tested a null vector was used to start the iterative process. It must be noted, that with the exception of the storage for the matrix of the coarse mesh, all the data structures are proportional to the number of elements or nodes of the mesh.

5.1. L-shaped region in plane stress

We consider a standard L-shaped domain problem. Geometry, essential and natural boundary conditions are given in Fig. 6.

In this example, five levels of uniform refinement were used, starting with a mesh of six elements. The initial and final meshes are shown in Fig. 7 together with the number of elements and degrees of freedom (dof).

In Fig. 8, the CPU time spent for each solver is shown. We can see that the diagonal PCG was the slowest of all the algorithms, and the hierarchical PCG was the fastest. Both multigrid algorithms were slower than the hierarchical PCG, and, in particular, for this example the MG-D solver spent almost 44% more time that the PCG-H solver, and the MG-H solver spent almost 24% more time than the PCG-H solver.

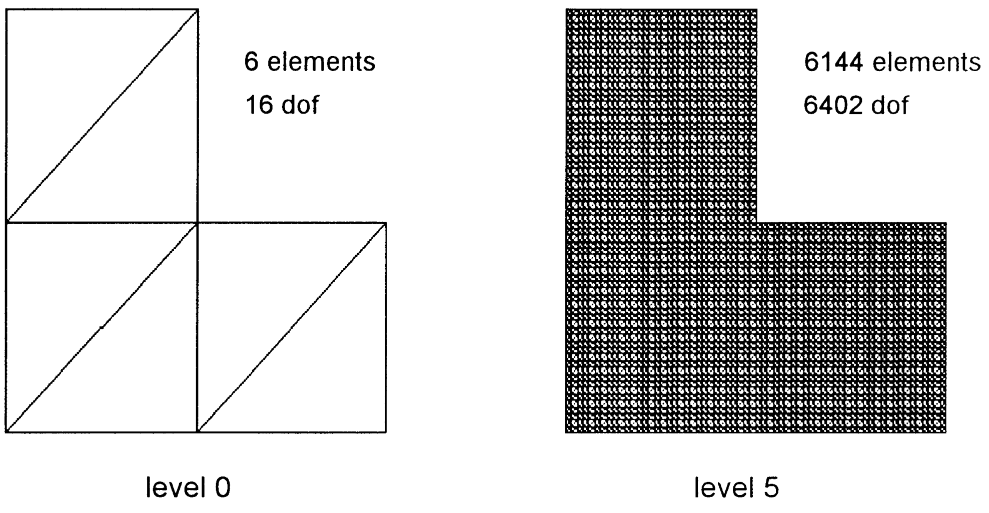

Fig. 7. L-shaped problem initial and final meshes.

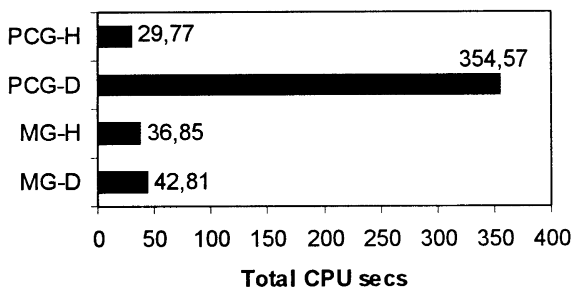

Fig. 8. L-shaped problem CPU time for fine mesh.

In Table 1 the total number of iterations employed by each solver on the fine mesh and the memory requirements are shown.

The same problem was solved for intermediate levels of uniform refinement that is for 384, 1536 and 6144 elements. In Fig. 9 the CPU time spent for each solver is shown, the diagonal PCG was discarded due to the excessive time. We can see that the CPU time spent for all the solvers varies almost linearly with the number of degrees of freedom, as predicted theoretically [2]. Obviously the rate of variation is problem dependent. In Fig. 10 the storage requirements for each solver are shown. We can see that the memory demand varies linearly with the degrees of freedom for all the solvers. Multigrid solvers require more memory since it is necessary to store the residual vectors of the intermediate levels, and MG-D requires more memory than MG-H since it is necessary to store several diagonal preconditioners for each level. MG-H requires only one hierarchical diagonal preconditioner for each level.

For this example, we also made a comparison of the convergence rate of the evolution of the relative energy norm of the increment $\|\mathbf{x}^{(k+1)}-\mathbf{x}^{(k)}\|_K/\|\mathbf{x}^{(k+1)}\|_K$, which was the measure adopted to signalize convergence of the solution, and the error in the energy norm

Table 1
L-shaped problem—total number of iterations and memory requirements on fine mesh

Solver	Iterations on fine mesh	Memory (KBytes)
PCG-H	36	943 (1.00)
PCG-D	502	943 (1.00)
MG-H	28	1070 (1.13)
MG-D	39	1131 (1.20)

of the solution $\|\mathbf{x}^{(k)}-\mathbf{x}_E\|_K/\|\mathbf{x}_E\|_K$. To compute the exact value $\mathbf{x}_E$ we iterate till we obtain a stabilized solution in the energy norm. For this example, the value computed for the energy norm of the solution on the finer mesh was $\|\mathbf{x}_E\|_K = 0.556651817505478$.

We can see in Fig. 11 that both measures are very similar and the variation of the relative energy norm of the increment is almost monotonic with the number of iterations. The comparisons were made using the hierarchical PCG.

5.2. Square plate with a circular hole in plane stress

We consider a plate with a circular hole subjected to uniform tractions in all boundary sides. Owing to symmetry conditions only one quarter of the plate was analysed. Geometry, essential and natural boundary conditions are given in Fig. 12.

In this example, three levels of uniform refinement were used, starting with a mesh of 42 elements. The initial and final meshes are shown in Figs 13 and 14, respectively, together with the number of elements and degrees of freedom (dof).

In Fig. 15 the CPU time spent for each solver is shown. We can see that the diagonal PCG was the slowest of all the algorithms, and the hierarchical PCG

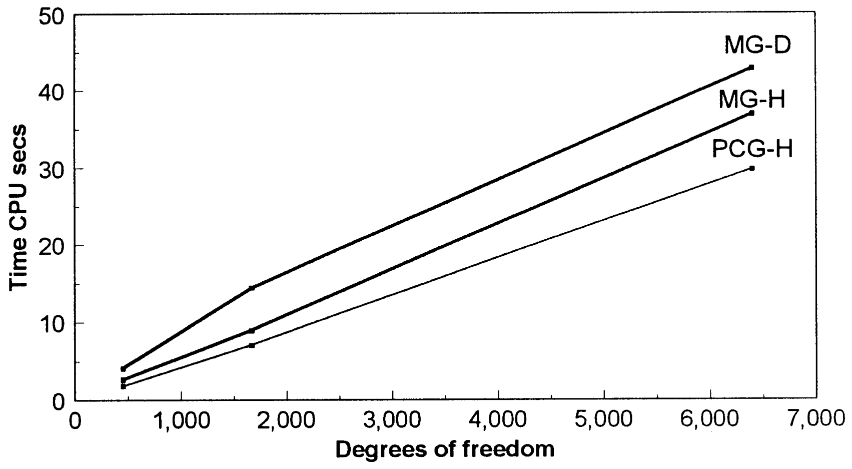

Fig. 9. L-shaped problem—CPU time for different discretizations.

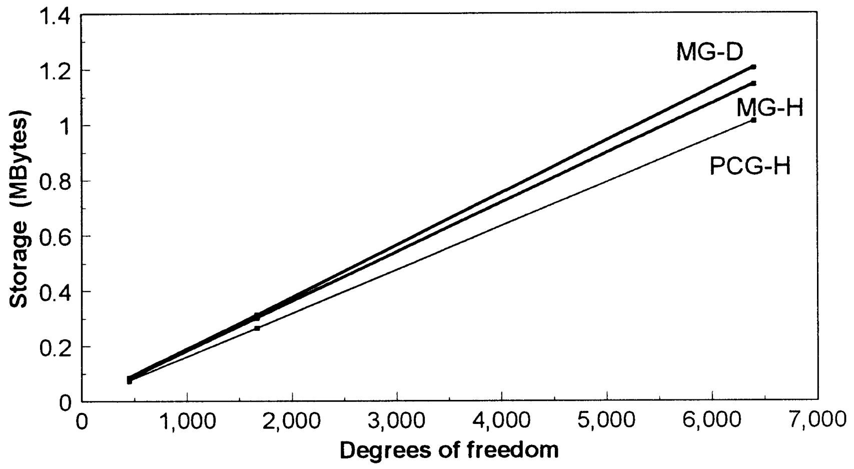

Fig. 10. L-shaped problem—storage for different discretizations.

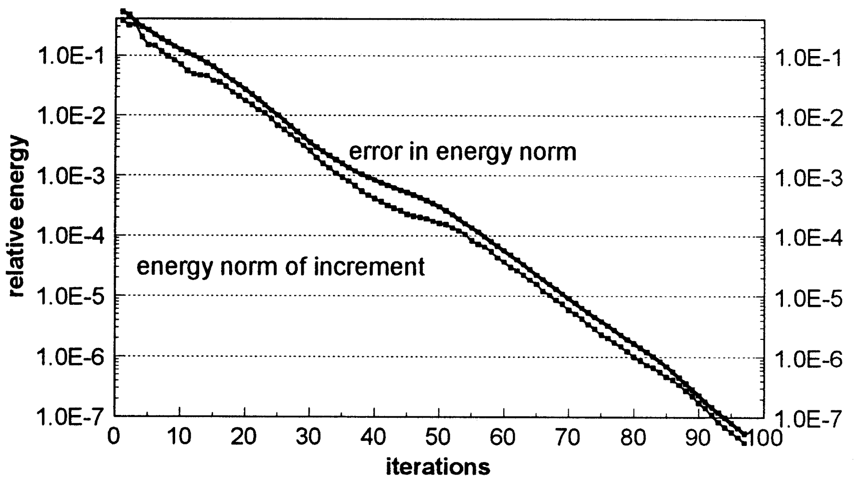

Fig. 11. Comparison between error in energy norm and relative energy norm of increment.

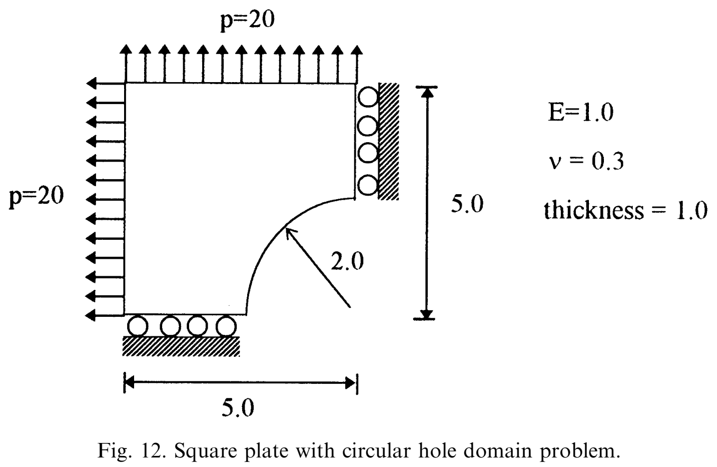

Fig. 12. Square plate with circular hole domain problem.

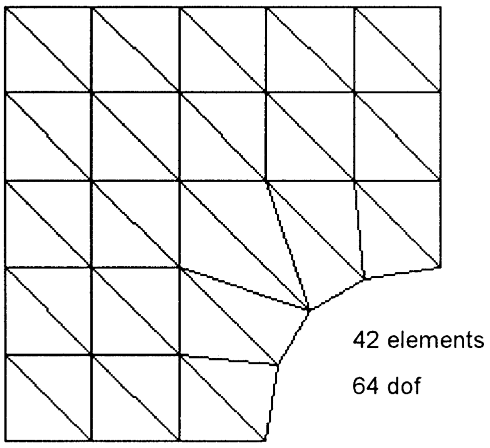

Fig. 13. Square plate with circular hole initial mesh.

was the fastest. Both multigrid algorithms were slower than the hierarchical PCG. In particular for this example, both multigrid solvers spent almost 17% more time than the PCG-H solver.

For this example, we made a comparison using the two convergence criteria explained in Section 2.4. We adopt an error tolerance $\epsilon = 1e\text{-}6$ for the residual norm and an error tolerance $\epsilon_U = 0.001$ (0.1%) for the energy norm. In Fig. 16 the CPU time spent for each solver and for each convergence criteria is shown. Also, in Table 2 we can see the total number of iterations employed by each solver on the fine mesh for the two convergence criteria and the memory requirements for the fine mesh using the energy criteria.

For this example, we also made a comparison of the convergence rate of the relative norm of the residual $\|\mathbf{r}^{(k)}\|/\|\mathbf{f}\|$ and the error in the energy norm of the solution $\|\mathbf{x}^{(k)} - \mathbf{x}_E\|_K / \|\mathbf{x}_E\|_K$ (Fig. 17). To compute the exact value $\mathbf{x}_E$ we iterate till we obtain a stabilized solution

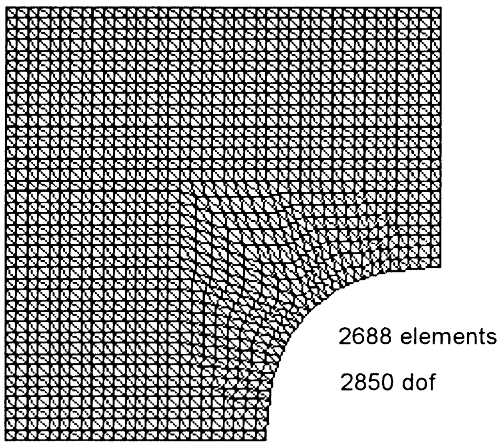

Fig. 14. Square plate with circular hole final mesh.

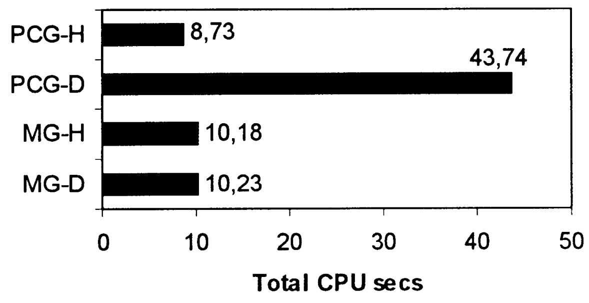

Fig. 15. Square plate with a hole problem CPU time for fine mesh.

in the energy norm. For this example, the value computed for the energy norm of the solution on the finer mesh was $\|\mathbf{x}_E\|_K = 142.582150746839$.

Comparing values of both measures for the same number of iterations, we can deduce that for the adopted error tolerance $\epsilon = 1e\text{-}6$ in the residual norm, the approximated error of the solution is $1e\text{-}7$ (0.00001%) in energy norm for this example.

As in the previous example, we made a comparison of the convergence rate of the evolution of the relative energy norm of the increment $\|\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}\|_K / \|\mathbf{x}^{(k+1)}\|_K$ and the error in the energy norm of the solution $\|\mathbf{x}^{(k)} - \mathbf{x}_E\|_K / \|\mathbf{x}_E\|_K$.

We can see in Fig. 18 that both measures are almost identical. The comparisons were made using the hierarchical PCG.

5.3. Wrench problem

We consider a wrench subject to uniform tractions concentrated on the extreme. Geometry, essential and natural boundary conditions are given in Fig. 19.

In this example, three levels of uniform refinement were used, starting with a mesh of 39 elements. The initial and final meshes are shown in Figs. 20 and 21, respectively, together with the number of elements and degrees of freedom (dof).

In Fig. 22 the CPU time spent for each solver is shown. We can see that, again, the diagonal PCG was the slowest of all the algorithms, and the hierarchical PCG was the fastest. Both multigrid algorithms were slower than the hierarchical PCG. The MG-D solver spent almost 94% more time than the PCG-H solver, and the MG-H solver spent almost 47% more time than the PCG-H solver. The total number of iterations and the memory requirements on the fine mesh are shown in Table 3.

As in the previous examples, we also made a comparison of the convergence rate of the evolution of the relative energy norm of the increment and the error in the energy norm of the solution. Again, we can see in Fig. 23 that both measures are almost identical.

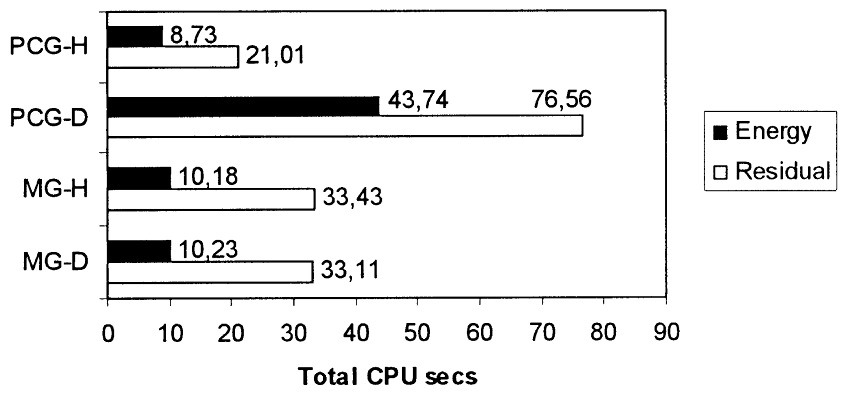

Fig. 16. CPU time for energy and residual convergence criteria.

Table 2
Square plate problem—total number of iterations and memory requirements on fine mesh

Solver	Iterations on fine mesh Energy norm	Residual norm	Memory (KBytes)
PCG-H	22	56	424 (1.01)
PCG-D	143	253	419 (1.00)
MG-H	18	65	484 (1.16)
MG-D	20	67	500 (1.19)

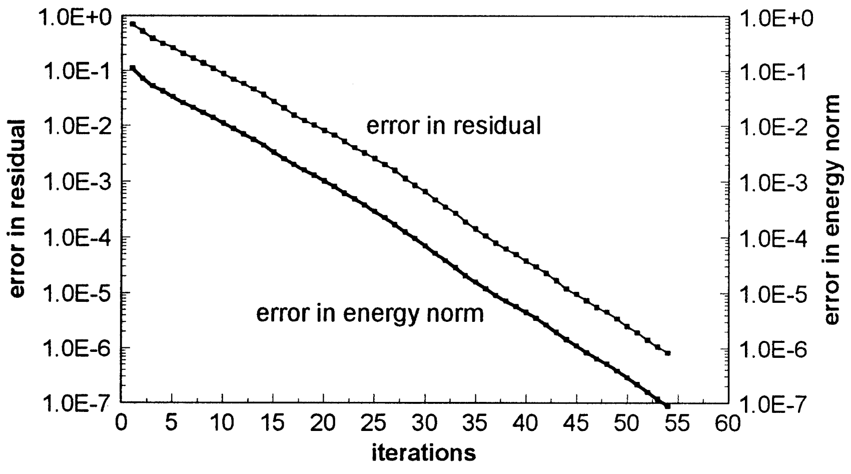

Fig. 17. Comparison between energy and residual convergence.

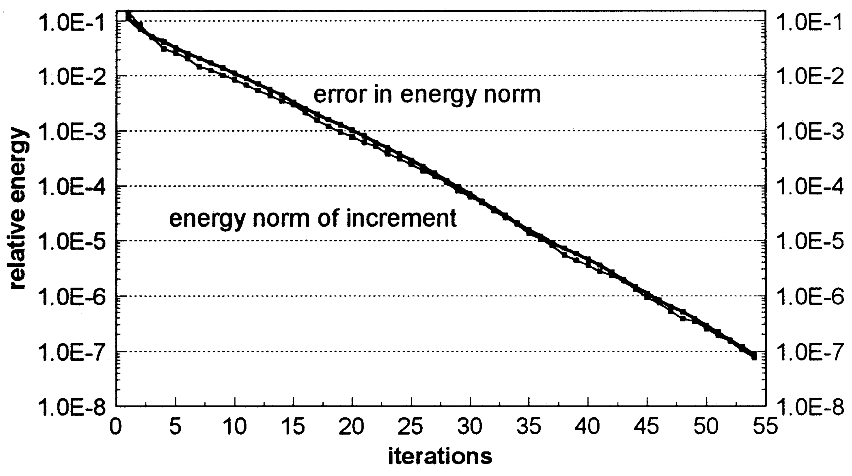

Fig. 18. Comparison between error in energy norm and energy norm of increment.

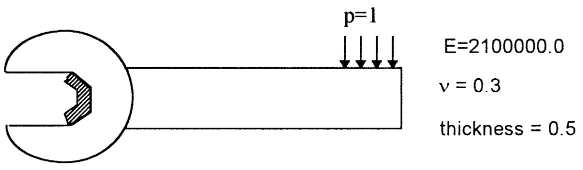

Fig. 19. Wrench domain problem.

6. Conclusions

From the examples tested we can conclude that the hierachical preconditioned conjugate gradients (PCG-H) was the most efficient of the algorithms. The multigrid solvers appear slower than the hierarchical PCG, but faster than the diagonal PCG. The fact that multigrid solvers can be slower than some PCG solvers has also been reported by other authors [8].

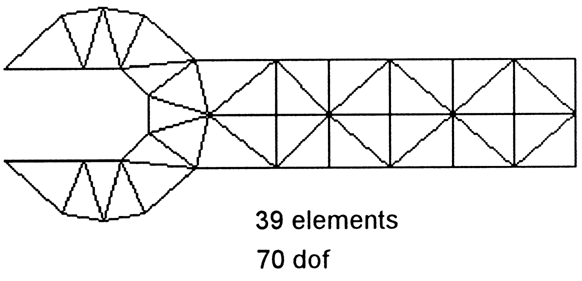

Fig. 20. Wrench problem initial mesh.

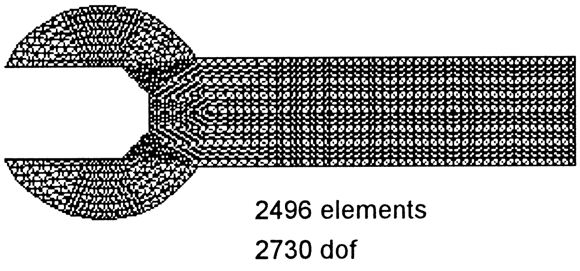

Fig. 21. Wrench problem final mesh.

Table 3
Wrench problem—total number of iterations and memory requirements on fine mesh

Solver	Iterations on fine mesh	Memory (KBytes)
PCG-H	26	407 (1.01)
PCG-D	381	404 (1.00)
MG-H	27	465 (1.15)
MG-D	40	480 (1.19)

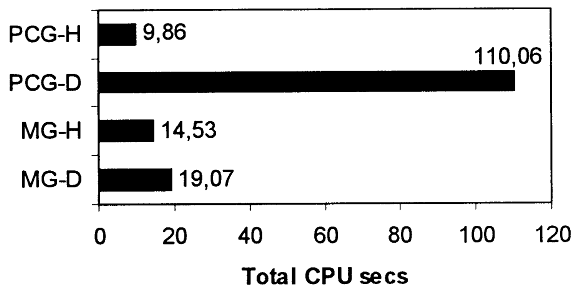

Fig. 22. Wrench problem CPU time for fine mesh.

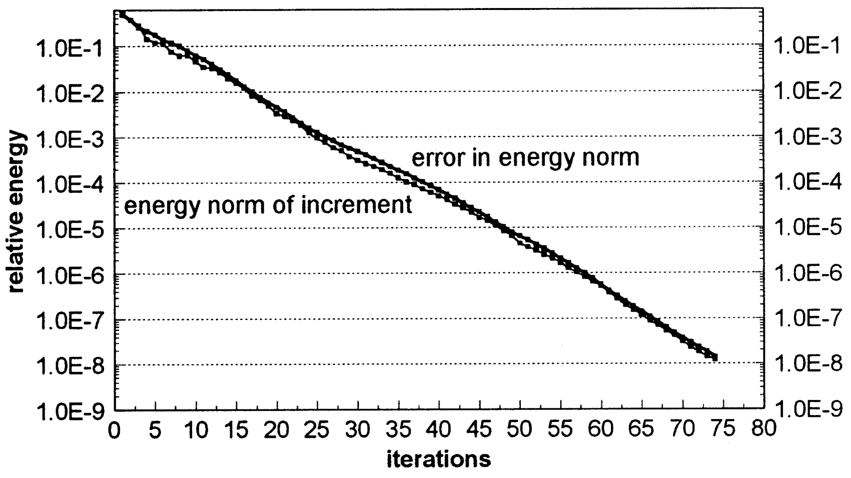

Fig. 23. Comparison between error in energy norm and energy norm of increment.

Comparing Tables 1–3, we can see that the memory requirements of the MG-H solver have an increment of 13–16% over the memory requirements of the PCG solvers, and the memory requirements of the MG-D solver have an increment of 19–20% over the memory requirements of the PCG solvers.

The superiority of the hierarchical PCG over the multigrid solvers in all the examples tested is an important fact, since multigrid solvers need more storage than PCG solvers, and the efficiency of multigrid solvers rely on coefficients which must be determined by numerical experiments, such as the numbers of relaxation steps v_1, v_2 on the intermediate meshes. We have used fixed values for these parameters, although for an optimal implementation a variable number of relaxations would be needed in each grid. The numerical determination of these optimum values was not considered in this paper, since we have compared the usual implementation of each algorithm.

The use of the energy norm of the increment as an indicator for convergence instead of the residual norm, appears to be more convenient. Comparing values of both measures for the same number of iterations we can deduce for this example that acceptable solutions (0.1% error in energy norm) can be obtained with fewer iterations than those needed to reach the numerical convergence in the residual norm, for the usual values of tolerances (1*e*-6). Also, there seems to be a direct relation between the energy norm of the increment and the error in the energy norm, but a deep theoretical analysis is needed to confirm this important fact, which can be particularly useful in the context of adaptive analysis. Another advantage of the energy indicators is that they are non-dimensional.

Finally, it must be noted that the examples tested are insufficient to give decisive conclusions about the absolute efficiency of one algorithm over the others, but indicates a tendency that must be confirmed with larger test problems, not only on uniform refined meshes, but also on adaptively refined meshes, and with more efficient implementations on vector and parallel computational architectures.

Acknowledgements

We are indebted to Prof I. D. Parsons from the University of Illinois at Urbana-Champaign, for the many discussions and sugestions throughout the course of this work.

Appendix A

A.1. Coordinate transformation

Consider a general linear coordinate transformation for displacements $\mathbf{x}$:

$$\mathbf{x} = \mathbf{T}\hat{\mathbf{x}}, \tag{A1}$$

where $\hat{\mathbf{x}}$ is the vector of transformed coordinates which can have less components than vector $\mathbf{x}$ and consequently, the transformation matrix $\mathbf{T}$ is not necessaring square.

The equilibrium equations in these two systems can be written as:

$$\mathbf{K}\mathbf{x} = \mathbf{f}, \tag{A2}$$

$$\hat{\mathbf{K}}\hat{\mathbf{x}} = \hat{\mathbf{f}}, \tag{A3}$$

and we are interested in the relations between quantities of both systems. Since $\mathbf{f}$ and $\hat{\mathbf{f}}$ describe the same resultant force, work done by the force during a prescribed virtual displacement must be independent of the coordinate system in which the work is computed. Let $\delta\mathbf{x}$ and $\delta\hat{\mathbf{x}}$ be the same virtual displacement expressed in both systems, and which components are related by eqn (A1) as:

$$\delta\mathbf{x} = \mathbf{T}\delta\hat{\mathbf{x}}, \tag{A4}$$

then from the equality of the virtual work in both systems we have:

$$\delta\mathbf{x}^{\mathrm{T}}\mathbf{f} = (\delta\hat{\mathbf{x}}^{\mathrm{T}}\mathbf{T}^{\mathrm{T}})\mathbf{f} = \delta\hat{\mathbf{x}}^{\mathrm{T}}\hat{\mathbf{f}}, \tag{A5}$$

from which

$$\delta\hat{\mathbf{x}}^{\mathrm{T}}(\mathbf{T}^{\mathrm{T}}\mathbf{f} - \hat{\mathbf{f}}) = 0, \tag{A6}$$

and since this expression must be valid for *any* virtual displacement, we obtain

$$\hat{\mathbf{f}} = \mathbf{T}^{\mathrm{T}}\mathbf{f}. \tag{A7}$$

Thus, by substitution into eqn (A2) we have

$$\hat{\mathbf{K}}\hat{\mathbf{x}} = \hat{\mathbf{f}} = \mathbf{T}^{\mathrm{T}}\mathbf{f} = \mathbf{T}^{\mathrm{T}}\mathbf{K}\mathbf{x} = \mathbf{T}^{\mathrm{T}}\mathbf{K}\mathbf{T}\hat{\mathbf{x}}, \tag{A8}$$

from which

$$\hat{\mathbf{K}} = \mathbf{T}^{\mathrm{T}}\mathbf{K}\mathbf{T}, \tag{A9}$$

which expresses the relation between the stiffness matrices in both systems.

References

[1] Papadrakakis M. Solving large-scale linear problems in solid and structural mechanics. In: Papadrakakis M, editor. Solving large-scale problems in mechanics. Chichester, UK: Wiley, 1993.

[2] Parsons ID, Hall JF. The multigrid method in solid mechanis: part I—algorithms description and behaviour. Int J. Numer. Meth Engng 1990;29:719–38.

[3] Parsons ID, Hall JF. The multigrid method in solid mechanis: part II—practical applications. Int J Numer Meth Engng 1990;29:739–54.

[4] Hestenes MR, Stiefel E. Methods of conjugate gradients for solving linear systems. J Res Nat Bur Stand 1952;49:409–36.

[5] Reid JK. On the method of conjugate gradients for the solution of large sparse systems of linear equations. In: Reid JK, editors. Large sparse sets of linear equations. Academic Press, New York, 1970. p. 231–254.

[6] Briggs WL. A multigrid tutorial. PA: SIAM, Philadelphia, 1987.

[7] Bank RH, Dupont T, Yserentant H. The hierarchical basis multigrid method. Numer Math 1988;52:427–58.

[8] Fish J, Markolefas S, Guttal R, Nayak P. On adaptive multilevel superposition of finite element meshes for linear elastostatics. Appl Numer Math 1994;14:135–64.

[9] Jouglard CE. A multilevel adaptive approach for the solution of non-linear finite element problems. PhD thesis. Department of Civil Engineering, COPPE/Federal University of Rio de Janeiro, 1996.

[10] Hackbush W. Multi-grid methods and applications. Berlin: Springer, 1985.

[11] McCormick SF. Multigrid methods. PA: SIAM, Philadelphia, 1987.

[12] Bank RE, Yserentant H. Some remarks on the hierarchical basis multigrid method. In: Chan TF, Glowinski R, Periaux J, Widlund OB, editors. Proceedings of the Second International Symposium on Domain Decomposition Methods. Philadelphia, PA: SIAM, 1989. 140–146.

[13] Yserentant H. On the multi-level splitting of finite element spaces for indefinite elliptic boundary value problems. SIAM J Num Anal 1986;23:581–95.

[14] Yserentant H. Hierarchical bases give conjugate gradient type methods a multigrid speed of convergence. Appl Math Comput 1986;19:347–58.

[15] Coutinho ALGA, Alves JLD, Ebecken NFF, Devloo PR. Two-level hierarchical preconditioners for finite element equations. In: Parsons ID, Nour-Omid B, editors. ASME Winter Annual Meeting. Atlanta, 1991. 47–55.

[16] Barra LPS, Coutinho ALGA, Telles JCF, Mansur WJ. Multi-level hierarchical preconditioners for boundary element systems. Engng Anal Bound Elem 1993;12:103–9.

[17] Parsons ID. Parallel adaptive multigrid methods for elasticity, plasticity and eigenvalue problems. In: Papadrakakis M, editor. Solving large-scale problems in mechanics—parallel and distributed computer applications. Chichester, UK: Wiley, 1996. vol. II.