

**Non-nested and non-structured multigrid methods applied to elastic problems.  
Part II: The three-dimensional case**

MARCO L. BITTENCOURT (\*)  
CRAIG C. DOUGLAS (\*\*)  
RAÚL A. FEIJÓO (\*\*\*)

\* Center for Computational Sciences  
University of Kentucky  
325 McVey Hall, Lexington, KY, 40506-0045, USA  
e-mail: mlb@ccs.uky.edu

\*\* Department of Mathematics  
University of Kentucky  
715 Patterson Office Tower, Lexington, KY, 40506-0027, USA  
e-mail: douglas@ccs.uky.edu

\*\*\* Laboratório Nacional de Computação Científica (LNCC/CNPq)  
Av. Getulio Vargas 333, CEP 25651-070, Petrópolis/RJ, Brazil  
e-mail: feij@alpha.lncc.br

## SUMMARY

Aspects of non-nested and non-structured multigrid methods with applications to two-dimensional elastic problems were presented in a companion paper [5] by the current authors, *Non-Nested and Non-Structured Multigrid Methods Applied to Elastic Problems, Part I: The Two-Dimensional Case*. In this paper a review of some multigrid strategies, procedures for geometric search to implement transfer operators, expressions for calculating the number of operations and memory space, and aspects of convergence are presented. Three-dimensional elastic problems are solved by multigrid, sparse Gaussian elimination, and conjugate gradient methods. The number of operations and memory requirements are compared.

## 1 Introduction

The main feature of multigrid methods is an asymptotic linear cost of  $\mathcal{O}(N)$  for the solution of  $N^{\text{th}}$ -order systems of equations. For this purpose, a sequence of meshes with different refinement levels is used to obtain the solution on the finest mesh. Based on the smoothing property of relaxation methods and concepts like nested iteration and correction schemes, many multigrid strategies can be defined [3, 5, 4, 9].

Traditionally, multigrid methods have been used with problems discretized with nested meshes. Thus, unknowns on a coarse mesh are also fine mesh unknowns. Difficulties arise when applying multigrid methods to hard engineering problems with complex geometries where nested grids are not appropriate. By using non-nested approximation spaces this difficulty can be overcome. Hence, the solution of hard engineering problems with the optimal order of multigrid methods can be obtained.

The weak (or variational) formulation for the linear elastic problem in a domain  $\Omega$  is given by

$$\int_{\Omega} \mathbf{T}(\mathbf{w}) \cdot \mathbf{E}(\mathbf{v}) \, dV = \int_{\Gamma^2} \mathbf{\Phi} \cdot \mathbf{v} \, dA + \int_{\Omega} \mathbf{f} \cdot \mathbf{v} \, dV, \quad \forall \mathbf{v} \in \mathcal{V}, \quad (1)$$

where  $\mathcal{V}$  is the vector space of admissible displacements,  $\mathbf{T}$  is the Cauchy stress tensor, and  $\mathbf{f}$  is the body force vector field [15]. For a linear elastic problem, the stress tensor is related to the infinitesimal strain tensor  $\mathbf{E}(\mathbf{w}) = \frac{1}{2} (\nabla \mathbf{w} + \nabla \mathbf{w}^T)$  linearly as  $\mathbf{T} = \mathbf{C}[\mathbf{E}] = 2\mu \mathbf{E} + \lambda(\text{tr } \mathbf{E})\mathbf{I}$ , where  $\mathbf{C}$  is the elasticity tensor,  $\mathbf{w}$  is the displacement vector field, and  $\lambda$  and  $\mu$  are the Lamé's coefficients. The essential and natural boundary conditions on  $\partial\Omega$  are defined by  $\mathbf{w} = \mathbf{0}$  for  $\mathbf{x} \in \Gamma^1$  and  $\mathbf{T}\mathbf{n} = \mathbf{\Phi}$  for  $\mathbf{x} \in \Gamma^2$ . Here,  $\partial\Omega = \Gamma^0 \cup \Gamma^1 \cup \Gamma^2$  and  $\Gamma^0 \cap \Gamma^1 \cap \Gamma^2 = \emptyset$ .  $\Gamma^0$  is a part of the boundary  $\partial\Omega$  without any prescribed boundary condition and  $\mathbf{\Phi}$  is the surface force vector field.

The approximated solution of the linear elastic problem requires solving a system of equations  $\mathbf{A}\mathbf{u} = \mathbf{b}$ , where  $\mathbf{A}$  is a symmetric matrix of order  $N$  [1].

Many concepts of multigrid methods like nested iterations, a coarse grid correction scheme, a variational formulation for transfer operators, multigrid strategies, and a C++ implementation were presented in [5]. Results for two-dimensional linear elastic examples discretized with non-structured, non-nested triangle meshes were also presented. This paper is a continuation of [5]. In the current paper we consider three-dimensional problems with non-nested meshes. We briefly review the most common multigrid strategies, geometric search techniques for the implementation of transfer operators, and general convergence aspects. For the examples studied, an average solution cost close to the optimal linear order is obtained. The results are compared with sparse Gaussian elimination and iterative methods based on conjugate gradient techniques [1, 3]. Results for adaptive non-nested multigrid methods applied to elastic problems are described in [4].

## 2 Multigrid Strategies

Figure 1 shows some commonly used multigrid schemes. The V and W cycles are based on the recursive application of the coarse grid correction scheme. For the V cycle, beginning on the finest mesh  $k$ ,  $\nu_1$  pre-relaxations are executed to obtain an initial approximation  $\mathbf{v}$  for the solution  $\mathbf{u}$  of system of equations. The respective residual  $\mathbf{r} = \mathbf{b} - \mathbf{A}\mathbf{v}$  is calculated and projected on the coarse mesh  $k-1$  using the restriction operator  $I_k^{k-1}$ . On the coarser meshes the same procedure is used except that the residual equation  $\mathbf{A}\mathbf{e} = \mathbf{r}$  is solved. On the coarsest mesh, the error equation is solved by a direct or an iterative method. Corrections  $\mathbf{e}$  are mapped to finer levels using the interpolation (prolongation) operator  $I_{k-1}^k$ . The current approximation is updated as  $\mathbf{v}^k \leftarrow \mathbf{v}^k + I_{k-1}^k \mathbf{e}^k$  and then  $\nu_2$  post-relaxations are applied.

The FMV algorithm uses the concept of nested iterations to obtain a better initial approximation for the intermediate V cycle. Each V cycle is preceded by  $\nu_0$  V cycles on coarser grids.  $\mathbf{A}\mathbf{u} = \mathbf{b}$  is solved on the coarsest mesh and its solution is prolonged as an initial approximation to the next level using the prolongation operator  $I_{k-1}^k$ . Then  $\nu_0$  V cycles are applied, ultimately obtaining a better approximation on the next finer mesh. This procedure is repeated until the finest mesh is reached. The FMW technique uses W cycles to obtain a better initial approximation to a next level. It is analogous to the FMV cycle. These procedures are illustrated in the Figure 1 for  $\nu_0 = 1$ .

For nested meshes, the unknowns on a coarse mesh are also on finer meshes. Transfer operators are relatively simple. For non-nested meshes, the transfer operators are not so simple.

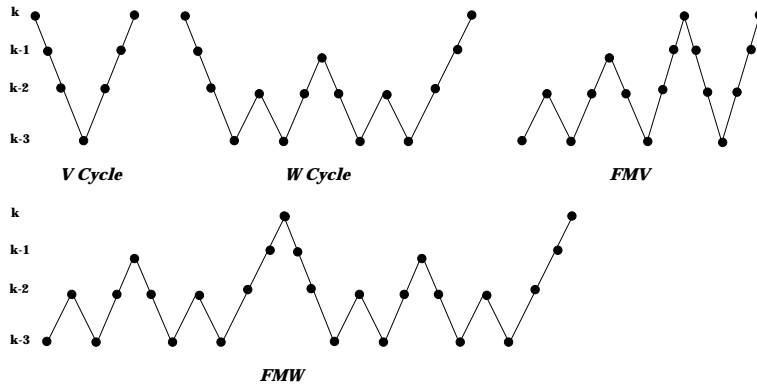


Figure 1: Multigrid strategies.

A variational formulation for non-nested transfer operators is presented in [5]. For each finer node we need to know the coarser element which contains the node and the respective values of the shape functions taken on the local coordinates of the finer node (see Figure 2). Efficient data structures and procedures must be considered for information retrieval and are discussed in Section 3.

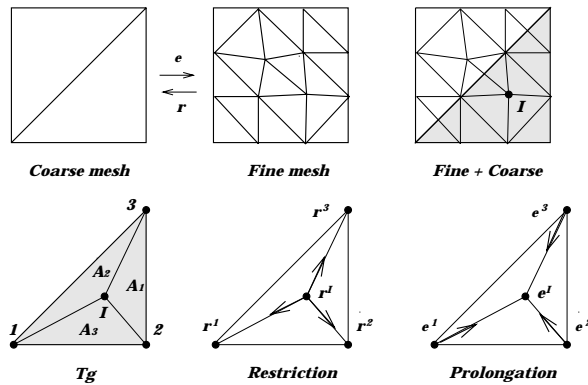


Figure 2: Restriction and prolongation operators for 2D non-nested meshes.

### 3 Information Retrieval

To apply the restriction and prolongation operators on a non-nested mesh the coarser element containing a given fine node must be known. Further, this must be known for all fine nodes. This kind of problem is called a *geometric search*. It is frequently found in areas such as mesh generation and geometric modeling [7, 10]. In general, given  $n$  elements, we want to know which ones contain a certain point. The trivial (or direct) solution consists of sweeping the list of elements and checking if a specified node is inside one of the elements. The computational time of this procedure is  $\mathcal{O}(n)$ . Procedures to decrease this linear cost are discussed in [10].

The basic idea of decreasing the cost of the search problem is to divide a region into a set of cells with simple shapes, making a correspondence of these cells with the elements. Initially, it is necessary to determine the cell containing the point. Then a search in the elements associated with the cell is conducted to obtain the element(s) containing the node.

A one time preliminary ordering phase is required. It is conducted by dividing the region

into cells and obtaining the associated elements. The corresponding cost is at least  $\mathcal{O}(n)$ , which is similar to the direct search procedure. Hence, the application of a search procedure is essential only in cases where the total number of search operations is large. The memory required for the auxiliary data structures to store cells and their elements must be considered in order to balance the CPU cost versus the memory space [10].

Four data structures for information retrieval are now discussed. The structures will be referred to as follows: uniform auxiliary mesh, non-uniform auxiliary mesh, quadtree, and uniform quadtree mesh.

The uniform auxiliary mesh is a rectangle that contains a considered region. It is defined and divided into a set of  $n_x \times n_y$  cells called the auxiliary mesh (see Figure 3). The extreme coordinates  $(X_{min}, Y_{min})$  and  $(X_{max}, Y_{max})$  of the rectangle are obtained from the element vertices. The cell size is  $s = t_m f_m$ , where  $t_m$  is the average element size and  $f_m$  is a parameter than can be modified to improve either the CPU time or the memory usage.

There are  $n_x = (X_{max} - X_{min})/s + 1$  and  $n_y = (Y_{max} - Y_{min})/s + 1$  subdivisions in the  $x$  and  $y$  directions. The cell size in each direction is given by  $d_x = (X_{max} - X_{min})/n_x$  and  $d_y = (Y_{max} - Y_{min})/n_y$ . Thus,  $n_x \times n_y$  cells are allocated. An initially empty element list is associated with each cell. For each element, the rectangle  $(x_{min}, y_{min}) \times (x_{max}, y_{max})$  containing this element is obtained. Cell indices  $(i_{min}, j_{min}) \times (i_{max}, j_{max})$  intersecting the element are calculated as  $i_{min} = (x_{min} - X_{min})/d_x$ ,  $j_{min} = (y_{min} - Y_{min})/d_y$ ,  $i_{max} = (x_{max} - X_{min})/d_x$  and  $j_{max} = (y_{max} - Y_{min})/d_y$ . The element is included in each cell  $C(i, j)$  with indices in the range  $[i_{min}, i_{max}] \times [j_{min}, j_{max}]$ . This procedure is the preliminary ordering phase for this data structure.

Given a point  $(x, y)$ , the cell containing this point is found using the expressions  $i = (x - X_{min})/d_x$  and  $j = (y - Y_{min})/d_y$ . Then the associated element list for this cell is searched to determine the element containing  $(x, y)$ .

The principal disadvantage of this method is the treatment of regions with a non-uniform element size distribution. The memory demand is increased when the cell size is calculated using the smallest element size. However, the number of elements in some cells can be quite large and the performance of the search procedure will deteriorate when the cell size is determined using the average element size.

A compromise is needed in order to create a balance. Consider non-uniform cell sizes, i.e., a non-uniform auxiliary mesh. Irregular divisions along each of the Cartesian axes are taken and the number of associated elements in each cell can be balanced better. Figure 3 shows both uniform and non-uniform auxiliary meshes with the number of elements in each cell. In both cases we have used a tensor product auxiliary mesh. In the non-uniform case, the divisions along each axis coincide with the element vertices based on a minimum cell size. In spite of reducing the minimum number of elements in each cell, the ordering phase and search algorithm costs are greater than that for the uniform auxiliary mesh [10].

For an arbitrary element size distribution, the algorithms associated with the uniform auxiliary mesh technique do not work well. A quadtree data structure can be adapted easily to any set of elements by limiting the maximum number of elements per cell. A quadtree is a sequence of squares superimposed on the considered region. They are recursively divided and adjust themselves according to cell and element sizes in each part of the domain. In Figure 4 the biggest square is the tree root, the squares with no subdivisions are the leaves, and the others are the branches. Procedures for quadtree data structure creation and information retrieval are presented in [10] with an overall cost of  $\mathcal{O}(\log n)$ .

An uniform quadtree mesh method uses the simplicity of the auxiliary uniform mesh and the efficiency of quadtree data structures. Regions with uniform and non-uniform element size

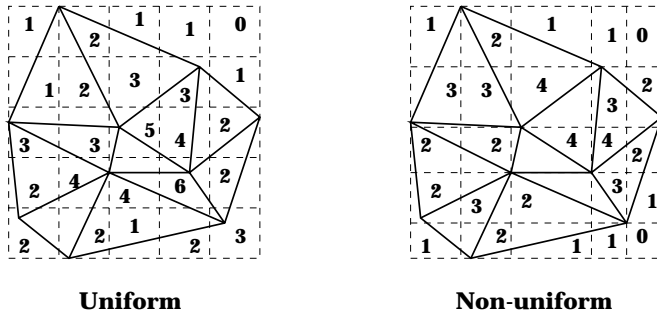


Figure 3: Uniform auxiliary and non-uniform meshes [10].

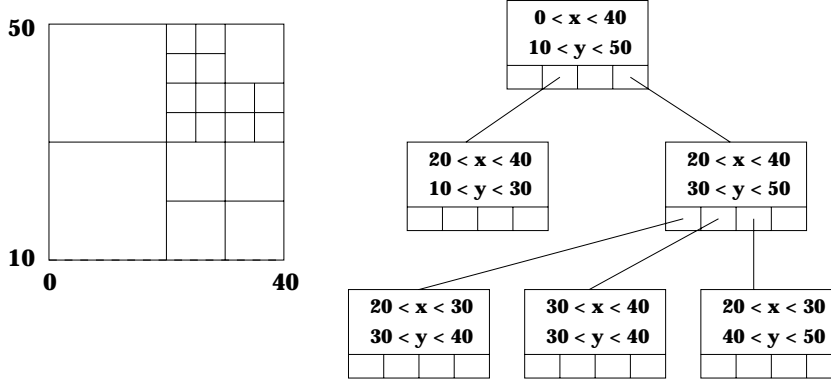


Figure 4: Auxiliary data structure based on quadtree [10].

distributions can be treated easily. An uniform auxiliary mesh is created where its cells are quadtree roots. In the search procedure the cell containing a given point is determined first. Then the quadtree leaf is obtained. Finally, the associated element list is swept to find the element.

The quadtree data structure was used to find the coarser element containing a finer node and its corresponding linear shape function values in [5]. This information is generated before using a multigrid algorithm and is stored in a data structure that is accessed each time the interpolation and restriction operators are used. For three-dimensional problems, an octree data structure is used instead.

## 4 Operation Counting and Memory Requirements

Table 2 presents the items used when calculating the number of operations and the amount of the memory space required for non-nested multigrid methods. They are based on the computational implementation in [3]. Let  $N_m$  be the number of meshes. The notation in Table 1 is used for each mesh  $i$ .

The number of operations is obtained by adding up the expressions in Table 2 for relaxations, direct solution on the coarsest mesh, transfer operators, and calculation of residuals. The equivalent number of finest mesh iterations is obtained by dividing the number of operations by the cost of one finest mesh iteration of the relaxation scheme.

The higher memory requirements for a multigrid method when compared to a traditional iterative method is a combination of the extra meshes and the extra space required for the data

$NIT_i$	Total number of relaxations conducted in all multigrid cycles
$CIT_i$	Cost of one iteration of the relaxation scheme
$N_i$	Number of equations
$m_i$	Average number of non-zero coefficients per row of the system matrix
$n_i$	Total number of times the multigrid technique visited level $i$
$n_i^r$	Total number of restriction operations
$n_i^p$	Total number of prolongation operations
$N_{nodes}$	Number of nodes in the considered element
$n_i^{aux}$	Memory allocation for auxiliary vectors required to implement the relaxation scheme
$N_i^{nodes}$	Total number of nodes
$n_t = 2.5, n_{eq} = 1, n_{inc} = 1.5$	For two-dimensional problems
$n_t = 3.5, n_{eq} = 1.5, n_{inc} = 2$	For three-dimensional problems
$N_i^{els}$	Total number of elements

Table 1: Notation for calculating the number of operations and memory space.

structures used by the restriction and prolongation operators (e.g., incidence and equation numbering). Therefore it is necessary to take into account the systems of equations (matrix+vectors), tables of data for operators, equation numbering, and incidence [3]. The total amount of memory required is the sum of all the expressions indicated in Table 2. Assuming 64 bit words, this sum is multiplied by a factor  $\frac{8}{1024}$  to convert to kilobytes used.

The remainder of this section justifies the contents of Table 2. We use the table to determine how much various multigrid components cost.

A Gauss-Seidel procedure was used as the relaxation scheme for all of the multigrid methods. The cost of one Gauss-Seidel relaxation on mesh  $i$  is given by  $CIT_i = N_i(2m_i - 1)$ . The implementation of the Gauss-Seidel procedure for sparse matrices stored in the row compressed format [1] requires an auxiliary vector, resulting in  $n_i^{aux} = N_i$ .

On the coarsest mesh we use sparse Gaussian elimination. However, it is sometimes less costly to apply an iterative procedure when solving the coarsest mesh problem. For example, using conjugate gradients with a symmetric Gauss-Seidel pre-conditioner [1], the coarsest mesh solution cost in a multigrid strategy is  $NIT_1 \cdot CIT_1 = NIT_1 \cdot [N_i(2m_i + 7)]$ .

Consider Figure 2. There are 3 multiplications for each equation of a fine node I when calculating the restriction or prolongation operators. This results in  $3N_i$  operations for each level  $i$ . For three-dimensional cases discretized with linear tetrahedron the cost is  $4N_i$ .

The cost of the residual calculation on mesh  $i$  is equal to the cost of one iteration of the Gauss-Seidel procedure, i.e.,  $N_i(2m_i - 1)$ . However, the expression given in Table 2 assumes that the residual is calculated together with Gauss-Seidel and shares part of the computation. Hence, a 15% to 20% reduction in the total number of operations for a multigrid solution has been observed compared with previous results obtained in [6].

Now consider the storage costs for the prolongation or restriction processes. As an example, let some fine mesh node lie in some coarse mesh triangle. The data and associated shape functions' values (e.g.,  $A_1 = 0.1$ ,  $A_2 = 0.5$ , and  $A_3 = 0.4$ ) are stored as  $[32, 0.1, 0.5]$ , where 32 is the coarse triangle's number and  $A_3 = 1 - A_1 - A_2$ . Thus, 2 floating point and 1 integer storage locations are required per fine node. For linear tetrahedron 3 floating point and 1 integer storage locations are required per fine node. Information about equation and incidence numbering must also be stored and used each time a prolongation or restriction operator is used.

Operations	
Relaxations	$\sum_{i=2}^{N_m} (\text{NIT}_i) (\text{CIT}_i)$
Coarsest mesh	$\left[ \sum_{i=1}^{N_1} (m_i - 1)(m_i + 2) \right] + 2N_1(m_1 - 1)n_1$
Operators	$N_{nodes} \sum_{i=2}^{N_m} N_i(n_i^r + n_i^p)$
Residual	$\sum_{i=2}^{N_m} N_i(m_i - 1)n_i^r$

Memory space	
Systems	$\sum_{i=1}^{N_m} [N_i(m_i + 2) + n_i^{aux}]$
Operators	$n_t \sum_{i=2}^{N_m} N_i^{nodes}$
Equations	$n_{eq} \sum_{i=1}^{N_m} N_i^{nodes}$
Incidence	$n_{inc} \sum_{i=1}^{N_m} N_i^{els}$

Table 2: Expressions for calculating the number of operations and memory space.

## 5 Convergence of Multigrid Methods

One important aspect of any iterative numerical method for the solution of a linear system of equations is its convergence rate. For linear elliptic problems discretized by finite differences or finite elements with nested uniform meshes, the convergence rate of some multigrid methods is presented in [2, 11, 16, 17, 18]. The solution cost for a nested iteration is linear, i.e.,  $O(N)$ , where  $N$  is the number of equations.

Non-nested meshes are considered in this paper. The main motivation for using non-nested approximation spaces is applying multigrid to the solution of three-dimensional problems discretized by linear tetrahedral elements. For two-dimensional cases it is possible to refine a triangular element into 4 congruent triangles by joining the midpoints of the edges. In general, a tetrahedron element cannot be subdivided into 8 equal elements. In some cases degenerate tetrahedron meshes are obtained in a sequence of successive refinements [21, 22].

Nested approximation spaces are not essential for multigrid methods although they provide many simplifications. Non-nested meshes are more suitable for solving problems with complex geometries and in adaptive mesh procedures. The same optimal order  $O(N)$  can be proved for non-nested spaces. However, the convergence rates are different.

Non-nested convergence aspects for multigrid procedures were presented in [11]. The convergence of a non-standard nested multilevel procedure was proved in [12] by introducing an additional finest level in such a way that the correction scheme is applied for all levels. This same procedure was extended to non-nested meshes in [13].

Some conditions must be imposed on a sequence of non-nested meshes to guarantee convergence of multigrid methods. The relationship between the number of elements in two successive meshes may be arbitrary. However, the meshes must be comparable in the sense that each element in a mesh must be covered by a finite number of elements on both the next coarser and finer levels. Further, the number of unknowns on each level must increase geometrically with a rate greater than 2 in order to possibly have a linear solution time [11, 22].

The convergence of three-dimensional non-nested multigrid methods for second order elliptic boundary value problems is proved for quasi-uniform meshes in [22], for non-quasi uniform meshes in [23], and with arbitrary polynomial interpolation order in [20]. Similar results were obtained in [8] for symmetric strategies in which the number of pre- and pos-relaxations are the same on each level.

## 6 Numerical Experiments

In this section, the behavior in terms of number of operations and memory requirements for sparse Gaussian elimination (SGE), conjugate gradients, and multigrid is analyzed when applied to three-dimensional linear elastic examples discretized by linear tetrahedral meshes. A flop is defined as a single floating point operation.

For sparse Gaussian elimination on sparse matrices, it is useful to apply a symbolic factorization procedure to determine non-zero coefficients in the matrix factors [3, 19]. Thus, we know in advance the total number of factored matrix coefficients to allocate. The minimum degree renumbering procedure was used to renumber the mesh nodes in order to reduce the fill-in [14].

For iterative and multigrid methods, the convergence criterion  $\|\mathbf{A}\mathbf{u} - \mathbf{b}\|_2 / \|\mathbf{b}\|_2 < \xi$  with precision  $\xi = 10^{-4}$  was used. The iterative methods are based on conjugate gradient (CG) with diagonal (CGD), SSOR (CGSS), and symmetric Gauss-Seidel (CGGS) pre-conditioners. Gauss-Seidel was used as the relaxation scheme in all of the multigrid methods.

All of the examples are three-dimensional. The domains are illustrated in Figures 5-8. The body force vector is zero, i.e.,  $\mathbf{f} = \mathbf{0}$  in (1). The definitions of  $\Gamma^1$  and  $\Gamma^2$  are different for each of the examples.

Figure 5 shows the meshes for a cantilever beam rigidly supported on one end and with an applied distributed load along its length. We define  $\Gamma^1$  and  $\Gamma^2$  by

$$\Gamma^1 = \{(x, y, z) \in \Omega \mid u_x = u_y = u_z = 0 \text{ for } x = 0\}$$

and

$$\Gamma^2 = \{(x, y, z) \in \Omega \mid \Phi_y = -1000 \text{ for } y = 10\}.$$

The second example is a piston with four meshes (see Figure 6). On the coarsest mesh the boundary conditions and the applied concentrated load can be observed. We define  $\Gamma^1$  and  $\Gamma^2$  by

$$\Gamma^1 = \{(x, y, z) \in \Omega \mid u_x = 0 \text{ for } x = 0, u_y = 0 \text{ for } y = 0, u_z = 0 \text{ for } x = 0, z = 60\}$$

and

$$\Gamma^2 = \emptyset.$$

A pontual load  $F_z = -100$  was applied at  $x = 0, y = 0, z = 143, 5$ .

The third example is a planar elliptic fracture problem in an infinite domain represented by a parallelepiped. There are three meshes (see Figure 7). We define  $\Gamma^1$  and  $\Gamma^2$  by

$$\Gamma^1 = \{(x, y, z) \in \Omega \mid u_x = 0 \text{ for } x = 0, u_y = 0 \text{ for } y = 0, u_z = 0 \text{ for } \frac{x^2}{4} + y^2 \geq 1, z = 0\}$$

and

$$\Gamma^2 = \{(x, y, z) \in \Omega \mid \Phi_z = 100 \text{ for } z = 15\}.$$

The fourth example is another planar fracture problem in a cylindrical bar, a penny-shaped crack. There are four meshes (see Figure 8). We define  $\Gamma^1$  and  $\Gamma^2$  by

$$\Gamma^1 = \{(x, y, z) \in \Omega \mid u_x = 0 \text{ for } x = 0, u_y = 0 \text{ for } y = 0, u_z = 0 \text{ for } x^2 + y^2 \geq 4, z = 0\}$$

and

$$\Gamma^2 = \{(x, y, z) \in \Omega \mid \Phi_z = 100 \text{ for } z = 32\}.$$

In the last two examples, symmetry conditions were considered. The sequences of meshes were generated by an adaptive procedure based on the Zienkiewicz-Zhu error estimator [24].

The mesh features for all examples are given in Table 3. This includes the numbers of nodes, elements, and equations. For a sparse matrix, the table includes both the total number of coefficients for the entire matrix and the average number of coefficients per row. Where relevant, numbers for direct (NCof<sup>dir</sup>,  $m^{dir}$ ) and iterative/multigrid (NCof<sup>ite</sup>,  $m^{ite}$ ) methods are included.

Table 4 includes the equivalent number of iterations on the finest mesh for any solver used. For multigrid methods, the total number of cycles (NC), the number of pre- and post-relaxations ( $\nu_1, \nu_2$ ), and the number of intermediate cycles ( $\nu_0$ ) are provided. Figures 9 and 10 illustrate the results in terms of the number of operations and the memory space requirements.

For the fracture examples, the solution on the coarsest meshes was obtained by the CGGS method. The high factorization cost on these meshes affects the multigrid performance as illustrated in Figures 9(c) and 9(d). The number iterations on the finest mesh given in Table 4 for the fracture problems is based on using CGGS on the coarsest meshes. The FMW results for the cylinder example, however, were obtained by using sparse Gaussian elimination on the coarsest mesh.

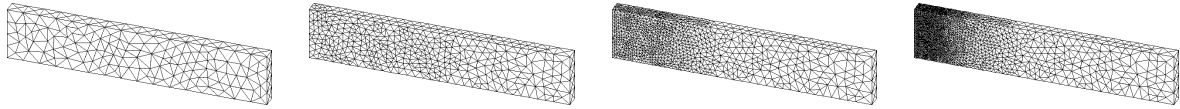


Figure 5: Meshes for the beam example ( $E = 1,0 \times 10^5$ ;  $\nu = 0, 3$ ).

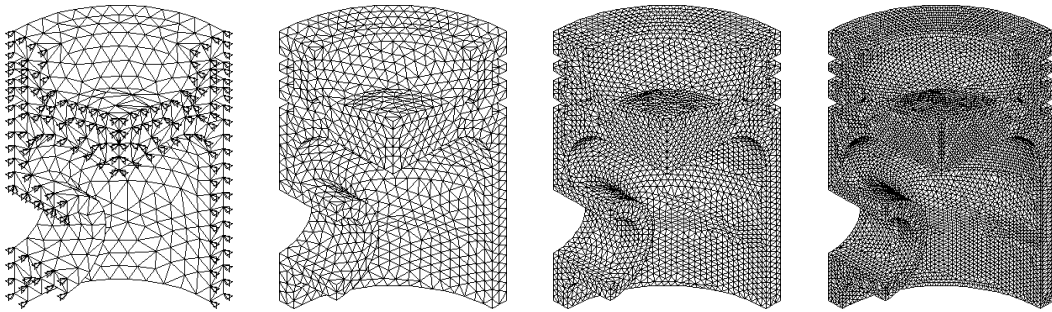


Figure 6: Meshes for the piston example ( $E = 1.0 \times 10^5$ ;  $\nu = 0, 3$ ).

## 7 Conclusions

In general, the iterative methods based on the conjugated gradient technique show superior behavior when compared to sparse Gaussian elimination. For example, the demand for memory increases significantly when using sparse Gaussian elimination (e.g., the finest mesh factorization for the piston example requires over 800 megabytes of memory space). It is necessary to use a model reduction or domain decomposition technique to solve this problem using sparse Gaussian elimination.

Beam							
Mesh	Nodes	Elements	Equations	$m^{dir}$	NCoeff <sup>dir</sup>	$m^{ite}$	NCoeff <sup>ite</sup>
1	335	921	963	40.0	38511	16.0	15444
2	1130	3922	3279	81.2	266226	17.6	57804
3	3024	11813	8724	164.2	1432572	18.6	162276
4	8633	37976	24606	361.9	8904168	19.6	482364
Piston							
Mesh	Nodes	Elements	Equations	$m^{dir}$	NCoeff <sup>dir</sup>	$m^{ite}$	NCoeff <sup>ite</sup>
1	861	2622	2416	75.2	181675	16.4	39654
2	2943	10888	8444	154.5	1304874	18.0	151630
3	12727	55480	37188	373.2	13900617	19.4	722428
4	32348	155792	95277	722.1	68798689	20.5	1948811
Parallelepiped							
Mesh	Nodes	Elements	Equations	$m^{dir}$	NCoeff <sup>dir</sup>	$m^{ite}$	NCoeff <sup>ite</sup>
1	4228	19871	11365	327.0	3716698	19.5	221037
2	9241	44015	25184	422.0	10627486	19.8	495535
3	27652	139348	77862	786.0	61196124	20.5	1596409
Cylinder							
Mesh	Nodes	Elements	Equations	$m^{dir}$	NCoeff <sup>dir</sup>	$m^{ite}$	NCoeff <sup>ite</sup>
1	1810	7599	4630	179.4	830385	18.0	83141
2	7753	36746	20933	428.2	8962786	19.6	409267
3	22670	114634	63778	792.9	50566338	20.5	1309692
4	46982	238546	133471	959.3	128036645	20.6	2746298

Table 3: Mesh attributes.

Method	Beam		Piston		Parallelepiped		Cylinder	
	NIT	NC, $\nu_0, \nu_1, \nu_2$	NIT	NC, $\nu_0, \nu_1, \nu_2$	NIT	NC, $\nu_0, \nu_1, \nu_2$	NIT	NC, $\nu_0, \nu_1, \nu_2$
SGE	4117	-	16934	-	22624	-	40705	
CG	2361	-	1441	-	1513	-	2107	
CGD	1279	-	1250	-	269	-	268	
CGSS	844	-	838	-	180	-	200	
CGGS	750	-	710	-	161	-	157	
FMV	73	9,1,1,1	57	7,1,1,1	125	6,1,1,1	65	4,1,1,1
	82	6,2,1,1	55	4,2,1,1	113	4,1,2,1	60	3,1,2,1
FMW	84	8,1,1,1	56	4,1,2,1	270	5,1,1,1	82	4,1,1,1
	94	5,2,1,1	55	3,2,1,1	177	3,1,2,1	81	3,1,2,1

Table 4: Results for sparse Gaussian elimination, conjugate gradients and multigrid methods.

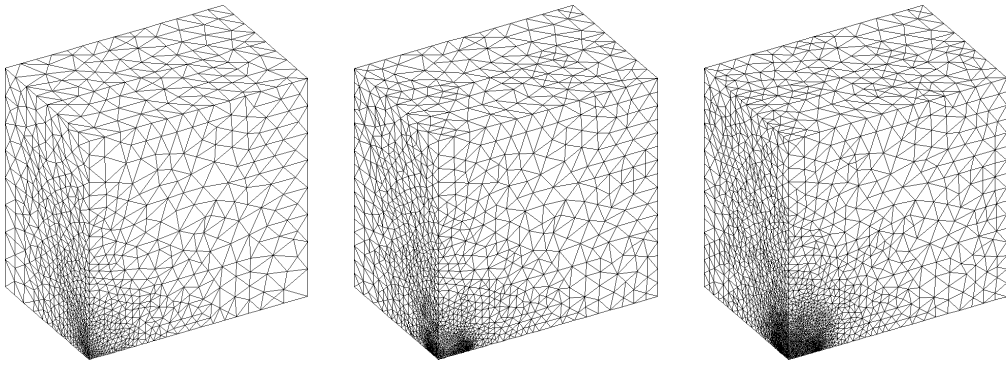


Figure 7: Meshes for the parallelepiped example ( $E = 2.1 \times 10^5$ ;  $\nu = 0, 3$ ).

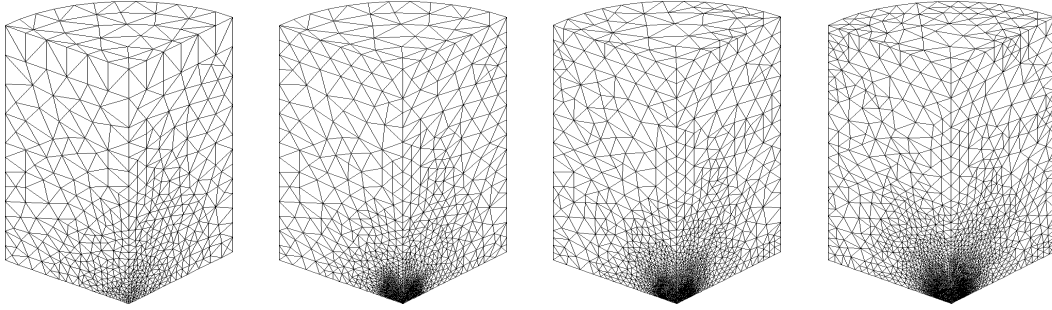


Figure 8: Meshes for the cylinder example ( $E = 2.1 \times 10^5$ ;  $\nu = 0, 3$ ).

The meshes for the beam example result in a smaller number of equations than in the other examples. Due to the presence of large gradients in the solution, however, both multigrid and the standard iterative techniques still require a large number of iterations to converge. This fact explains the choice of coefficients for the multigrid strategies in Figure 9.

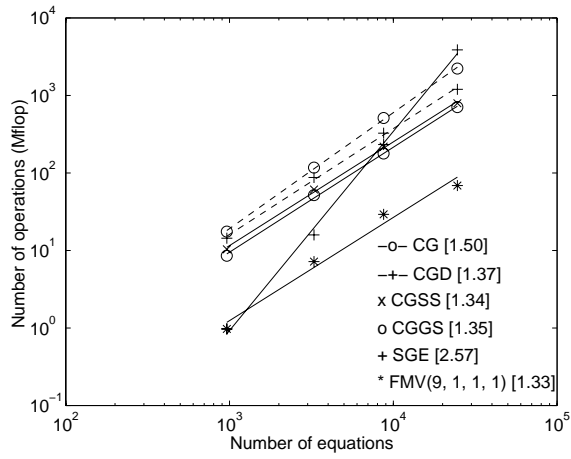
For multigrid methods applied to three-dimensional linear elastic problems, the number of equations in the coarsest mesh should be under 4000 to allow an efficient solution by sparse Gaussian elimination. The number 4000 is based on the performance of current workstations and PC's of today.

Iterative techniques are better in three-dimensional problems due to the smaller demand for memory. The acceleration obtained by using multigrid is significantly superior in terms of the number of operations when compared to conjugate gradient-based methods. The increase in memory space caused by multigrid methods is not significant. This allows all data used in the solution process to be loaded into main memory. For linear problems, multigrid is significantly superior to the iterative algorithms based on the conjugated gradient technique. Multigrid is also superior to direct sparse Gaussian elimination.

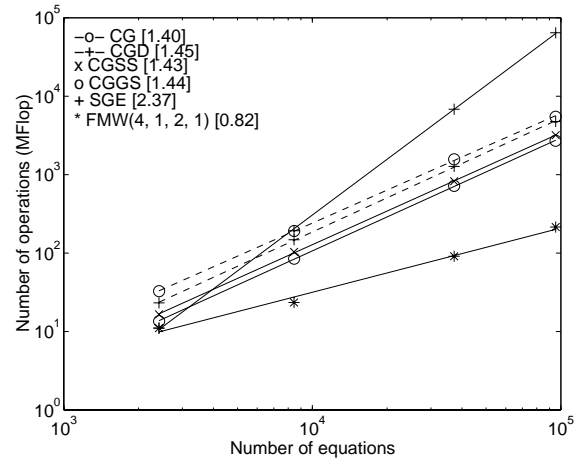
Figure 11 presents the average behavior for the examples with respect to sparse Gaussian elimination, CGGS, and multigrid. For multigrid, the number of operations varied almost linearly with respect to the number of equations.

## Acknowledgments

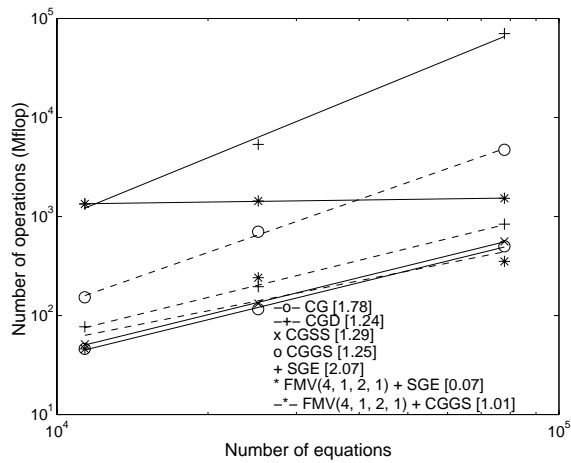
This work was partially supported by Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), Fundação de Amparo a Pesquisa do Estado de São Paulo (FAPESP), NSF



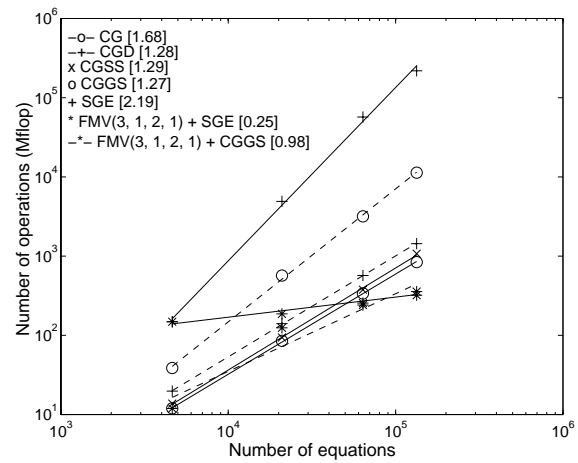
(a) Beam.



(b) Piston.

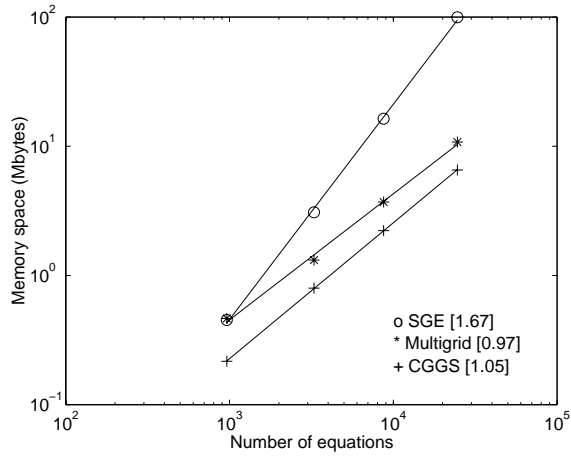


(c) Parallelepiped.

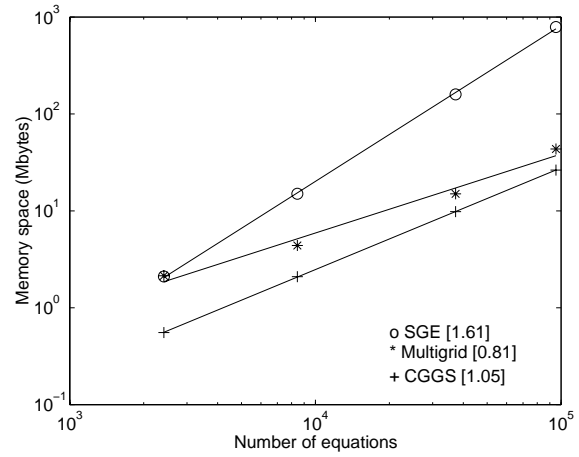


(d) Cylinder.

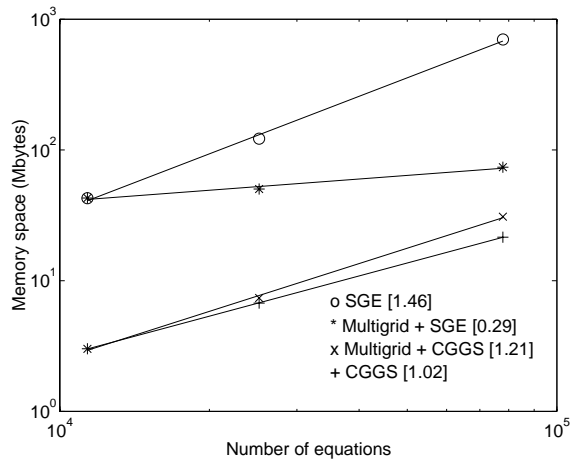
Figure 9: Number of operations.



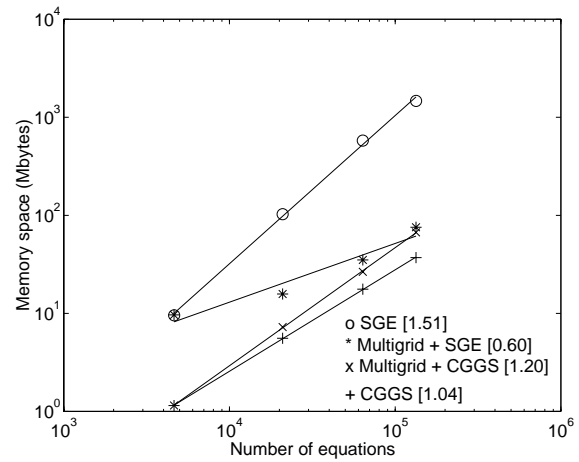
(a) Beam.



(b) Piston.



(c) Parallelepiped.



(d) Cylinder.

Figure 10: Memory space.

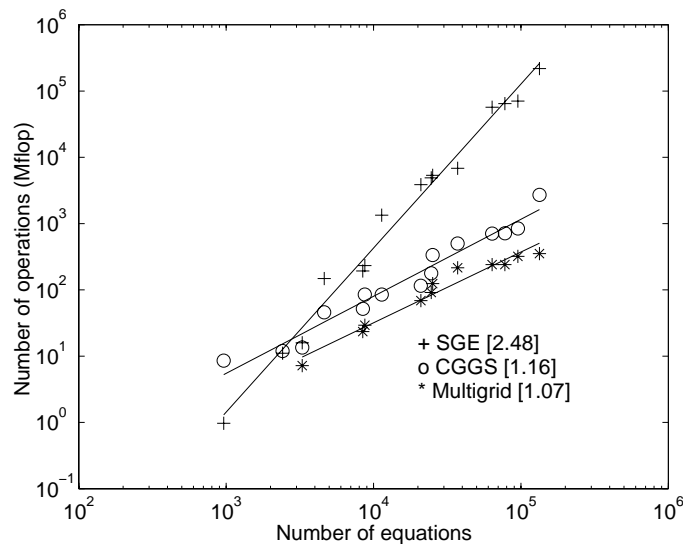


Figure 11: Number of operations of SGE, CGGS and multigrid methods.

grants 9707040 and 9721388, NATO grant CRG 971574, and the University of Kentucky Center for Computational Sciences. The authors are grateful for the software facilities provided by the TACSOM Group (<http://www.lncc.br/~tacsom>).

## References

- [1] O. Axelsson and V. A. Barker. *Finite Element Solution of Boundary Value Problems – Theory and Computation*. Academic Press, Orlando, 1984.
- [2] R. E. Bank and T. F. Dupont. An optimal order process for solving finite element equations. *Mathematics of Computation*, 15(153):35–51, 1981.
- [3] M. L. Bittencourt. *Adaptive Iterative and Multigrid Methods Applied to Non-Structured Meshes (in Portuguese)*. PhD thesis, State University of Campinas, Brazil, 1996.
- [4] M. L. Bittencourt, C. C. Douglas, and R. A. Feijóo. Adaptive non-nested multigrid methods. Submitted for publication, 1998.
- [5] M. L. Bittencourt, C. C. Douglas, and R. A. Feijóo. Non-nested and non-structured multigrid methods applied to elastic problems. Part I: The two-dimensional case. Submitted for publication, 1998.
- [6] M. L. Bittencourt and R. A. Feijóo. Non-nested multigrid methods in finite element linear structural analysis. In *Virtual Proceedings of the 8th Copper Mountain Conference on Multigrid Methods*, <http://www.mgnet.org>, April 1997. MGNet.
- [7] J. Bonet and J. Peraire. An alternating digital tree (ADT) algorithm for 3D geometric searching. *International Journal for Numerical Methods in Engineering*, 38:3529–3544, 1995.
- [8] J. H. Bramble, J. E. Pasciak, and J. Xu. The analysis of multigrid algorithms with nonnested quadratic forms. *Mathematics of Computation*, 56(193):1–34, 1991.

- [9] W. L. Briggs. *A Multigrid Tutorial*. SIAM Books, Philadelphia, 1987.
- [10] E. Dari. *Contribuciones a la Triangulación Automática de Dominios Tridimensionales*. PhD thesis, Instituto Balseiro, Bariloche, Argentina, 1994.
- [11] C. C. Douglas. Multi-grid algorithms with applications to elliptic boundary-value problems. *SIAM Journal on Numerical Analysis*, 21:236–254, 1984.
- [12] C.C. Douglas and J. Douglas, Jr. A unified convergence theory for abstract multigrid on multilevel algorithms, serial and parallel. *SIAM Journal on Numerical Analysis*, 30(1):136–158, 1993.
- [13] C.C. Douglas, J. Douglas, Jr., and D. E. Fyfe. A unified convergence theory for non-nested grids and/or quadrature. *East-West Journal Numerical Math*, 30(1):136–158, 1993.
- [14] A. George and J. W. Liu. *Computer Solution of Large Sparse Positive Definite Systems*. Prentice-Hall Series in Computational Mathematics, Englewood Cliffs, 1981.
- [15] M. E. Gurtin. *An Introduction to Continuum Mechanics*, volume 158 of *Mathematics in Science and Engineering*. Academic Press, 1981.
- [16] W. Hackbush. A fast iterative method for solving Poisson’s equation in general region. In *Numerical Treatment of Differential Equations (R. Bulirsch et al., editors)*, Berlin, 1977. Springer-Verlag.
- [17] S. F. McCormick. *Multigrid Methods*, volume 3 of *Frontiers Series*. SIAM Books, Philadelphia, 1987.
- [18] R. A. Nicolaides. On multiple grid and related techniques for solving discrete elliptic systems. *Journal of Computational Physics*, 19:418–431, 1975.
- [19] S. Pissanetzky. *Sparse Matrix Technology*. Academic Press, London, 1984.
- [20] L. R. Scott and S. Zhang. Higher-dimensional nonnested multigrid methods. *Mathematics of Computation*, 58(198):457–466, 1992.
- [21] S. Zhang. *Multi-Level Iterative Techniques*. PhD thesis, Department of Mathematics, Pennsylvania State University, 1988.
- [22] S. Zhang. Optimal-order nonnested multigrid methods for solving finite element equations I: On quasi-uniform meshes. *Mathematics of Computation*, 55(191):23–46, 1990.
- [23] S. Zhang. Optimal-order nonnested multigrid methods for solving finite element equations II: On non-quasi-uniform meshes. *Mathematics of Computation*, 55(191):439–450, 1990.
- [24] O. C. Zienkiewicz and J. Z. Zhu. A simple error estimator and adaptative procedure for practical engineering analysis. *International Journal for Numerical Methods in Engineering*, 24:337–357, 1987.