Finite Elements in Analysis and Design 57 (2012) 15–31

Contents lists available at SciVerse ScienceDirect

Finite Elements in Analysis and Design

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

Implicit solutions with consistent additive and multiplicative components

P. Areias [a,f,*], T. Rabczuk [b], D. Dias-da-Costa [c,d], E.B. Pires [e,f,*]

[a] Physics Department, University of Évora, Colégio Luís António Verney, Rua Romão Ramalho, 59, 7002-554 Évora, Portugal
[b] Institute of Structural Mechanics, Bauhaus-University Weimar, Marienstraße 15, 99423 Weimar, Germany
[c] INESC Coimbra, Rua Antero de Quental 199, 3000-033 Coimbra, Portugal
[d] Civil Engineering Department, University of Coimbra, Rua Luís Reis Santos, 3030-788 Coimbra, Portugal
[e] Departamento de Engenharia Civil e Arquitectura, Instituto Superior Técnico, Av. Rovisco Pais, 1049-001 Lisboa, Portugal
[f] ICIST, Portugal

ARTICLE INFO

Article history:
Received 17 November 2011
Accepted 12 March 2012
Available online 9 April 2012

Keywords:
Multiple-point constraints
Sparse matrices
Multibody dynamics
Computational fracture

ABSTRACT

This work describes an algorithm and corresponding software for incorporating general nonlinear multiple-point equality constraints in a implicit sparse direct solver. It is shown that direct addressing of sparse matrices is possible in general circumstances, circumventing the traditional linear or binary search for introducing (generalized) constituents to a sparse matrix. Nested and arbitrarily interconnected multiple-point constraints are introduced by processing of multiplicative constituents with a built-in topological ordering of the resulting directed graph. A classification of discretization methods is performed and some re-classified problems are described and solved under this proposed perspective. The dependence relations between solution methods, algorithms and constituents becomes apparent. Fracture algorithms can be naturally casted in this framework. Solutions based on control equations are also directly incorporated as equality constraints. We show that arbitrary constituents can be used as long as the resulting directed graph is acyclic. It is also shown that graph partitions and orderings should be performed in the innermost part of the algorithm, a fact with some peculiar consequences. The core of our implicit code is described, specifically new algorithms for direct access of sparse matrices (by means of the clique structure) and general constituent processing. It is demonstrated that the graph structure of the second derivatives of the equality constraints are cliques (or pseudo-elements) and are naturally included as such. A complete algorithm is presented which allows a complete automation of equality constraints, avoiding the need of pre-sorting. Verification applications in four distinct areas are shown: single and multiple rigid body dynamics, solution control and computational fracture.

1. Introduction

Most, if not all, discretizations of continuum engineering problems generate constituents belonging to two classes: additive constituents (finite and meshless elements including loading, contact elements and other smooth and non-smooth force elements) and multiplicative constituents (certain equality constraints, master–slave relations, rigid parts and arc-length constraints). This classification tolerates some overlapping, and a definite choice is usually made by considerations of efficiency. Specific formulations of many of such constituents are provided in the book by Belytschko et al. [14] and related Journals. Details concerning the systematic creation and combination of new constituents, independently of the specific problem treated, have not been shown with details in the literature before. A systematization of the technical implementation of models of mechanics, in the sense of Klarbring [26],[1] after discretization, is the aim of this work. This perspective is shared both by the governing equations, constraints and solution methods.

Concerning multiplicative components, although all equality constraints can be imposed with Lagrange multipliers, often this is uneconomical or inconvenient if a direct sparse solver is adopted (see, e.g. [24]). Essential boundary conditions are a good example of the effectiveness of multiplicative components used in many commercial and academic codes.[2] The same applies to rod and shell parameterization: director inextensibility is imposed

* Corresponding authors. Physics Department, University of Évora, Colégio Luís António Verney, Rua Romão Ramalho, 59, 7002-554 Évora, Portugal. ISI search: Areias P*. Tel.: +351 96 3496307; fax: +351 266745394.
E-mail addresses: pareias@civil.ist.utl.pt, pmaa@uevora.pt (P. Areias).
URL: http://evunix.uevora.pt/~pmaa (P. Areias).

[1] Chapter 12 shows several continuum applications of constraints, some as "constitutive assumptions".

[2] Usually, the affected coefficients are implicitly multiplied by zero, which is equivalent to the removal of the equations as will become apparent.

0168-874X/$ - see front matter © 2012 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.finel.2012.03.007

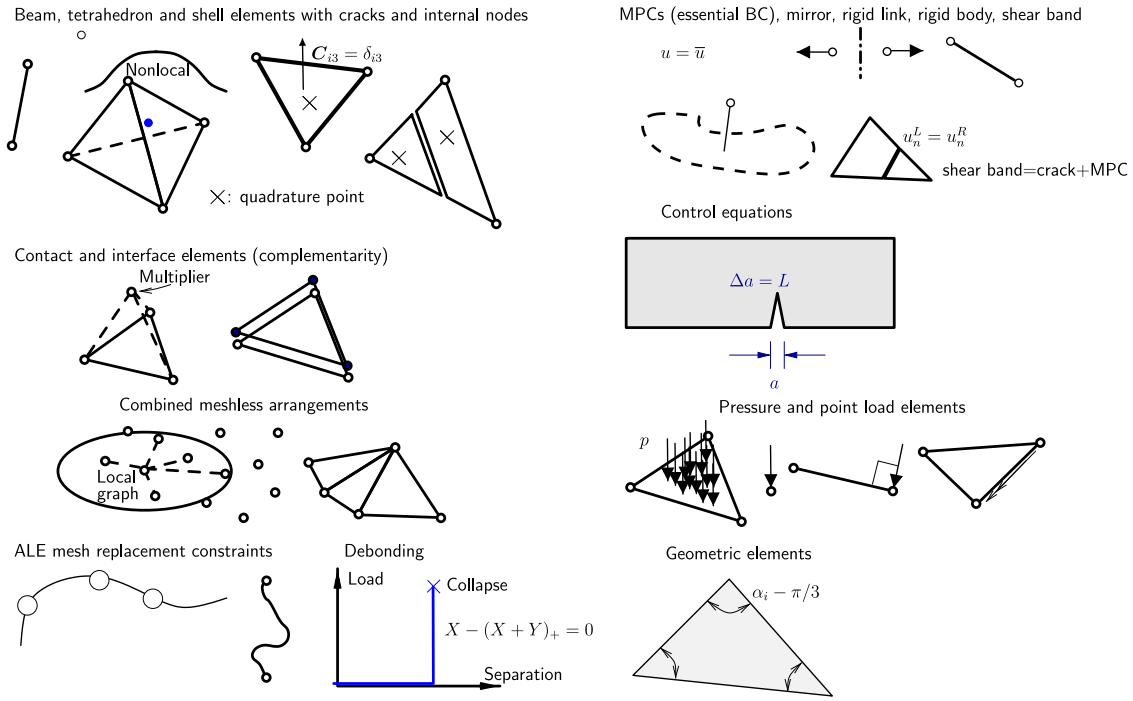

Fig. 1. Classification of common discretization components as either additive (elements) or multiplicative (MPC).

with multiplicative constituents (at the *continuum* level by coordinate transformation, see e.g. Antman [6,5]).

We here are concerned in imposing nodal trajectories, rigid body constraints and more complex interactions such as frictional contact. Generality is limited by the resulting DOF graph, as we shall see, but also the well-posedness of the resulting discrete system (dependent on the values of the coefficients). To incorporate all constituents prior to the solution (currently carried out by sparse methods of linear algebra), we apply transformations to a clique list of additive constituents incorporating specific cost-saving properties.

Contact and friction constituents, which introduce (nonlinear) complementarity conditions, are adequately treated with additive elements since they are often part of an active-set algorithm which deactivates constraints that cannot be active in a given step. Other behavior, such as rigid motion, kinematic links, periodicity boundary conditions (see, e.g. [10] for such an application), are best treated with multiple-point constraints (MPC) or, as a synonym, matrix transformation methods (MTM). In the context of multibody dynamics, these methods are also known as *coordinate reduction methods* [4]. These techniques have been increasingly relevant in recent years for unit cell analysis in multiscale methodologies. Multiple purpose shell and solid analysis and computational fracture [19,32] and PUFEM [30] methodologies exert a burden on software architecture. Necessary flexibility in modeling techniques motivates the present treatment of this problem.

Belytschko et al. [14] have shown that solid-based shell elements can be obtained by transformation of degrees-of-freedom of a standard 3D element. In addition, computational fracture techniques often make use of specific motions of the mesh; a shear band only allows tangential relative motion, a mode I crack only allows opening, etc. Another obvious application is static condensation, very convenient for mixed and hybrid FE element technology and also nodeless degrees-of-freedom. The presence of "condensable" degrees-of-freedom should be detected by the solver prior to decomposition and the subsequent post-processing of slave degrees-of-freedom should be effected without user intervention. Static condensation of the nodeless degrees-of-freedom is simple to code and can provide substantial savings [20]. Besides these classical problems, which are now well solved (with strong restrictions in their generality) by commercial software packages, MTM can be successfully used in solution control. Localized arc-length, COD-control and related techniques, which were, until now, introduced as a "added feature" prone to coding errors and maintenance requirements are reclassified as equality constraints and therefore MPC. As generalizations of boundary conditions for partial differential equations, essential boundary conditions are also classified as multiplicative constituents and natural boundary conditions as additive constituents; this classification is illustrated in detail in Fig. 1. The literature concerned with this subject is often restricted to direct sparse multiplication [2] (unrealistic for large-scale problems since it creates temporary objects of potentially enormous size and not easily parallelized) or unnested constraints [1,38]. Efficient methods are available for iterative sparse linear solvers [22] since the transformation matrix ($\boldsymbol{T}_\star$ in our notation) can pre-multiply the iterative solution, allowing considerable savings. Our approach works independently of the linear solver used, although we use a direct sparse solver.

2. Multiple-point constraints

2.1. The need for degree-of-freedom elimination

For certain equality constraints, such as those arising from prescribed degrees-of-freedom and rigid body motions, matrix transformation methods can be more efficient than the Lagrange multiplier or related methods.[3] This is particularly acute when many degrees-of-freedom appear in the constraints with a regular pattern (e.g., in rigid body constraints). The adjacency lists of a CSR (compact sparse row) or CSC (compact sparse column) representation of a sparse matrix graph (see [20] for this nomenclature)

[3] Such as the augmented-Lagrangian or perturbed-Lagrangian.

are useful by themselves (in the absence of the coefficients) for describing many-to-many relations. For example, connectivity lists (lists of nodes for each element—say NOEL) are based on the symbolic part of the CSR representation and use of the transposition generates the converse relation: elements for each node, say ELNO. Multiplication of the first by the second results in a graph relating each node to its neighbor, where the neighborhood relation is established by the elements. For computational fracture applications, it is often required to know which elements share a given edge, or which faces share a given node. The full adjacency lists are highly useful in this case.

When using clique structures for finite elements, it becomes apparent that classical list-based or simply CSR representations (cf. [20]) are inefficient for applications where numerous multiplications are performed. Besides nested constraints and arbitrary combinations, other related aspect is the insertion and access to the global stiffness matrix. Linear search is still widely employed in many codes. Balanced binary trees (specifically AVL due to the high number of insertions [27]) can, in theory, be a more efficient option and have been used for years by the first Author. However, direct access (i.e. $\bigcirc(1)$) is preferable. We here show a direct access algorithm combined with MTM in full generality. Self-balanced trees are only used in nodal grouping of degrees-of-freedom. A clique-based implementation, despite some well-known redundant operation costs, is easier to parallelize than a traditional row (or column) based solver and has better locality properties. The necessary operations occur on rectangular sparse matrices using either the CSR or clique format.

2.2. Independent constraints

The equality-constrained problem is described as follows. Starting with n degrees-of-freedom and corresponding nonlinear equations, a set of m nonlinear constraints is appended. Two residual vectors (containing n and m components, respectively) are introduced, corresponding to these two sets of equations: $\boldsymbol{f}$ and $\boldsymbol{g}$ whose components are of the class C^q, $q\geq 1$. The degrees-of-freedom are grouped in a n-dimensional array $\boldsymbol{a}$ and we can express the system as[4]

$$\boldsymbol{f}(\boldsymbol{a})=\boldsymbol{0} \tag{1}$$

$$\boldsymbol{g}(\boldsymbol{a})=\boldsymbol{0} \tag{2}$$

where the gradient of $\boldsymbol{g}$ with respect to a given subset (s) of m degrees-of-freedom is full rank (this is called the *submersion* assumption [37, p. 84])

$$\mathrm{RANK}(\boldsymbol{g}'_s)=m$$

for $\boldsymbol{a}\in\mathbb{R}^n$. This condition ensures that (2) is a C^q submanifold of $\mathbb{R}^n$. Since, for $m>0$, more equations than unknown degrees-of-freedom are introduced, a subset $n-m$ of $\boldsymbol{f}$, identified as $\boldsymbol{f}_r$, has to be retained. Along with this subset, the corresponding subset of $\boldsymbol{a}$, $\boldsymbol{a}_r$ is also selected. Succinctly, if $\boldsymbol{a}_r$, $r\in\mathcal{I}_r$ where $|\mathcal{I}_r|=n-m$ is the index set of retained (or *eliminated*) degrees-of-freedom and $\boldsymbol{a}_s$,$s\in\mathcal{I}_s$ where $|\mathcal{I}_s|=m$ is the set of slave, or dependent, degrees-of-freedom. We can split the degrees-of-freedom $\boldsymbol{a}$ as an ordered pair $\{\boldsymbol{a}_s,\boldsymbol{a}_r\}^T$. Components of this list are a_i with $i\in\mathcal{I}$. The choice of $\mathcal{I}_s$ is usually a matter of efficiency. It is noticeable that Eq. (1) can also be written as $\delta\boldsymbol{a}\cdot\boldsymbol{f}=0$ where $\delta\boldsymbol{a}$ is the virtual degree-of-freedom array (cf. [14]). When using this virtual degree-of-freedom array, we can apply the Newton method to the system (1) and obtain

$$\delta\boldsymbol{a}_r^T\boldsymbol{f}'_{rr}\,\mathrm{d}\boldsymbol{a}_r+\delta\boldsymbol{a}_s^T\boldsymbol{f}'_{sr}\,\mathrm{d}\boldsymbol{a}_r+\delta\boldsymbol{a}_r^T\boldsymbol{f}'_{rs}\,\mathrm{d}\boldsymbol{a}_s+\delta\boldsymbol{a}_s^T\boldsymbol{f}'_{ss}\,\mathrm{d}\boldsymbol{a}_s+\mathrm{d}\delta\boldsymbol{a}_s^T\boldsymbol{f}+\mathrm{d}\delta\boldsymbol{a}_r^T\boldsymbol{f}=-\boldsymbol{f} \tag{3}$$

where $\boldsymbol{f}_{rs}$ is the derivative of the r-part of the equation vector $\boldsymbol{f}$ with respect to $\boldsymbol{a}_s$, etc. The partition $\boldsymbol{f}=\{\boldsymbol{f}_s\boldsymbol{f}_r\}^T$ is assumed. Note that, in (3), the two last terms in the left-hand side only exist if the final degrees-of-freedom are related to $\boldsymbol{a}_s$ and $\boldsymbol{a}_r$ in a non-linear form. In particular, this occurs with rotations. Since the retained degrees-of-freedom are also considered final, the term $\mathrm{d}\delta\boldsymbol{a}_r$ is null. We can group the terms $\boldsymbol{f}'_{rr}$, $\boldsymbol{f}'_{rs}$, $\boldsymbol{f}'_{sr}$ and $\boldsymbol{f}'_{ss}$ in one matrix $\boldsymbol{K}$ split according to the previous partition

$$\boldsymbol{K}=\begin{bmatrix}\boldsymbol{f}'_{ss} & \boldsymbol{f}'_{sr}\\ \boldsymbol{f}'_{rs} & \boldsymbol{f}'_{rr}\end{bmatrix} \tag{4}$$

Under the previous condition for $\boldsymbol{g}'_s$, the application of Newton method to (2) results in[5]

$$\mathrm{d}\boldsymbol{a}_s=-\boldsymbol{g}_s'^{-1}\boldsymbol{g}'_r\mathrm{d}\boldsymbol{a}_r-\boldsymbol{g}_s'^{-1}\boldsymbol{g} \tag{5}$$

or, if $\boldsymbol{T}=-\boldsymbol{g}_s'^{-1}\boldsymbol{g}'_r$ and $\boldsymbol{b}_s=-\boldsymbol{g}_s'^{-1}\boldsymbol{g}$, we can write

$$\mathrm{d}\boldsymbol{a}_s=\boldsymbol{T}\mathrm{d}\boldsymbol{a}_r+\boldsymbol{b}_s \tag{6}$$

In the optimization literature the elimination of $\boldsymbol{a}_s$ results in the so-called reduced Hessian method (cf. [33], p. 487). We use a specific null-space matrix using the gradient, which is also called *variable reduction method*. The null-space property can be observed by rewriting

$$\begin{Bmatrix}\mathrm{d}\boldsymbol{a}_s\\ \mathrm{d}\boldsymbol{a}_r\end{Bmatrix}=\overbrace{\underbrace{\begin{bmatrix}\boldsymbol{T}\\ I_{(n-m)\times(n-m)}\end{bmatrix}}_{\text{null-sspace term}}}^{\boldsymbol{T}_\star}\mathrm{d}\boldsymbol{a}_r+\overbrace{\underbrace{\begin{Bmatrix}\boldsymbol{b}_s\\ \boldsymbol{0}\end{Bmatrix}}_{\text{corrective term}}}^{\boldsymbol{b}_\star} \tag{7}$$

In a more concise notation, (7) reads

$$\mathrm{d}\boldsymbol{a}=\boldsymbol{T}_\star\mathrm{d}\boldsymbol{a}_r+\boldsymbol{b}_\star \tag{8}$$

The derivative of $\boldsymbol{T}_\star$ with respect to $\boldsymbol{a}$ is given by

$$\boldsymbol{T}'_\star=-\boldsymbol{c}\boldsymbol{g}''\boldsymbol{T}_\star \tag{9}$$

where the matrix $\boldsymbol{c}$ is given by

$$\boldsymbol{c}=\begin{bmatrix}\boldsymbol{g}_s'^{-1}\\ \boldsymbol{0}_{(n-m)\times m}\end{bmatrix} \tag{10}$$

The second variation of $\boldsymbol{a}$ present in (3) is determined by the previous quantities (9) and (10). Using index notation, it results

$$\mathrm{d}\delta a_i=-\delta a_{r_j}T_{qj}c_{ip}g''_{pqk}T_{kl}\,\mathrm{d}a_{r_l} \tag{11}$$

with $i,k,q\in\mathcal{I}$, $p\in\mathcal{I}_s$ and $j,l\in\mathcal{I}_r$. Summation is implied in repeated indices. To the authors' knowledge, despite its straightforward appearance, this term was not considered before in the literature. Newton's iteration can be summarized as

$$\underbrace{\boldsymbol{T}_\star^T[\boldsymbol{K}-\boldsymbol{f}^T\boldsymbol{c}\boldsymbol{g}'']\boldsymbol{T}_\star}_{\boldsymbol{K}_\star}\mathrm{d}\boldsymbol{a}_r=-\underbrace{\boldsymbol{T}_\star^T(\boldsymbol{f}+\boldsymbol{K}\boldsymbol{b}_\star)}_{\boldsymbol{f}_\star} \tag{12}$$

where $\boldsymbol{K}_\star$ is the reduced stiffness matrix and $\boldsymbol{f}_\star$ is the reduced force vector. Iterations generated by (12) are tangent to the constraints' level surfaces. Compared with classical optimization works such as Byrd and Schnabel [16], there is no explicit Lagrange multiplier term in $\boldsymbol{K}_\star$. These can be calculated as follows: values for the reactions conjugate to $\boldsymbol{g}$ are grouped in an array of m Lagrange multipliers, $\boldsymbol{\lambda}^*$, which is obtained from $\boldsymbol{g}'$

[4] Note that this system can be written to comply with the constrained optimization notation (cf. [35]) as $\min[\frac{1}{2}\boldsymbol{f}(\boldsymbol{a})^T\boldsymbol{f}(\boldsymbol{a})]$, s.t. $\boldsymbol{g}(\boldsymbol{a})=\boldsymbol{0}$.

[5] There is the requirement of partitioning the list of degrees-of-freedom using the two index sets $\mathcal{I}_s$ and $\mathcal{I}_r$ by means of a permutation, see [35] concerning the definition of the required permutation matrices.

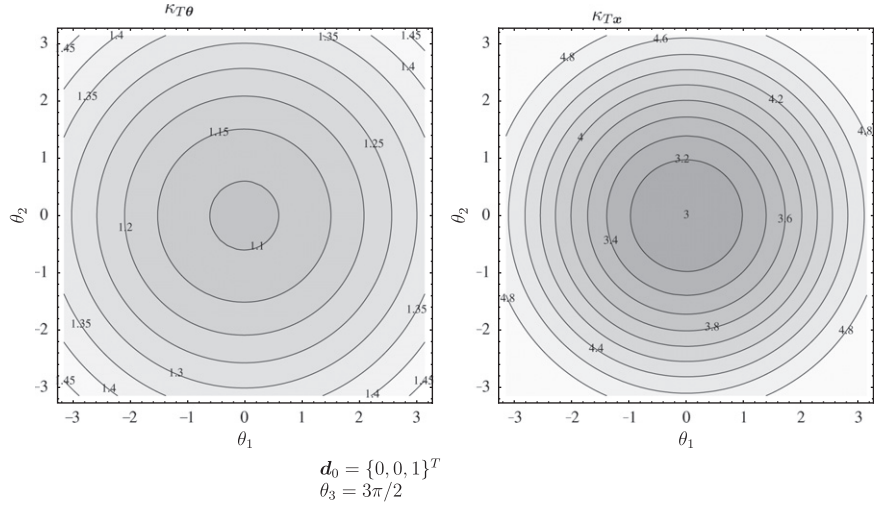

Fig. 2. Condition numbers for the two director parameterizations as a function of θ_1 and θ_2.

and $\boldsymbol{f}$ as[6]

$$\boldsymbol{\lambda}^* = -\boldsymbol{c}\boldsymbol{f}_s \tag{13}$$

Considering the graph structure, $\boldsymbol{K}$ and $\boldsymbol{f}^T\boldsymbol{c}\boldsymbol{g}''$ both result from sparse sums of clique graphs. The number of cliques for the formation of $\boldsymbol{K}$ is the same as the number of elements n_e. If sparse sum (e.g. [18]) is considered we can simply write

$$\boldsymbol{K} = \sum_{i=1}^{n_e} \boldsymbol{K}_i^e \tag{14}$$

$$\boldsymbol{f}^T\boldsymbol{c}\boldsymbol{g}'' = \sum_{j=1}^{m} \left[\left(\sum_{k=1}^{n_e} \boldsymbol{f}_k^e \right)^T (\boldsymbol{c}_j \boldsymbol{g}_j'') \right] \tag{15}$$

where $\boldsymbol{K}_i^e$ is the ith element stiffness matrix, $\boldsymbol{f}_k^e$ is the kth element force and $\boldsymbol{c}_j\boldsymbol{g}_j''$ is obtained from the jth constraint gradient and Hessian. The superscript e indicates a element quantity. The matrices $\boldsymbol{T}_\star$ and the vector $\boldsymbol{b}_\star$ must be fully formed (this will be detailed in the next section) before the multiplications by $\boldsymbol{T}_\star$ in (12) are performed. The actual implementation of (12) separates terms (14) and (15) since the degrees-of-freedom destinations of $\boldsymbol{g}''$ does not coincide with those of $\boldsymbol{K}$. From the graph structure perspective, $-\boldsymbol{f}^T\boldsymbol{c}\boldsymbol{g}''$ are also cliques, since the result connects retained degrees-of-freedom which are mutually visible. Recalling that our matrices are sparse, Eq. (12) can be written as

$$\boldsymbol{T}_\star^T \left(\sum_{i=1}^{n_e} \boldsymbol{K}_i^e \right) \boldsymbol{T}_\star + \boldsymbol{T}_\star^T \left\{ -\sum_{j=1}^{m} \left[\left(\sum_{k=1}^{n_e} \boldsymbol{f}_k^e \right)^T (\boldsymbol{c}_j \boldsymbol{g}_j'') \right] \right\} \boldsymbol{T}_\star \, \mathrm{d}\boldsymbol{a}_r \tag{16}$$

$$= -\boldsymbol{T}_\star^T \left(\sum_{j=1}^{n_e} \boldsymbol{f}_j^e \right) - \boldsymbol{T}_\star^T \left(\sum_{l=1}^{n_e} \boldsymbol{K}_j^e \right) \boldsymbol{b}_\star \tag{17}$$

The format of Eq. (16) discloses a useful property: edges of the graph structure of $\boldsymbol{K}_\star$ are completely defined by each $\boldsymbol{K}_i^e$ and the transformation matrix $\boldsymbol{T}_\star$. The term containing the constraints' Hessian $\boldsymbol{g}''$ will produce edges of the same graph, since it is also pre-and-post multiplied by $\boldsymbol{T}_\star$. The two cliques ($\boldsymbol{K}_i^e$ and $-\boldsymbol{f}^T\boldsymbol{c}_i\boldsymbol{g}_i''$) participate additively in the formation of the global stiffness matrix $\boldsymbol{K}_\star$. In terms of condition number of the reduced stiffness matrix, it can be shown that

$$\mathrm{cond}(\boldsymbol{K}_\star) \le \mathrm{cond}(\boldsymbol{K} - \boldsymbol{f}^T\boldsymbol{c}\boldsymbol{g}'') \underbrace{\mathrm{cond}(\boldsymbol{T}^T\boldsymbol{T} + \boldsymbol{I})}_{\kappa_T} \tag{18}$$

An application of (18) relies on the selection of degrees-of-freedom to eliminate (i.e. the selection of set $\mathcal{I}_s$) for $\boldsymbol{g}(\boldsymbol{a}) = \boldsymbol{0}$. This could be, in theory, performed automatically. However, in most engineering applications this is preferably left to the analyst since there are other factors to include. For example, let us consider the classical 3-parameter director representation with two distinct parameterizations:

- Exponential form with the axis-angle, $\boldsymbol{\theta}$.
- The parameterization with Rodrigues *parameters,* $\boldsymbol{x}$ (using the Cayley formula).

Let $\boldsymbol{d}$ represents a director in the deformed configuration and $\boldsymbol{d}_0$ the corresponding director in the undeformed configuration. Using either of the parameterizations, it is straightforward to show that $\boldsymbol{d}_\theta = \boldsymbol{R}_\theta(\boldsymbol{\theta})\boldsymbol{d}_0$ and $\boldsymbol{d}_x = \boldsymbol{R}_x(\boldsymbol{x})\boldsymbol{d}_0$ are, respectively[7]

$$\boldsymbol{d}_\theta = \boldsymbol{d}_0 + \frac{\sin\|\boldsymbol{\theta}\|}{\|\boldsymbol{\theta}\|}\boldsymbol{\theta}\times\boldsymbol{d}_0 + 2\frac{\sin^2\frac{\|\boldsymbol{\theta}\|}{2}}{\|\boldsymbol{\theta}\|^2}\boldsymbol{\theta}\times(\boldsymbol{\theta}\times\boldsymbol{d}_0) \tag{19}$$

$$\boldsymbol{d}_x = \begin{bmatrix} -1+\frac{2(1+x_1^2)}{1+x_1^2+x_2^2+x_3^2} & \frac{2(x_1x_2-x_3)}{1+x_1^2+x_2^2+x_3^2} & \frac{2(x_1x_3+x_2)}{1+x_1^2+x_2^2+x_3^2} \\ \frac{2(x_1x_2+x_3)}{1+x_1^2+x_2^2+x_3^2} & -1+\frac{2(1+x_2^2)}{1+x_1^2+x_2^2+x_3^2} & \frac{2(x_2x_3-x_1)}{1+x_1^2+x_2^2+x_3^2} \\ \frac{2(x_1x_3-x_2)}{1+x_1^2+x_2^2+x_3^2} & \frac{2(x_2x_3+x_1)}{1+x_1^2+x_2^2+x_3^2} & -1+\frac{2(1+x_3^2)}{1+x_1^2+x_2^2+x_3^2} \end{bmatrix} \boldsymbol{d}_0 \tag{20}$$

Using $\boldsymbol{a}_r$ as $\boldsymbol{\theta}$ and $\boldsymbol{a}_s$ as $\boldsymbol{d}_0$ for the exponential form, we can represent κ_T graphically to assess the two parameterizations, as Fig. 2 illustrates. The reason we mention the need for analyst input is that, although the exponential form appears favorable from the inspection of κ_T, formula (19) it has a $\frac{0}{0}$ indetermination at the origin, which is of cumbersome computational treatment. In addition, the calculation of the first and second derivatives for the Cayley formula is straightforward.

At this point, the reader can observe that a set of independent constraints established as $\boldsymbol{g} = \boldsymbol{0}$ with the variable reduction method can be replaced by m equations applied regardless of the dependence. The interconnected case is therefore also the general case.

[6] After a trivial manipulation of the expression in [33, p. 495].

[7] $\boldsymbol{x} = \tan(\|\boldsymbol{\theta}\|/2)\boldsymbol{\theta}/\|\boldsymbol{\theta}\|$.

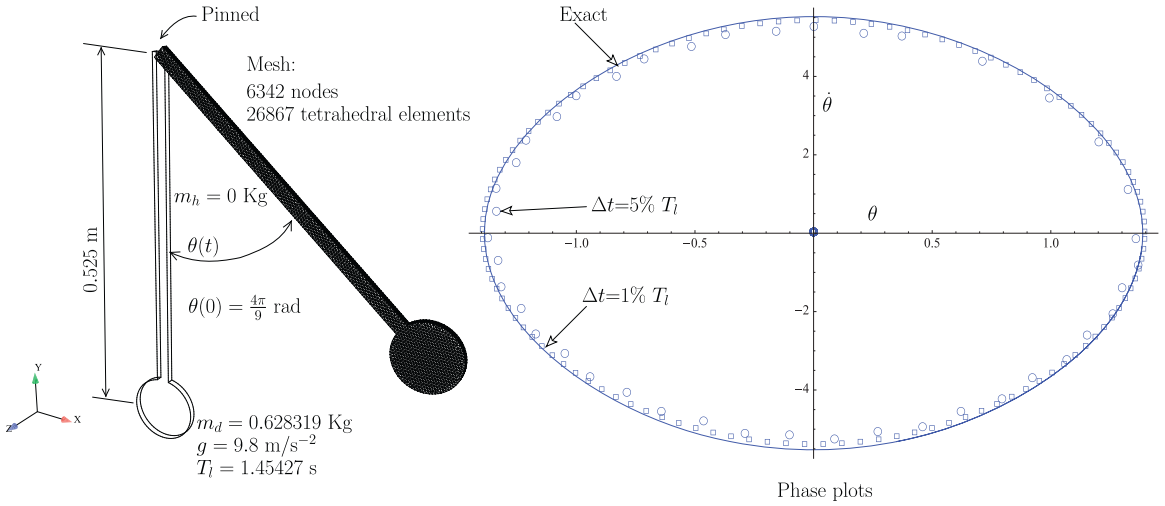

Fig. 3. Large amplitude pendulum (rigid-body constraint) integrated with two time-steps (1% and 5% of the linear period).

2.3. Inertial forces

Many studies in multibody dynamics are focused in constraint imposition and time integration (see, e.g. [34]). The proposed algorithm can be directly used for multibody dynamics without specific requirements. Considering time-step algorithms and using the subscript n for a given time step and $n+1$ for the subsequent time step, we can write the second time derivative of $\boldsymbol{a}$ as a function of $\boldsymbol{a}_n$, $\boldsymbol{a}_{n+1}$, $\dot{\boldsymbol{a}}_n$ and $\ddot{\boldsymbol{a}}_n$

$$\ddot{\boldsymbol{a}}_{n+1} = \ddot{\boldsymbol{a}}(\boldsymbol{a}_n, \boldsymbol{a}_{n+1}, \dot{\boldsymbol{a}}_n, \ddot{\boldsymbol{a}}_n) \tag{21}$$

The total force vector including inertial forces is given by

$$\boldsymbol{f}_{\text{dyn}} = \boldsymbol{f} + \boldsymbol{M}\ddot{\boldsymbol{a}}_{n+1} \tag{22}$$

where $\boldsymbol{M}$ is an appropriate mass matrix (cf. [14]). We can therefore write the unconstrained solution scheme as

$$\delta\boldsymbol{a}^T \underbrace{\left(\boldsymbol{K} + \boldsymbol{M}\frac{\partial \ddot{\boldsymbol{a}}}{\partial \boldsymbol{a}_{n+1}}\right)}_{\boldsymbol{K}_{\text{dyn}}} \mathrm{d}\boldsymbol{a} = -\delta\boldsymbol{a}^T \underbrace{(\boldsymbol{f} + \boldsymbol{M}\ddot{\boldsymbol{a}}_{n+1})}_{\boldsymbol{f}_{\text{dyn}}} \tag{23}$$

Damping is indirectly included as viscous constitutive behavior and hence it is not explicitly present in (23). It is very clear that there is no need to calculate the inertia matrix since it is accounted by the transformation technique. Beam dynamics which result in intricate inertia forces are also taken care by our approach if director constraints are imposed by MPC. The incorporation of inertial forces in the analysis with constraints is performed in a straightforward manner:

- The function $\ddot{\boldsymbol{a}}$ is specified for a given time-integration method, as well as the derivative with respect to $\boldsymbol{a}_{n+1}$ (and the half-step $\boldsymbol{a}_{n+1/2}$).
- $\boldsymbol{f}_{\text{dyn}}$ replaces $\boldsymbol{f}$ in (12).
- $\boldsymbol{K}_{\text{dyn}}$ replaces $\boldsymbol{K}$ in (12).

The half-step mean-acceleration/three-point backward Euler time-integration algorithm is used as a prototype model (it is described in [13]). In that case we specify $\ddot{\boldsymbol{a}}$ as

$$\ddot{\boldsymbol{a}} = \begin{cases} \dfrac{16(\boldsymbol{a}_{n+1/2} - \boldsymbol{a}_n)}{\Delta t^2} - \dfrac{8\dot{\boldsymbol{a}}_n}{\Delta t} - \ddot{\boldsymbol{a}}_n, & h_s = 1 \quad (\ddot{\boldsymbol{a}} \equiv \ddot{\boldsymbol{a}}_{n+1/2}) \\ \dfrac{1}{\Delta t}\dot{\boldsymbol{a}}_n - \dfrac{4}{\Delta t}\dot{\boldsymbol{a}}_{n+1/2} + \dfrac{3}{\Delta t}\dot{\boldsymbol{a}}_{n+1}, & h_s = 2 \quad (\ddot{\boldsymbol{a}} \equiv \ddot{\boldsymbol{a}}_{n+1}) \end{cases} \tag{24}$$

where h_s is the homotopy step counter (two homotopy steps are used). It is also worth noting that the rigid-body constraint results in the application of (24) to all degrees-of-freedom, regardless of being slaves or not. Rotational inertia is *indirectly* considered by the application of the rigid-body constraint but all classical terms (cf. [12]) are included. An simple application is the pendulum which we can of course integrated in closed form. Fig. 3 shows an application with two fixed time step increments ($\Delta t = 1\%$ and $\Delta t = 5\%$ of the linear period T_l) for $E = \infty$ (with the rigid body multiple-point constraint). Exceptional robustness and accuracy are verified.

Prescribed degrees-of-freedom are applied as multiple-point-constraints and therefore naturally included in the proposed framework. However, initial conditions are not topologically sorted and must be preliminarily sorted to be applied to force-transmitting constituents.

2.4. Interconnected constraints

Assuming that an order of constraint application is pre-established (this order will be determined by a topological ordering), then each constraint beyond the first one will be applied to an already constrained system. It is obvious that if a certain constraint only affects degrees-of-freedom of the unconstrained system, it should be among the first to be applied. If we assume all constraints to be interconnected, then an ordered sequence must follow according to the closeness to the original degrees-of-freedom. Each constraint will contribute with a matrix $\boldsymbol{T}_{\star l}$, a vector $\boldsymbol{b}_{\star l}$ a matrix $\boldsymbol{c}_l$ and the tensor $\boldsymbol{g}_l''$. To facilitate the interpretation, matrices $\boldsymbol{T}_\star$, after ordering, relate degrees-of-freedom at position l with the ones at several positions which are farther away from the original degrees-of-freedom.[8] The generalization of the slave update formula for m *interconnected* equality constraints is presented, after the preliminary step of topological ordering, as

$$\boldsymbol{T}_\star^m = \prod_{l=m}^{1} \boldsymbol{T}_{\star l} \tag{25}$$

$$\boldsymbol{b}_\star^m = \sum_{l=m}^{1} \left[\left(\prod_{p=m}^{l} \boldsymbol{T}_{\star p} \right) \boldsymbol{b}_{\star l} \right] \tag{26}$$

Note that, since not all degrees-of-freedom participate in the constraints, the transformation matrices $\boldsymbol{T}_{\star l}$ contain the

[8] In the sense of the original unconstrained system.

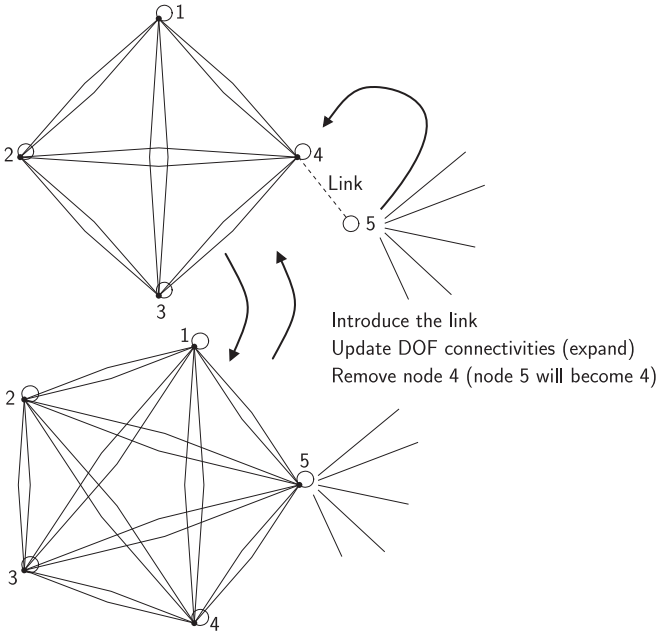

Fig. 4. Example of a local graph update from the introduction of a rigid link. The slave node (here 4) is removed from the graph.

appropriate unit diagonals corresponding to these. Both $\boldsymbol{T}_{\star l}$ and $\boldsymbol{b}_{\star l}$ are sparse, but with different properties: in the sparse $\boldsymbol{T}_{\star}$-matrices there are 1's for degrees-of-freedom that remain active and in the sparse $\boldsymbol{b}_{\star}$-vectors these will be 0. This perspective of interconnected[9] constraints is motivated by classical static analysis. Fill-in (or profile) concerns during Gauss decomposition are described in earlier works [1,38,17] but can now be attenuated[10] with the Approximate Minimum Degree (AMD) [3] and, in a lesser extent, with profile compressors (such as the one of Kumfert and Pothen [28]). Many MPC applications only moderately increase the fill-in in decomposition if an efficient post-ordering is performed, as the rigid link of Fig. 4 suggests: DOF renumbering must be performed after the graph updating. The user must specify $\boldsymbol{T}_{\star}$ and $\boldsymbol{b}_{\star}$ either obtained explicitly from the knowledge of the problem, or preprocess the constraint in the form $\boldsymbol{g}(\boldsymbol{a})=\boldsymbol{0}$. A pseudo-code preprocessor is shown as Algorithm 1. The use of a sparse linear solver is important at this preliminary stage since $\boldsymbol{g}_s$ is frequently very sparse (often close to the identity matrix) and $|\mathcal{I}_s|$ can be large. Many calculations make use of cliques, since it is well known that clique processing allows for computational savings as it circumvents the need for a dynamic structure (see, e.g. [17]).

Algorithm 1. Pre-processing of a single constraint $\boldsymbol{g}(\boldsymbol{a})=\boldsymbol{0}$.

```
!*** pre-processing of a single constraint
mpctreat(gr,gs,eq,b,t)
! allocates b and t in the heap
! solve for b and t with multiple right-hand-sides
(with sparse solver):
gs.[b|t]=-[eq|gr]
```

3. One-to-many and many-to-many adjacency lists

The adjacency lists of a CSR representation of a sparse matrix define many-to-many relations (the seminal work of Gustavson [23] explores this aspect by means of the transposition algorithm). The complete CSR representation can identify a graph or a digraph. This latest representation is of concern here. Each row index (or a map of it) may represent an entity of a given type and each column another entity of the same or other type. Of course, the natural row order may be inappropriate and a mapping can be used for the rows.[11] Furthermore, the natural order of the column indices provides more than the strictly required for a digraph (see, on this subject, [21]), with some interpretation, a relation with the Kuper and Vardi "logical data model" [29]. Two interpretations occur

- The pair (**mp**(**i**),**d**(**p**(**i**)-1+**j**)) where **i** is the natural position of the row, **mp**(**i**) the image under some map and **j** is the local column index represents an edge of a digraph.
- The row **d(p(i)):d(p(i=1)-1)** represents a generalized edge of a hypergraph.

The definition of a one-to-many relation is used in the construction of connectivities and related structures for constituents (typically, element input is performed with each element defined as a set of nodes). When one constituent is tied to a set of constituents, its local numbers must be mapped to the numbers of the set. This is a simple mapping described in Algorithm 2. The one-to-many relation is represented by a list, where the natural ordering provides the "many" part of the relation and the "one" is the number stored at every position. For example, a list of degrees-of-freedom (DOF) related with a hypothetical constituent can be represented by a list: **dof**={3,2,2,4,1,2}. The one-to-many relation is interpreted as: `DOF1` is related to local number `5`, `DOF2` is related to local numbers `2`, `3` and `6`, etc. Both the corresponding many-to-many complete representation and its transpose can be obtained as shown in Algorithm 2 (routine **enlarge**). For clarity, the beginning is represented by the letter **p** (pointer) and the lists are stored in a list **d** (destination). No search (linear or binary[12]) operations are required.

Algorithm 2. Transposition of a many-to-many representation and conversion of a one-to-many to a many-to-many representation.

```
!*** transposition of a many-to-many adjacency list
transp(n1,p1,d1,n2,p2,d2,ij)
  n2=max(d1)
  do i=1,n1
    do j=p1(i),p1(i+1)-1
      k=d1(j)
      p2(k)=p2(k)+1
    end do
  end do
  lol=p2(1)
  p2(1)=1
  do i=1,n
    new=p2(i)+lol
    i1=i+1
    lol=p2(i1)
    p2(i1)=new
  end do
  l=0
  do i=1,n1
    do j=p1(i),p1(i+1)-1
```

[9] This nomenclature is also adopted in textbook statics, e.g. [31].

[10] The reader can note that Lagrange multiplier methods may also increase the fill-in due to pivoting.

[11] Three integer arrays are used for each CSR representation (**p**, **d** and **mp**) which store the initial position of each row, the column number and the row map.

[12] Note that appropriate data structures for binary search (AVL or Red-Black trees, Splays, hash tables, etc.) do not provide direct access.

```
    l=l+1
      k=d(j)
      next=p2(k)
      ij(next)=l
      p2(k)=next+1
      d2(next)=i
      end do
    end do
    do i=n2,1,-1
      p2(i+1)=p2(i)
    enddo
    p2(1)=1
end
!*** conversion from a one-to-many to
!*** a many-to-many representation
enlarge(nl,list,nt,pt,dt,n,p,d,ij)
  pt(1)=1
  do i=2,nl+1
    pt(i)=pt(i-1)+1
  end do
  nt=nl
  dt=list
  transp(nt,pt,dt,n,p,d,ij)
end
```

4. Assembling

4.1. Symbolic and numeric assembling

For assembling, Gustavson [23] explicitly used a transpose and a temporary array with the column positions in the original clique matrices. These two operations can be avoided by creating a dedicated $\boldsymbol{e}_g^T\boldsymbol{e}_g$ sparse multiplication where $\boldsymbol{e}_g$ is the element degrees-of-freedom connectivity list. This step lacking in Gustavson's method is detailed here, as well as the procedure for direct addressing, not shown in that paper. The algorithm is shown in listing 3; note that large storage can be avoided by invoking a routine to form a specific element matrix. With MPC, modified element connectivity tables are necessary to obtain a new ordering that reduces the fill-in. As an additional benefit, memory fragmentation is minimized. There are repetitions in MPC multiplications, since DOF are usually shared by more than one node, but memory movements are reduced. Having the structure defined, assembling of a single element is performed as in listing 4. Only the essential operations are shown, as further details can be consulted in [7].

Algorithm 3. Symbolic assembling.

```
!*** symbolic assembling
symassemb(nel,lep,led,clqp,clqd,neq,mp,md)
  transp(nel,lep,letp,neq,led,letd,ijle)
  clqaddress(nel,lep,lep,clqp) ! obtains clique
addresses
  atimesb1(neq,letp,letd,nel,lep,led,igash,mp)
  l=0
  do ira=1,neq
    do iza=letp(ira),letp(ira+1)-1
      iel=letd(iza)
      igl=ijle(iza)
      jgl=0
      do izb=lep(iel),lep(iel+1)-1
        jgl=jgl+1
        mpb=led(izb)
        ip=iw(mpb)
        if(ip.eq.0)then
          l=l+1
          md(l)=mpb
          iw(mpb)=l
          llp=indstiff(clqp,lep,iel,igl,jgl)
          call insert(clqd,llp,l)
        else
          llp=indstiff(clqp,lep,iel,igl,jgl)
          call insert(clqd,llp,ip)
        end if
      end do
    end do
    do izc=mp(ira),l
      iw(md(izc))=0
    end do
  end do
end
!*** half sparse multiplication
atimesb1(na,ia,ja,nb,ib,jb,nc,ic)
  ncb=numinj(nb,ib,jb)
  nc=na
  do i=1,na
    ldg=0
    llast=-1
    do j=ia(i),ia(i+1)-1
      jr=ja(j)
      do k=ib(jr),ib(jr+1)-1
        jc=jb(k)
        if(iw(jc).eq.0)then
          ldg=ldg+1
          iw(jc)=llast
          llast=jc
        end if
      end do
    end do
    ic(i)=ldg
    do k=1,ldg
      j=iw(llast)
      iw(llast)=0
      llast=j
    end do
  end do
  mudlis(nc,ic) ! creates pointers from number of
elements
end
```

Algorithm 4. Numerical assembling.

```
!*** numerical assembling
nmassb(iel,ind,clqp,clqd,estif,matrix)
  nedof=ind(iel+1)-ind(iel)
  do jedof=1,nedof
    do iedof=1,nedof
      iz=clqd(indstiff(clqp,ind,iel,iedof,
      jedof))
      matrix(iz)=matrix(iz)+estif(id2d(nedof,
      iedof,jedof))
    end do
  end do
end
  !*** index in a clique list
  indstiff(p,ind,iel,iedof,jedof)
    indstiff=p(iel)-
    1+id2d(mmaddress(ind,iel,0),
    iedof,jedof)
end
```

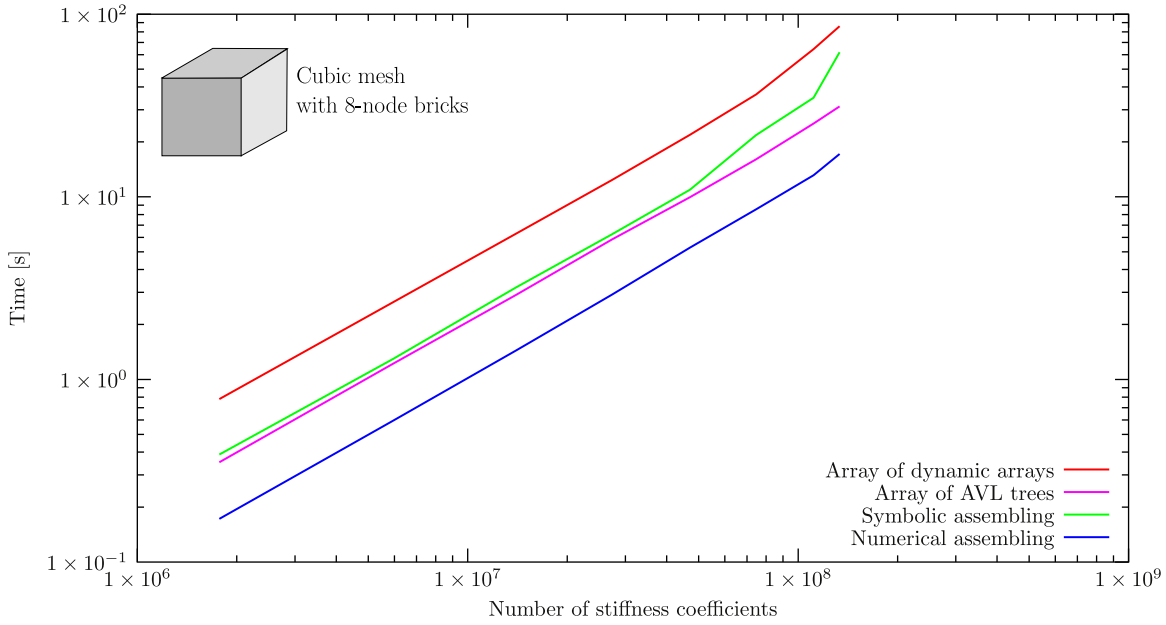

Fig. 5. Assembling times as a function of number of stiffness matrix coefficients. Machine: Apple MacBook Pro 2.66 GHz Intel Core i7, 8 GB RAM. Compiler: gfortran (GCC 4.5.0) with -O3 option.

Remarks:

- To minimize non-productive operations, conditionals are absent from the numerical assembling stage.
- In terms of operation count, it is equivalent to add terms before assembling or by the assembling process. This means that in the element DOF lists, repetitions do not have extra costs. List repetitions allow significant simplifications in the MPC algorithm.

4.2. Recovery of slave degrees-of-freedom

After the linear solution is carried out for the master degrees-of-freedom, slave values must recovered and reactions calculated (part of these are calculated in the assembling loop). The pseudo-code to perform this task is shown in Algorithm 5. See also [7] for further details.

Algorithm 5. Recovery of slave degrees-of-freedom.

```
...
  soluc=0.0
  do i=1,n
    ityp=typdf(i) ! type of dof
    soluc(ityp)=newdestvec(i) ! part of the
    solution
    do j=p(i),p(i+1)-1
      if(d(j).ne.0)then
        itemp=nwdof(d(j)) ! dof number
        if(itemp.ne.0)then
          soluc(ityp)=soluc(ityp)+mat(j)*
          vec(itemp) ! update of solution
        end if
      end if
    end do
  end do
...
```

4.3. Performance comparison

To assess the performance of both the numerical and symbolic parts of the assembling algorithm we compare its performance with two other implementations of different data structures. Note that flexibility is delegated for the column number lists, since the total number of degrees-of-freedom is known prior to filling the global stiffness matrix from a simple calculation.

- Array of self-resizing arrays, allowing linear search but requiring resizing operations (we double the required size every time a resize is needed). An analogous approach with a linked-list is discussed by Duff et al. [20].
- Array of AVL trees [27], allowing binary search but requiring branch balancing.
- Our clique/adjacency structure, allowing direct access with symbolic pre-processing required.

Fig. 5 shows the results. Some conclusions are:

- The linear search using an array of dynamic arrays is clearly slower than the other two options.
- The array of AVL trees results slightly faster than the symbolic part of the assembling technique proposed here. After the symbolic part is performed, the numerical assembling is much faster than the array of AVL trees.
- The direct access provided by the preliminary symbolic assembling is clearly faster than the two alternatives.

Further improvements of the numerical assembling performance can be achieved by ordering the element loop according to the destinations in the global stiffness matrix.

4.4. Recursive processing of MPC by clique format operations

We introduce the notion of extension number, en_i, of a degree-of-freedom. This is the cardinality of the set of masters tied to that degrees-of-freedom. Degrees-of-freedom which do not participate as slaves in a MPC have unitary extension numbers (and are considered their own masters). Slave degrees-of-freedom can have any non-negative en_i (for example Dirichlet conditions result in a zero en_i). Consider the DOF arrangement of Fig. 6 where the directed graph and the Hasse diagram for this arrangement are shown. Traversing from the top the Hasse diagram we obtain the

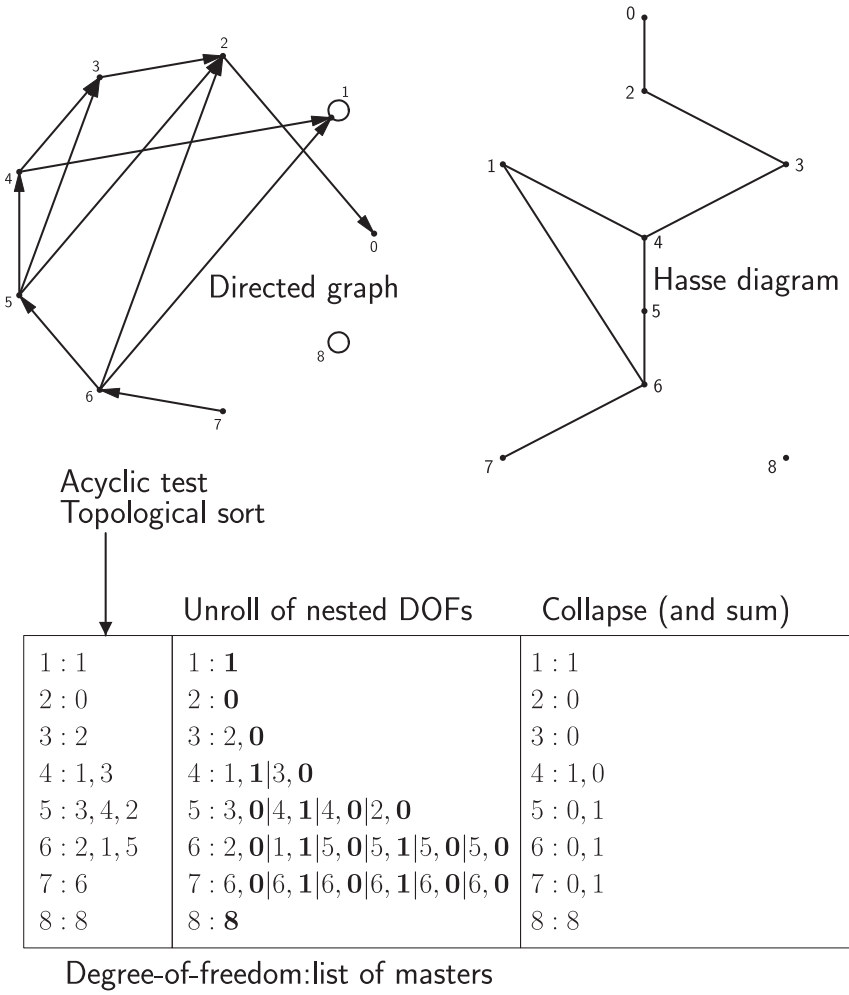

Fig. 6. Specific DOF distribution: directed graph and Hasse diagram. Collapse of DOF destinations.

correct sequence for DOF processing. Note that if the graph is cyclic, the problem is ill-posed since a DOF cannot be simultaneously slave and not slave. The one in the picture is acyclic [25]. Due to the self-loop in DOF 1, it is positioned at the same level of DOF 3. Self-loops are only possible in non-slave DOF (i.e. a slave DOF cannot master itself).

In the sequence of operations in Fig. 6, it can also be observed that DOFs are sorted by their inter-dependence. In this case, after collapse, only two DOFs survive: 1 and 8. Surviving DOF are characterized by having no proper outer edges. Two properties from graph theory [25] are relevant for our application (proofs are given in that reference):

- A partially order set corresponds to an acyclic directed graph.
- Every directed graph admits a topological ordering.
- The resulting DOF depth is at most 2, and can be made exactly either 2 or 0.

Algorithm 6. Verify if a given digraph given by p_{old} and d_{old} is acyclic and perform a topological ordering.

```
doftop(na,p,d,acyclic,top)
  m=1
  do i=1,na
    if(d(p(i)).ne.i)then
      do j=p(i),p(i+1)-1
        if(d(j).gt.0)ind(d(j))=ind(d(j))+1
      end do
    end if
  end do
  ik=0
  do i=1,na
    if(ind(i).eq.0)then
      ik=ik+1
      l(na+1-ik)=i
    end if
  end do
  mk=na
  do while(ik.ne.0)
    i=l(mk)
    mk=mk-1
    ik=ik-1
    top(m)=i
    m=m+1
    if(d(p(i)).ne.i)then
      do j=p(i),p(i+1)-1
        ig=d(j)
        if(ig.gt.0)then
          ind(ig)=ind(ig)-1
          if(ind(ig).eq.0)then
            ik=ik+1
            l(mk+1-ik)=ig
```

```
    end if
  end if
 end do
end if
```

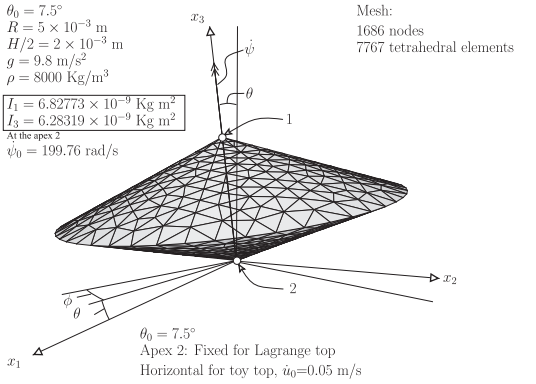

Fig. 7. Lagrange and toy tops: relevant geometrical data and mass properties.

```
end do
if(m.eq.na+1)then
  acyclic=.true.
else
  acyclic=.false.
end if
do i=1,na/2
  i1=top(na+1-i)
  top(na+1-i)=top(i)
  top(i)=i1
end do
end
```

We convert the pair **pold**, **dold** by the pair **p**, **d** performing the operations in Algorithm 7. User input must guarantee that the digraph is acyclic (a test is performed at the sorting stage) and, after that, a partial ordering must be established from the DOF edges. This extension is usually called topological order [25]. Algorithm 6 shows this operation. A solvable problem results in a Direct Acyclic Graph (DAG). The scheduling of DOF processing is

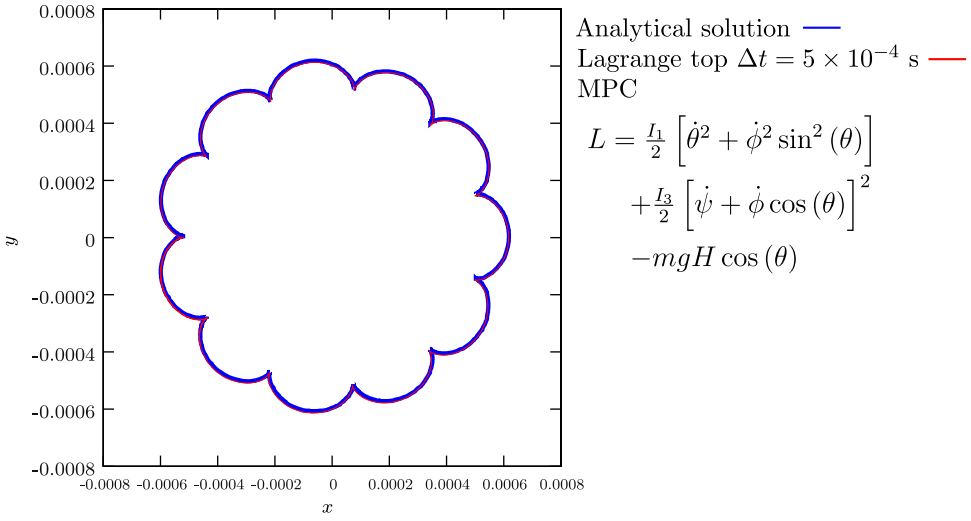

Fig. 8. Apex 1 trajectory for the Lagrange top.

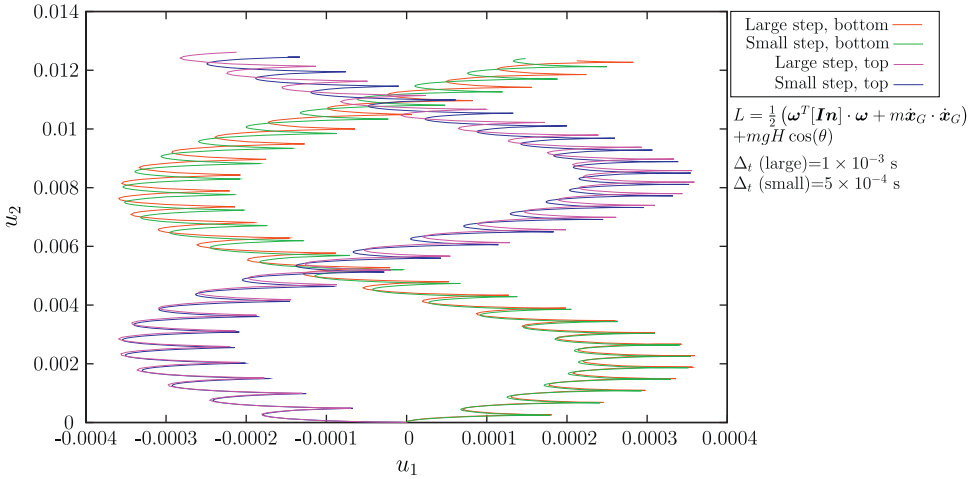

Fig. 9. Apexes 1 and 2 displacement components for the toy top.

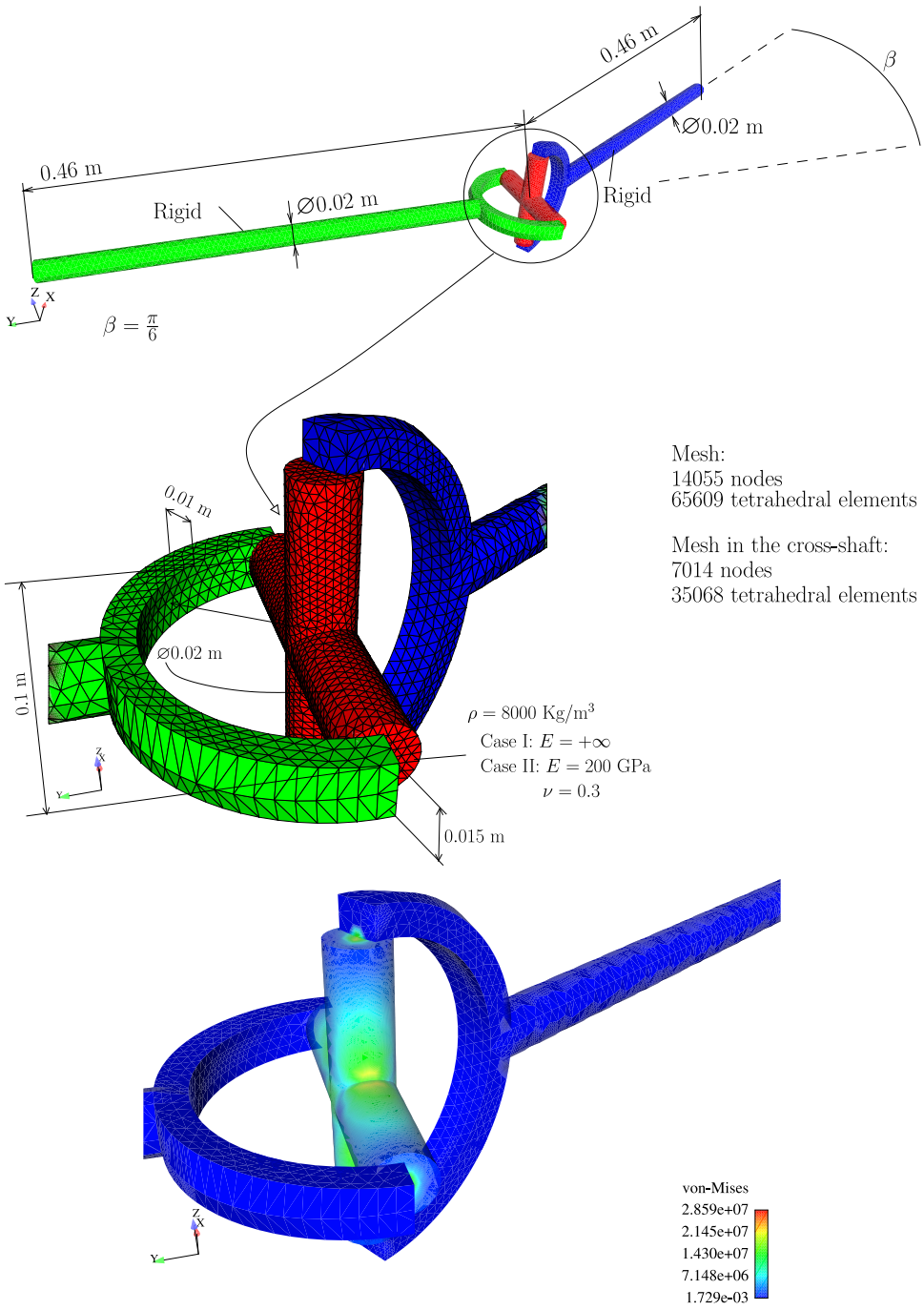

Fig. 10. Universal joint: geometry and relevant problem data. The von-Mises equivalent stress at the cross-shaft (the only deformable part) is shown.

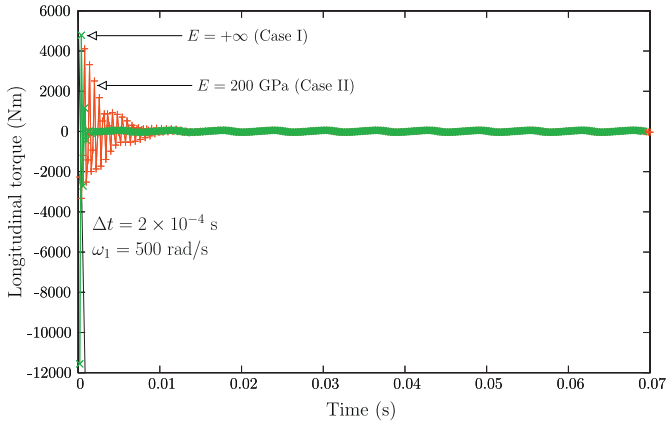

Fig. 11. Universal joint: response to constant angular velocity at the input shaft.

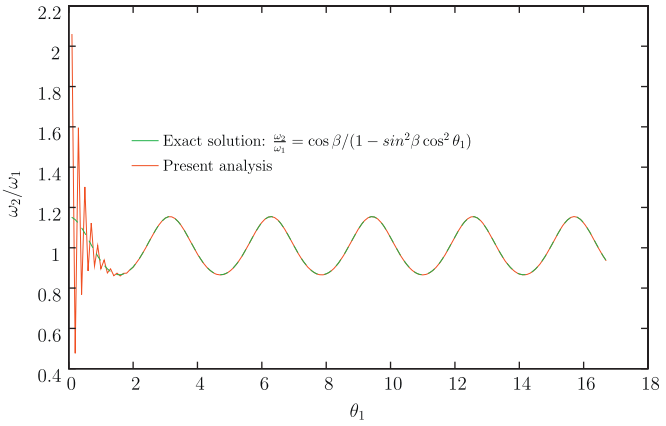

Fig. 12. Universal joint: ratio between angular velocities, comparison with exact solution.

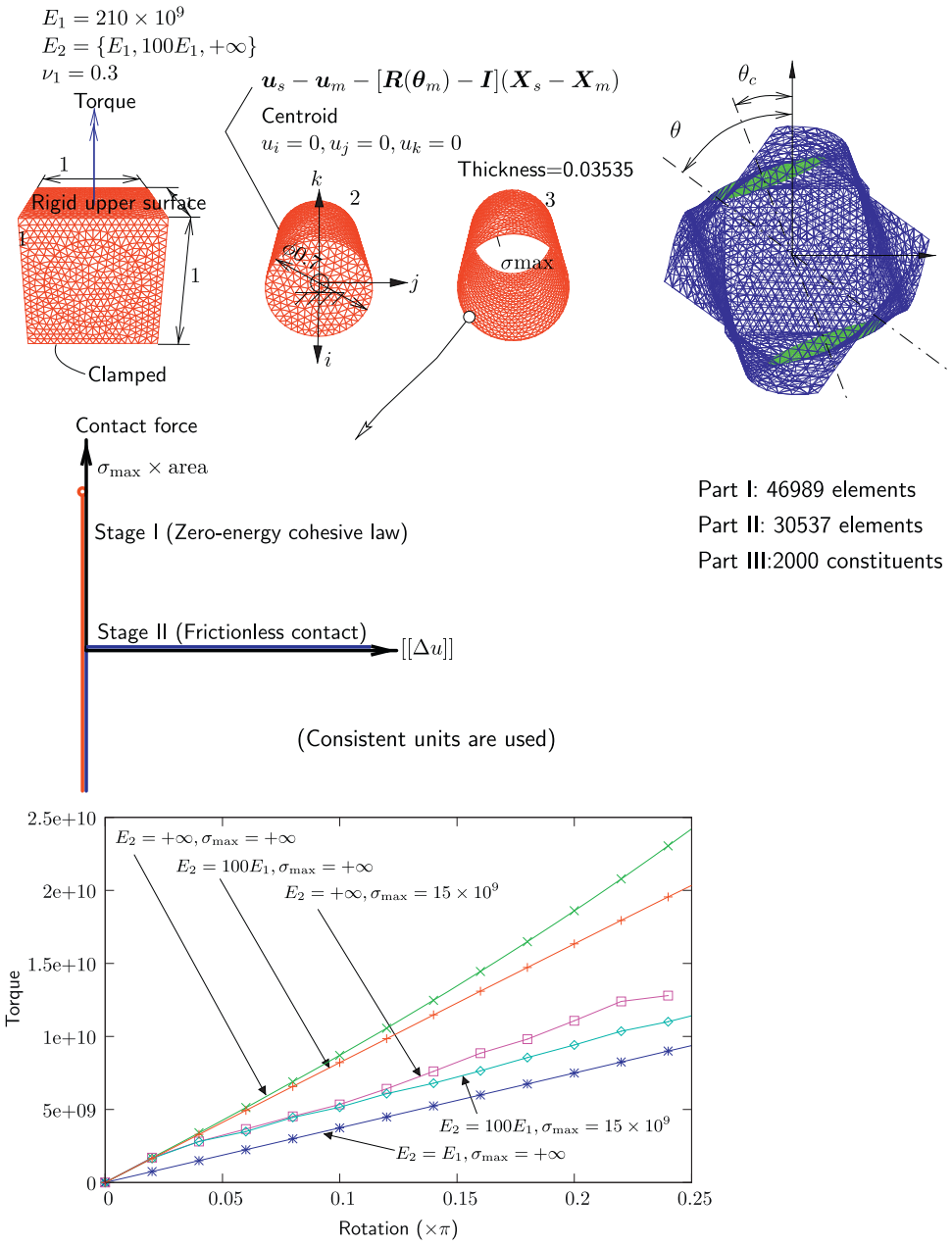

Fig. 13. Cylindrical inclusion torsion test: geometry, boundary conditions and material properties. Part I contains 46 989 elements, part II contains 30 537 elements and part III contains 2000 combined element/MPC components.

required to avoid repetitions. As can be observed in Fig. 6, there are no repetitions[13] in the processing of the sequences of DOFs. Multiplication of transformation matrices will benefit from this procedure. Non-slave nodes have unit $\boldsymbol{T}_\star$-coefficients whereas slave nodes' $\boldsymbol{T}_\star$-coefficients depend on the constraint imposed.

Algorithm 7. Conversion from **pold**, **dold** to **p**, **d** (collapse).

```
...
do i=1,n
  ieq=top(i)
  k=0
  do j=pold(ieq),pold(ieq+1)-1
    k=k+p(dold(j))
  enddo
  p(ieq)=k
enddo
mudlis(n,p) ! creates pointers
do i=1,n
  if(pold(i+1).eq.pold(i)+1)then
    if(dold(destp(i)).eq.i)d(p(i))=dold
    (pold(i))
  end if
enddo
do i=1,n
  ieq=top(i)
  l=0
  do j=pold(ieq),pold(ieq+1)-1
    jeq=dold(j)
    do k=p(jeq),p(jeq+1)-1
```

[13] To simplify the routines, we retain the multiplications by 1 for self-masters.

```
      l=l+1
      d(p(ieq)-1+l)=d(k)
    enddo
  enddo
enddo
...
```

4.5. Further details concerning the algorithm

The previous algorithms, in addition to a sparse solver that performs the required operations (linear solution only at this stage), do not ensure an efficient ordering of degrees-of-freedom. MPC transform the original elements and graph, and therefore the use of external node ordering codes is compromised. The same occurs with partitioning for parallel solution. An ordering subsequent to the calculation of the symbolic assembling is required. Either Approximate Minimum Degree (AMD) [3] variants or the modifications by Kumfert and Pothen of Sloan's algorithm [28] are well-known fill-in and profile minimizers, respectively. Although connectivities are introduced by DOF adjacency lists, and not nodes, since nodes would be useless in the presence of variable DOF types, MPC and the desired generality, we still group degrees-of-freedom which share the same adjacency list (besides each other). This grouping allows savings in the DOF ordering algorithm. Since degrees-of-freedom are distributed by clique, contraction of the list must be performed to remove non-surviving DOF. After this is performed, sorting of DOF is effected and the final sparse matrix is formed. This circumvents the permutation operation on the symbolic sparse matrix. After this stage is completed, the linear solution and recovery of slave DOF are performed.

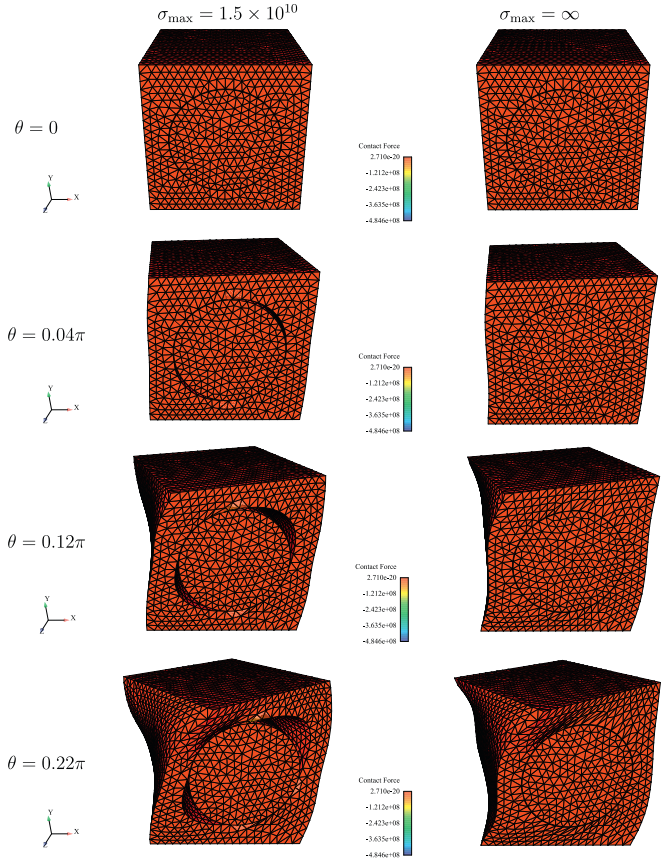

Fig. 14. Cube with rigid inclusion: sequence of deformed meshes and contact force in stage II. The left column shows the case where deactivation of node tie MPC is performed and a complementarity element is adopted. In the right column, tying MPC are retained.

The specific problem data, both element and MPC information, are communicated to a driver routine by use of two subroutines. A clique (a given generalized element) is inserted by invoking the overloaded routine **`store`**:

- **`store`**(**`iel`**,**`ndofiel`**,**`lnods`**,**`ltyps`**,**`efor`**,**`emat`**) where:
  - **`iel`** is the global element number.
  - **`ndofiel`** is the number of degrees-of-freedom of element iel.
  - **`lnods`** (size **`ndofiel`**) is the list of global nodes corresponding to each degree-of-freedom.
  - **`ltyps`** (size **`ndofiel`**) is the list of global types corresponding to each degree-of-freedom.
  - **`efor`** (size **`ndofiel`**) is the element "force" vector $\boldsymbol{f}$.
  - **`emat`** (size **`ndofiel`** $\times$ **`ndofiel`**) is the element "stiffness" matrix $\boldsymbol{K}$.

For example, a 3D MINI element (cf. [12]) which has four outer nodes and one inner node, we identify the degrees-of-freedom as: three displacement degrees-of-freedom (types 1, 2, and 3) and 1 pressure (here identified as type 7) degree-of-freedom per *outer* node and one internal node with three displacement degrees-of-freedom, we can set **`ltyps`** as: {**`1,2,3,7,1,2,3,7,1,2,3,7,1,2,3,7,1,2,3`**}. Multiple-point constraints are inserted by a similarly-named routine:

- **`store`**(**`mnods`**,**`mtyps`**,**`nmast`**,**`nnodm`**,**`ntypm`**,**`rhs`**,**`trm`**,**`trm2`**) where:
  - **`mnods`** is the global node of a slave degree-of-freedom.

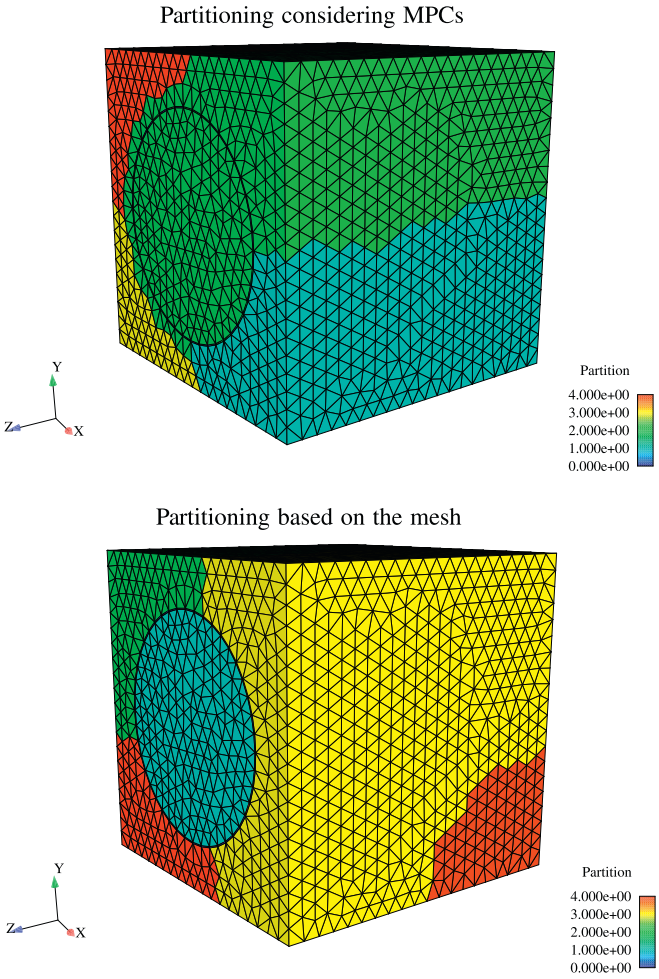

Fig. 15. Partitioning by METIS: effect of the MPC.

- **mtyps** is the global type of a slave degree-of-freedom.
- **nmast** is the number of master DOFs corresponding to **mnods**.
- **nnodm** (size **nmast**) is the list of global nodes corresponding to each degree-of-freedom.
- **ntypm** (size **nmast**) is the list of global types corresponding to each degree-of-freedom.
- **rhs** is the value $\boldsymbol{b}_\star$.
- **trm** is the matrix $\boldsymbol{T}_\star$.
- **trm2** is the derivative $\boldsymbol{T}'_\star$.

Contrary to the cliques, multiple-point constraints are subsequently sorted and therefore their numbers are not required.

5. Numerical tests

Several examples are herein computed in order to fully illustrate our approach in the following areas: (i) single rigid body dynamics, (ii) multi-body dynamics combining rigid and deformable parts, (iii) element implementation and (iv) computational fracture in 2D and shells with control equations.

5.1. Single rigid body dynamics: Lagrange and toy tops

Basic applications of our algorithm to rigid body dynamics (a verification example was shown in Section 2.3) are presented. Two tops (Lagrange and toy top) are tested and, for the Lagrange top, results are compared with the numerical solution of the exact problem statement. The Lagrange top has three degrees-of-freedom (three Euler angles) and the toy top has five degrees-of-freedom (three Euler angles and two displacement components at the contact tip). Rodrigues parameters are obtained from Euler angles. The same geometry for the top is adopted in both cases: two shallow cones joined at the bases. Initial angular velocity is imposed and a single rigid-body constraint is adopted. Fig. 7 summarizes the relevant data for this problem. For the Lagrange top, results are shown in Fig. 8 for the apex 1 trajectory and compared with the solution of the exact problem. Excellent results can be observed. Two time steps are used for the toy top (cf. Fig. 9) with good agreement between the results of the two time steps for both apices 1 and 2.

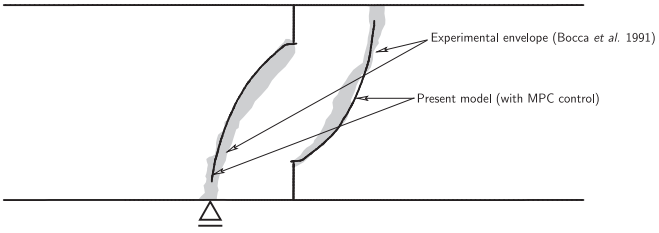

Fig. 17. Four-point bending of a concrete beam: crack paths compared with the envelope of experimental results by Bocca et al. [15].

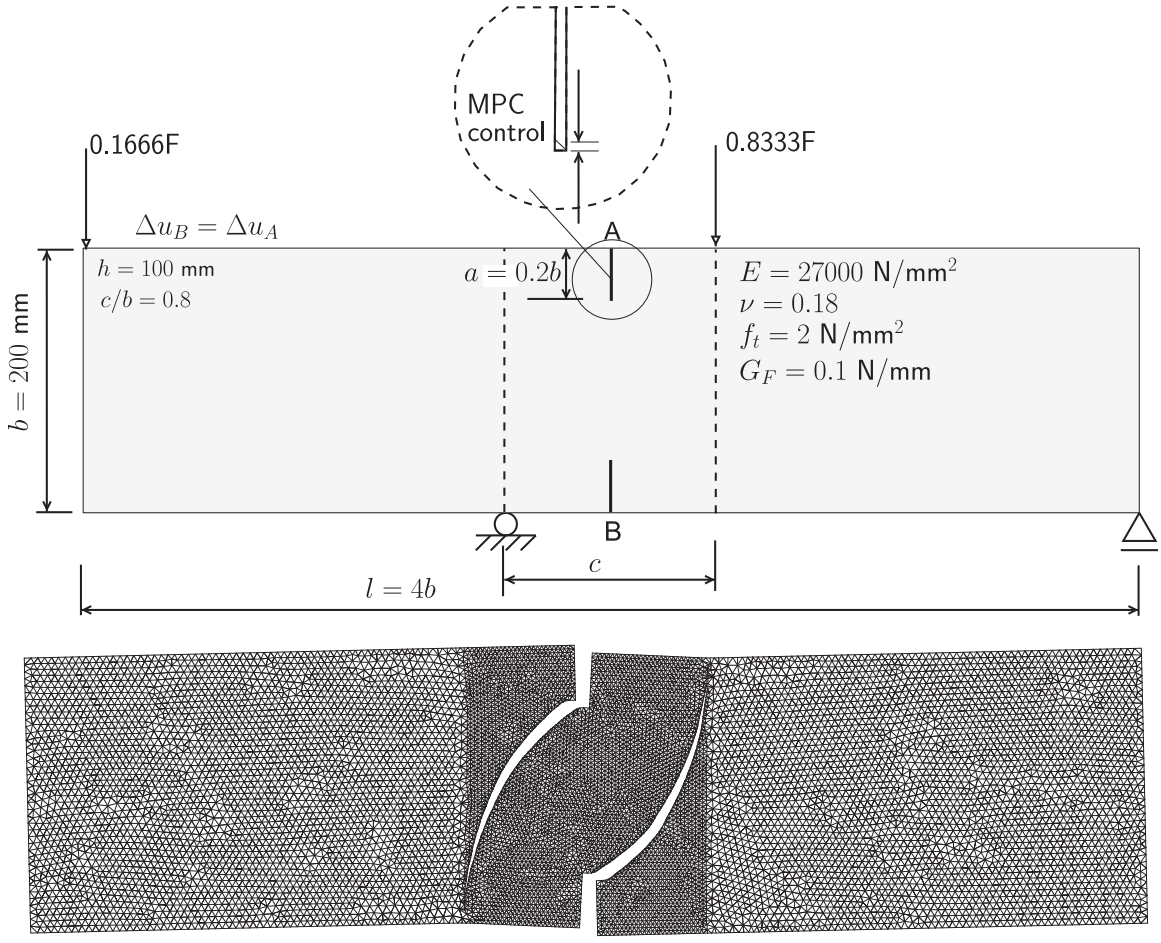

Fig. 16. Four-point bending of a concrete beam: geometry, boundary conditions, multiple-point constraints ($\Delta u_B = \Delta u_A$) and material properties. Also shown is the final deformed mesh $10\times$ magnified.

5.2. Multi-body dynamics: universal joint

The universal joint is analyzed with a combination of rigid body and deformable parts. With the proposed algorithm, each component can be either considered rigid or deformable according to the focus and interest of analysis. As Fig. 10 illustrates, we consider the central cross-shaft as either rigid or deformable and the remaining components as rigid. Imposed constant angular velocity at one of the shafts produces a transient torque response followed by a periodic torque response corresponding to the variable output angular velocity. This transient response was found to be significantly different between case I and case II. Fig. 11 shows this difference. Excellent agreement between the theoretical output angular velocity and the measured one can be seen in Fig. 12.

5.3. Torsion of a cube with a 3D rigid inclusion

A 3D cube with a rigid cylindrical inclusion is considered (see Fig. 13). The algorithm involves non-smooth elements which deactivate node-tying MPC during execution. Rigid body torsion is applied to the upper surface and the cube is clamped in its lower surface, see Fig. 13. Parts 1 and 2 shown in that figure are tied by a combination of elements and MPC represented as part 3 in the same figure. Two cases are tested. In the first case, parts 1 and 2 are exactly tied and remain that way. In the second case, decohesion occurs when the surface traction exceeds a stress threshold ($\sigma_{\max}$). In this situation, two stages occur: in stage I the gap remains constant and there is a limiting force of $\sigma_{\max} \times$ area shown in the figure. After the first violation of the limiting force, the problem becomes a frictionless contact one. A sequence showing the contact force in stage II is represented in Fig. 14.

An interesting aspect of this 3*D* problem is that it requires both topological sorting and slave node permutations to work properly. The partitioning in five regions by the software METIS shows (see Fig. 15) that the rigid cylinder is directly tied to one of the outer regions if MPC are included.

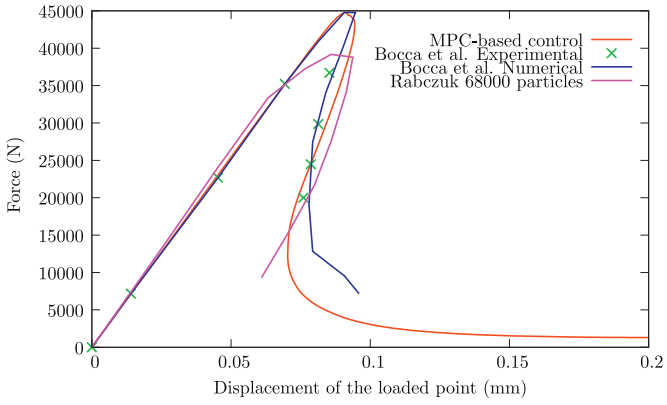

Fig. 18. Four-point bending of a concrete beam: load-displacement results, compared with the results of Bocca et al. [15] and the cracking particle method of Rabczuk and Belytschko [36] with their 68,000 particle analysis.

5.4. Quasi-static crack propagation control and geometrical elements

Quasi-static fracture processes are simulated using either displacement (or rotation) control or crack-opening-displacement control (cf. [8] where the ALE procedure is described). This is ideal for the use of MPC. Two problems are solved. The first problem is the one proposed by Bocca et al. [15], with relevant data shown in Fig. 16. Multiple-point constraints are used to force anti-symmetry conditions: the same mouth opening at the edge of notches A and B: $\Delta u_B = \Delta u_A$. Good agreement with the experimental crack paths is shown in Fig. 17. A comparison with the measurements of Bocca et al. [15] is shown in Fig. 18 along with the results by the cracking particle method of Rabczuk and Belytschko [36]. Note that geometrical elements are used to retain mesh quality after element splitting Fig. 19 shows the relevant data.

In the following fracture example, we test the control algorithm with the *quasi-brittle* shell fracture algorithm recently presented in CFRAC 2011 [11]. Relevant data for this problem are shown in Fig. 21. A Rankine-based criterion is adopted (coupled with isotropic damage—here represented by the void fraction variable f) as recently discussed in [9]. Note that constrained geometrical elements are used to retain mesh quality after element splitting (see [8,11] for further details). Two initial meshes are employed: one containing 5440 and another with 10 270 triangular elements. A sequence of deformed meshes of the shell is shown in Fig. 20 as well as the void fraction (f) contour plot. Very large displacements and rotations are observed with exceptional robustness. To confirm mesh-insensitivity, we show the control displacement/pressure results in Fig. 21 for both meshes.

6. Concluding remarks

In this work a new algorithm and corresponding code to process both additive and multiplicative components in an

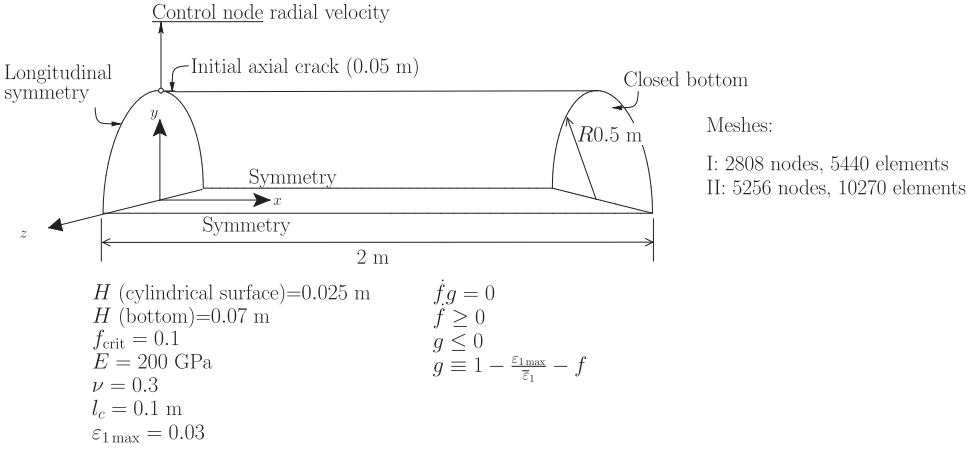

Fig. 19. Quasi-brittle fracture of a cylindrical shell: relevant data and discretization.

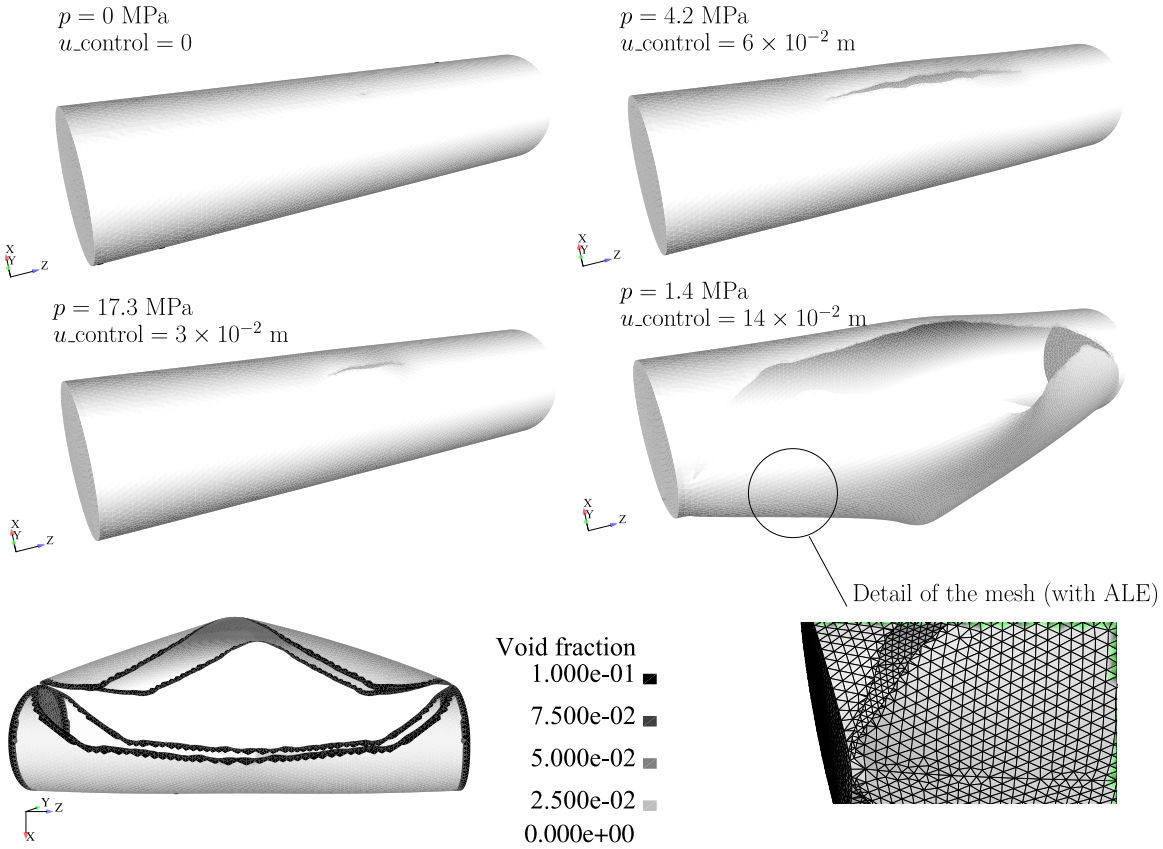

Fig. 20. Quasi-brittle fracture of a cylindrical shell: sequence of deformed meshes.

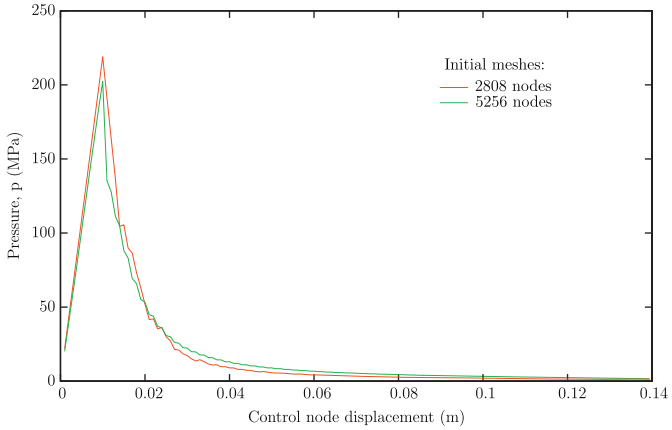

Fig. 21. Quasi-brittle fracture of a cylindrical shell: control displacement/pressure results.

implicit framework. Conditions for solvability were introduced and the two main problems (MPC sequential processing and reaction calculations) were identified as a path traversal in a directed acyclic graph. Processing by use of clique format was described in detail and advantages of this method were also discussed. The most important advantage is the direct access of sparse matrix components by means of clique addressing. Besides classical applications such as rigid (and multibody) dynamics, node links and continuation methods were also incorporated. Many other techniques and ad hoc features, previously considered unrelated to MPC, are now be included as multiplicative components. The main problems with MPC processing were solved, including the previously required ordering, which is no longer needed. A set of numerical examples showing a wide variety of applications was presented, making use of our publicly available software. Results made use of previously algorithms but these are now integrated in the framework.

7. Software availability

The basic clique and MPC framework is available on Google Code [7] under the LGPL license. It requires a Fortran 2003 compatible compiler.

Acknowledgments

The authors gratefully acknowledge financing from the "Fundaç ão para a Ciência e a Tecnologia" under the Project PTDC/EME-PME/108751 and the Program COMPETE FCOMP-01-0124-FEDER-010267.

Appendix A. Supplementary material

Supplementary data associated with this article can be found in the online version of http://dx.doi.org/10.1016/j.finel.2012.03.007.

References

[1] J.F. Abel, M.S. Shephard, An algorithm for multipoint constraints in finite element analysis, Int. J. Numer. Methods Eng. 14 (3) (1979) 464–467.
[2] M. Ainsworth, Essential boundary conditions and multi-point constraints in finite element analysis, Comput. Methods Appl. Mech. Eng. 190 (2001) 6323–6339.
[3] P.R. Amestoy, T. Enseeiht-Irit, T.A. Davis, I.S. Duff, Algorithm 837: AMD, an approximate minimum degree ordering algorithm, ACM Trans. Math. Software 30 (3) (2004) 381–388.

[4] F. Amirouche, Fundamentals of Multibody Dynamics Theory and Applications, Birkhäuser, 2006.
[5] S.S. Antman, Nonlinear Problems of Elasticity, second ed., Springer, 2005.
[6] S.S. Antman, R.S. Marlow, Material constraints, lagrange multipliers, and compatibility, Arch. Ration. Mech. Anal. 116 (1991) 257–299.
[7] P. Areias, Simplasmpc 〈http://code.google.com/p/simplasmpc/〉.
[8] P. Areias, D. Dias-da-Costa, J. Alfaiate, E. Júlio, Arbitrary bi-dimensional finite strain cohesive crack propagation, Comput. Mech. 45 (1) (2009) 61–75.
[9] P. Areias, J. Garção, E.B. Pires, J. Infante Barbosa, Exact corotational shell for finite strains and fracture, Comput. Mech. 48 (2011) 385–406.
[10] P. Areias, K. Matous, Finite element formulation for modeling nonlinear viscoelastic elastomers, Comput. Methods Appl. Mech. Eng. 197 (2008) 4702–4717.
[11] P. Areias, N. Van Goethem, E.B. Pires, Constrained ale-based discrete fracture in shells with quasi-brittle and ductile materials, in: CFRAC 2011 International Conference, Barcelona, Spain, June 2011, CIMNE.
[12] V.I. Arnold, Mathematical Methods of Classical Mechanics, Springer, 1989.
[13] K.-J. Bathe, Conserving energy and momentum in nonlinear dynamics: a simple implicit time integration scheme, Comput. Struct. 85 (2006) 437–445.
[14] T. Belytschko, W.K. Liu, B. Moran, Nonlinear Finite Elements for Continua and Structures, John Wiley & Sons, 2000.
[15] P. Bocca, A. Carpinteri, S. Valente, Mixed mode fracture of concrete, Int. J. Solids Struct. 27 (9) (1991) 1139–1153.
[16] R.H. Byrd, R.B. Schnabel, Continuity of the null space basis and constrained optimization, Math. Program. 35 (1986) 32–41.
[17] J.I. Curiskis, S. Valliappan, A solution algorithm for linear constraint equations in finite element analysis, Comput. Struct. 8 (1978) 117–124.
[18] T.A. Davis, Direct Methods for Sparse Linear Systems, SIAM, 2006.
[19] J. Dolbow, N. Moës, T. Belytschko, Modeling fracture in Mindlin–Reissner plates with the extended finite element method, Int. J. Solids Struct. 37 (2000) 7161–7183.
[20] I.S. Duff, A.M. Erisman, J.K. Reid, Direct Methods for Sparse Matrices, Clarendon Press, Oxford, 1986.
[21] J. Gibbons, An initial-algebra approach to directed acyclic graphs, CDMTCS, Centre for Discrete Mathematics and Theoretical Computer Science, 1995.
[22] N.I.M. Gould, M.E. Hribar, J. Nocedal, On the solution of equality constrained quadratic programming problems arising in optimization, SIAM J. Sci. Comput. 23 (4) (2001) 1376–1395.
[23] F.G. Gustavson, Two fast algorithms for sparse matrices: multiplication and permuted transposition, ACM Trans. Math. Software 4 (3) (1978) 250–269.
[24] N. Ian, M. Gould, On modified factorizations for large-scale linearly constrained optimization, SIAM J. Optim. 9 (4) (1999) 1041–1063.
[25] D. Jungnickel, Graphs, networks and algorithms, in: Algorithms and Computation in Mathematics, vol. 5, second ed., Springer, 2005.
[26] A. Klarbring, Models of Mechanics, Springer, 2006.
[27] D. Knuth, The Art of Computer Programming, third ed., vol. 3, Addison-Wesley, 1997.
[28] G. Kumfert, A. Pothen, Two improved algorithms for envelope and wavefront reduction, BIT 35 (1997) 1–32.
[29] G.M. Kuper, M.Y. Vardi, A new approach to database logic, in: PODS '84: Proceedings of the Third ACM SIGACT-SIGMOD Symposium on Principles of Database Systems, ACM, New York, NY, USA, 1984, pp. 86–96.
[30] J.M. Melenk, I. Babuska, The partition of unity finite element method: basic theory and applications, Comput. Methods Appl. Mech. Eng. 139 (1996) 289–314.
[31] J.L. Meriam, L.G. Kraige, Engineering Mechanics: Statics, fifth ed., John Wiley and Sons, 2002.
[32] N. Moës, J. Dolbow, T. Belytschko, A finite element method for crack growth without remeshing, Int. J. Numer. Methods Eng. 46 (1999) 131–150.
[33] S. Nash, A. Sofer, Linear and Nonlinear Programming, McGraw-Hill International Editions, 1996.
[34] P.E. Nikravesh, Computer-Aided Analysis of Mechanical Systems, Prentice-Hall, 1988.
[35] J. Nocedal, S. Wright, Numerical Optimization, Series Operations Research, second ed., Springer, 2006.
[36] T. Rabczuk, T. Belytschko, Cracking particles: a simplified meshfree method for arbitrary evolving cracks, Int. J. Numer. Methods Eng. 61 (2004) 2316–2343.
[37] W.C. Rheinboldt, Geometric notes on optimization with equality constraints, Appl. Math. Lett. 9 (3) (1996) 83–87.
[38] M.S. Shephard, Linear multipoint constraints applied via transformation as part of a direct stiffness assembly process, Int. J. Numer. Methods Eng. 20 (1984) 2107–2112.