SYMCOMP 2013
Lisbon, 9-10 September 2013

SYSTEMATIC SYMBOLIC GENERATION OF ADDITIVE AND MULTIPLICATIVE DISCRETE CONSTITUENTS

P. Areias[1*] and T. Rabczuk[2]

1: Physics Department University of Évora
Colégio Luís António Verney
Rua Romão Ramalho, 59
7002-554 Évora, Portugal
e-mail: pmaa@uevora.pt
2: Director of Institute of Structural Mechanics
Bauhaus-University
Weimar Marienstraße, 15
99423 Weimar, Germany
e-mail: timon.rabczuk@uni-weimar.de

Keywords: Symbolic Computation, Computational Framework, Finite Element Technology

Abstract. *The combination of symbolic computation with a adaptable many-to-many relationship framework allows the creation of varied additive (elements, loads, etc) and multiplicative (multiple-point constraints) constituents. From the framework perspective, both the algorithms and data structures and the architecture are tailored to accept the wide variety of typical constituents: Shell and continuum elements, rigid body constraints and essential boundary conditions. The symbolic computation is employed to generate, using Acegen, the specific calculations:*

- *Element technology weak forms and linearization, including mixed methods*
- *Kinematic routines*
- *Residual vectors and transformation matrices for multiple-point constraints*
- *Flow vectors and derivatives*
- *Dedicated small-sized operations, such as sensitivity analysis for linear algebra calculations*

The dependence relations between solution methods, algorithms and constituents are described. Fracture algorithms can be naturally casted in this framework. Data input makes use of the relationship framework.

1 Introduction

Discretizations of continuum engineering problems generate constituents belonging to two classes: additive constituents 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.* [10]. Details concerning the systematic creation and combination of new constituents, independently of the specific problem treated, has not been shown with details in the literature before. A systematization of the technical implementation of models of mechanics, in the sense of A. Klarbring [17] after discretization, is the aim of this work. This perspective is shared both by the governing equations, constraints and solution methods.

Essential boundary conditions are a good example of the effectiveness of multiplicative components used in many commercial and academic codes. The same applies to rod and shell parametrization: director inextensibility is imposed with multiplicative constituents (at the *continuum* level by coordinate transformation, see e.g. S.S. Antman [3, 2]).

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.

In the context of multibody dynamics, these methods are also known as *coordinate reduction methods* [1].

Belytschko *et al.* [10] 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. 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. This classification is illustrated in detail in Figure 1. Specific operations are now performed with Mathematica and Acegen, as pointed out in Figure 2. The framework allows the nearly automated creation of the input software.

2 Multiple-point constraints

2.1 Independent constraints

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:

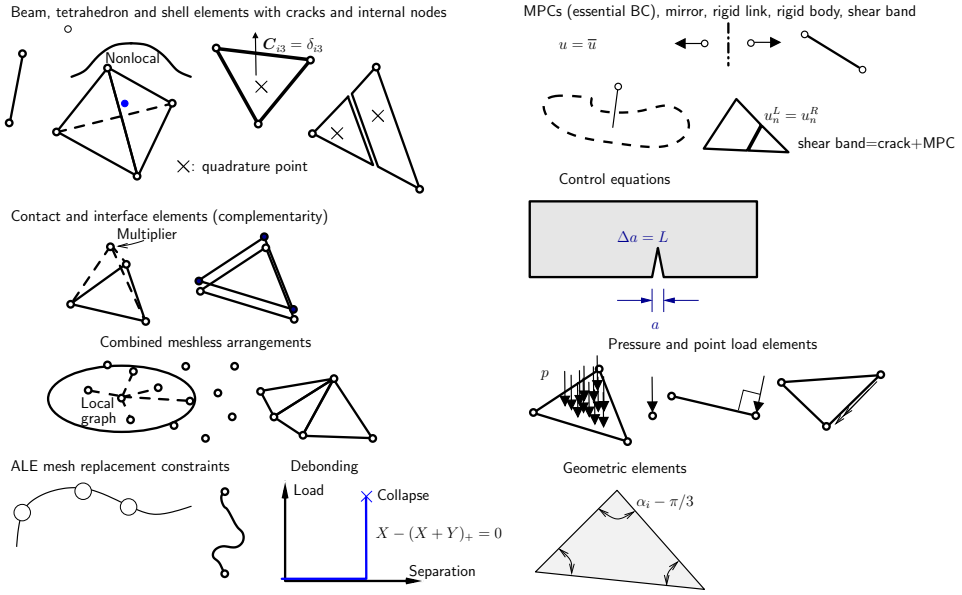


Figure 1: Classification of common discretization components as either additive (elements) or multiplicative (MPC).

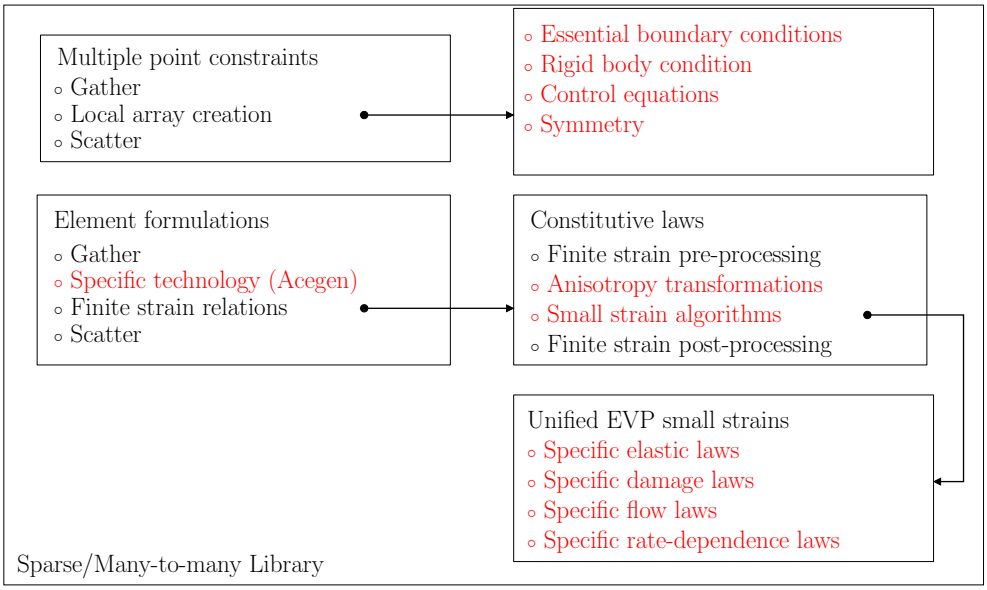


Figure 2: Typical discrete constituent operations. In red are shown the operations performed symbolically.

$$\boldsymbol{f}(\boldsymbol{a}) = \boldsymbol{0} \tag{1}$$
$$\boldsymbol{g}(\boldsymbol{a}) = \boldsymbol{0} \tag{2}$$

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 [23], p. 84):

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

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 equation (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. [10]). When using this virtual degree-of-freedom array, we can apply the Newton method to the system (1) and obtain,

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

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 nonlinear 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[1]:

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

[1]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 [21] concerning the definition of the required permutation matrices

or, if $\boldsymbol{T} = -\boldsymbol{g}_s'^{-1}\boldsymbol{g}_{\boldsymbol{r}}'$ 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. [19], 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

$$\left\{ \begin{array}{c} \mathrm{d}\boldsymbol{a}_s \\ \mathrm{d}\boldsymbol{a}_{\boldsymbol{r}} \end{array} \right\} = \overbrace{\underbrace{\left[\begin{array}{c} \boldsymbol{T} \\ I_{(n-m)\times(n-m)} \end{array} \right]}_{\text{null-space term}} \mathrm{d}\boldsymbol{a}_r}^{\boldsymbol{T}_\star} + \overbrace{\underbrace{\left\{ \begin{array}{c} \boldsymbol{b}_s \\ \boldsymbol{0} \end{array} \right\}}_{\text{corrective term}}}^{\boldsymbol{b}_\star} \tag{7}$$

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:

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

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:

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

where $\boldsymbol{K}_\star$ is the reduced stiffness matrix and $\boldsymbol{f}_\star$ is the reduced force vector.

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. [13]) is considered we can simply write:

$$\boldsymbol{K} = \sum_{i=1}^{n_e} \boldsymbol{K}_i^e \tag{13}$$

$$\boldsymbol{f}^T \boldsymbol{c} \boldsymbol{g}'' = \sum_{j=1}^{m} \left[\left(\sum_{k=1}^{n_e} \boldsymbol{f}_k^e \right)^T \left(\boldsymbol{c}_j \boldsymbol{g}_j'' \right) \right] \tag{14}$$

where $\boldsymbol{K}_i^e$ is the i^{th} element stiffness matrix, $\boldsymbol{f}_k^e$ is the k^{th} element force and $\boldsymbol{c}_j \boldsymbol{g}_j''$ is obtained from the j^{th} 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 (13) and (14) since the degree-of-freedom destinations of $\boldsymbol{g}''$ do 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, equation (12) can be written as:

$$\boldsymbol{T}_\star^T \left(\sum_{i=1}^{n_e} \boldsymbol{K}_i^e \right) \boldsymbol{T}_\star + \boldsymbol{T}_\star^T \left\{ - \sum_{j=1}^{m} \left[\left(\sum_{k=1}^{n_e} \boldsymbol{f}_k^e \right)^T \left(\boldsymbol{c}_j \boldsymbol{g}_j'' \right) \right] \right\} \boldsymbol{T}_\star \mathrm{d}\boldsymbol{a}_r = \tag{15}$$

$$- \boldsymbol{T}_\star^T \left(\sum_{j=1}^{n_e} \boldsymbol{f}_j^e \right) - \boldsymbol{T}_\star^T \left(\sum_{l=1}^{n_e} \boldsymbol{K}_j^e \right) \boldsymbol{b}_\star \tag{16}$$

The format of equation (15) 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 formation of the global stiffness matrix $\boldsymbol{K}_\star$. In terms of condition number of the reduced stiffness matrix, it can be shown that:

$$\mathrm{cond}(\boldsymbol{K}_\star) \leq \mathrm{cond}(\boldsymbol{K} - \boldsymbol{f}^T \boldsymbol{c} \boldsymbol{g}'') \underbrace{\mathrm{cond}(\boldsymbol{T}^T \boldsymbol{T} + \boldsymbol{I})}_{\kappa_T} \tag{17}$$

An application of (17) 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 parametrizations:

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

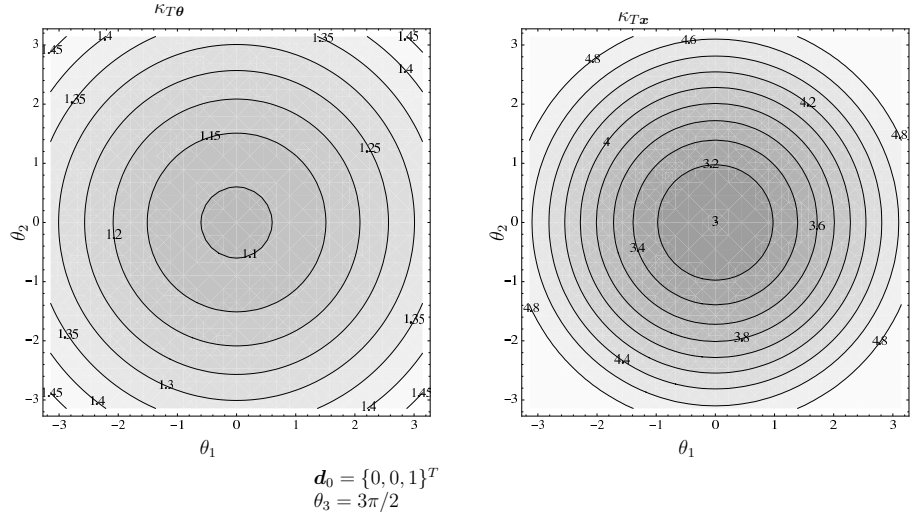


Figure 3: Condition numbers for the two director parametrizations as a function of θ_1 and θ_2.

Let $\boldsymbol{d}$ represent a director in the deformed configuration and $\boldsymbol{d}_0$ the corresponding director in the undeformed configuration. Using either of the parametrizations, 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[2]:

$$\boldsymbol{d}_\theta = \boldsymbol{d}_0 + \frac{\sin\|\boldsymbol{\theta}\|}{\|\boldsymbol{\theta}\|}\boldsymbol{\theta}\times\boldsymbol{d}_0 + 2\frac{\sin^2\frac{\|\boldsymbol{\theta}\|}{2}}{\|\boldsymbol{\theta}\|^2}\boldsymbol{\theta}\times(\boldsymbol{\theta}\times\boldsymbol{d}_0) \tag{18}$$

$$\boldsymbol{d}_x = \begin{bmatrix} -1+\frac{2(1+x_1^2)}{1+x_1^2+x_2^2+x_3^2} & \frac{2(x_1x_2-x_3)}{1+x_1^2+x_2^2+x_3^2} & \frac{2(x_1x_3+x_2)}{1+x_1^2+x_2^2+x_3^2} \\ \frac{2(x_1x_2+x_3)}{1+x_1^2+x_2^2+x_3^2} & -1+\frac{2(1+x_2^2)}{1+x_1^2+x_2^2+x_3^2} & \frac{2(x_2x_3-x_1)}{1+x_1^2+x_2^2+x_3^2} \\ \frac{2(x_1x_3-x_2)}{1+x_1^2+x_2^2+x_3^2} & \frac{2(x_2x_3+x_1)}{1+x_1^2+x_2^2+x_3^2} & -1+\frac{2(1+x_3^2)}{1+x_1^2+x_2^2+x_3^2} \end{bmatrix}\boldsymbol{d}_0 \tag{19}$$

2.2 Inertial forces

Many studies in multibody dynamics are focused in constraint imposition and time integration (see, e.g. Nikravesh [20]). 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 $\ddot{\boldsymbol{a}}_n$:

$$\ddot{\boldsymbol{a}}_{n+1} = \ddot{\boldsymbol{a}}\left(\boldsymbol{a}_n, \boldsymbol{a}_{n+1}, \dot{\boldsymbol{a}}_n, \ddot{\boldsymbol{a}}_n\right) \tag{20}$$

The total force vector including inertial forces is given by:

[2] $\boldsymbol{x} = \tan\left(\frac{\|\boldsymbol{\theta}\|}{2}\right)\frac{\boldsymbol{\theta}}{\|\boldsymbol{\theta}\|}$

$$\boldsymbol{f}_{\rm dyn} = \boldsymbol{f} + \boldsymbol{M}\ddot{\boldsymbol{a}}_{n+1} \tag{21}$$

where $\boldsymbol{M}$ is an appropriate mass matrix (cf. [10]). We can therefore write the unconstrained solution scheme as:

$$\delta\boldsymbol{a}^T \underbrace{\left(\boldsymbol{K} + \boldsymbol{M}\frac{\partial \ddot{\boldsymbol{a}}}{\partial \boldsymbol{a}_{n+1}}\right)}_{\boldsymbol{K}_{\rm dyn}} {\rm d}\boldsymbol{a} = -\delta\boldsymbol{a}^T \underbrace{(\boldsymbol{f} + \boldsymbol{M}\ddot{\boldsymbol{a}}_{n+1})}_{\boldsymbol{f}_{\rm dyn}} \tag{22}$$

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 $\boldsymbol{a}_{n+\frac{1}{2}}$).
- $\boldsymbol{f}_{\rm dyn}$ replaces $\boldsymbol{f}$ in (12).
- $\boldsymbol{K}_{\rm dyn}$ 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 [9]). In that case we specify $\ddot{\boldsymbol{a}}$ as:

$$\ddot{\boldsymbol{a}} = \begin{cases} \frac{16\left(\boldsymbol{a}_{n+\frac{1}{2}} - \boldsymbol{a}_n\right)}{\Delta t^2} - \frac{8\dot{\boldsymbol{a}}_n}{\Delta t} - \ddot{\boldsymbol{a}}_n & , \quad h_s = 1 \quad (\ddot{\boldsymbol{a}} \equiv \ddot{\boldsymbol{a}}_{n+\frac{1}{2}}) \\ \frac{1}{\Delta t}\dot{\boldsymbol{a}}_n - \frac{4}{\Delta t}\dot{\boldsymbol{a}}_{n+\frac{1}{2}} + \frac{3}{\Delta t}\dot{\boldsymbol{a}}_{n+1} & , \quad h_s = 2 \quad (\ddot{\boldsymbol{a}} \equiv \ddot{\boldsymbol{a}}_{n+1}) \end{cases} \tag{23}$$

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 (23) 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. [8]) are included. An simple application is the pendulum which we can of course integrated in closed form. Figure 4 shows an application with two fixed time step increments (Δt=1% and Δt=5% of the linear period T_l) for $E = \infty$ (with the rigid body multiple-point constraint). Exceptional robustness and accuracy are verified.

2.3 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

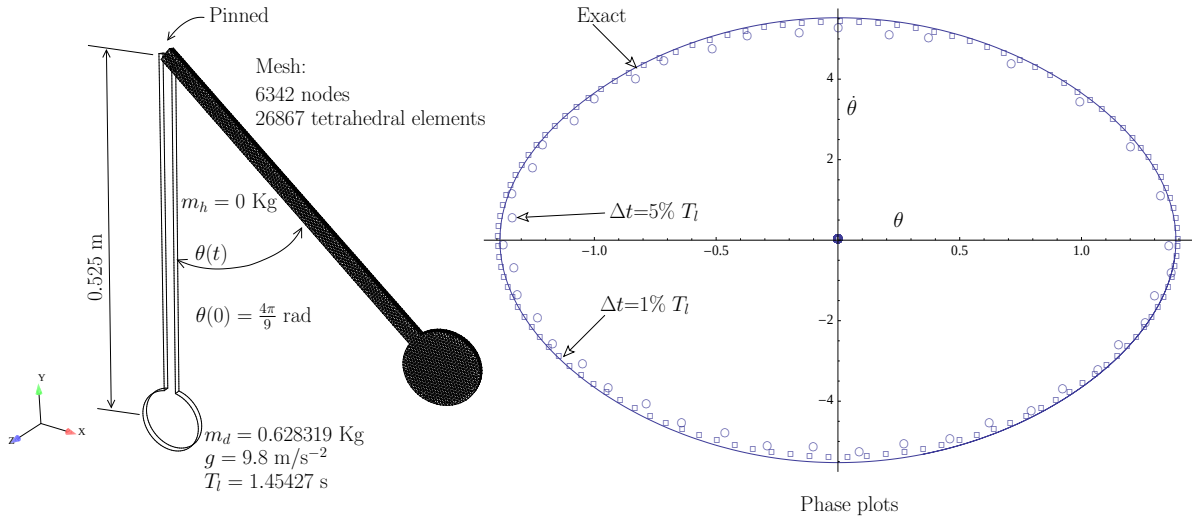


Figure 4: Large amplitude pendulum (rigid-body constraint) integrated with two time-steps (1% and 5% of the linear period).

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[3]. The generalization of the slave update formula for m *interconnected* equality constraints is presented, after the preliminary step of topological ordering, as:

$$\boldsymbol{T}_\star^m = \prod_{l=m}^{1} \boldsymbol{T}_{\star l} \tag{24}$$

$$\boldsymbol{b}_\star^m = \sum_{l=m}^{1} \left[\left(\prod_{p=m}^{l} \boldsymbol{T}_{\star p} \right) \boldsymbol{b}_{\star l} \right] \tag{25}$$

Many MPC applications only moderately increase the fill-in in decomposition if an efficient post-ordering is performed, as the rigid link of Figure 5 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 pre-process 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

[3]In the sense of the original unconstrained system

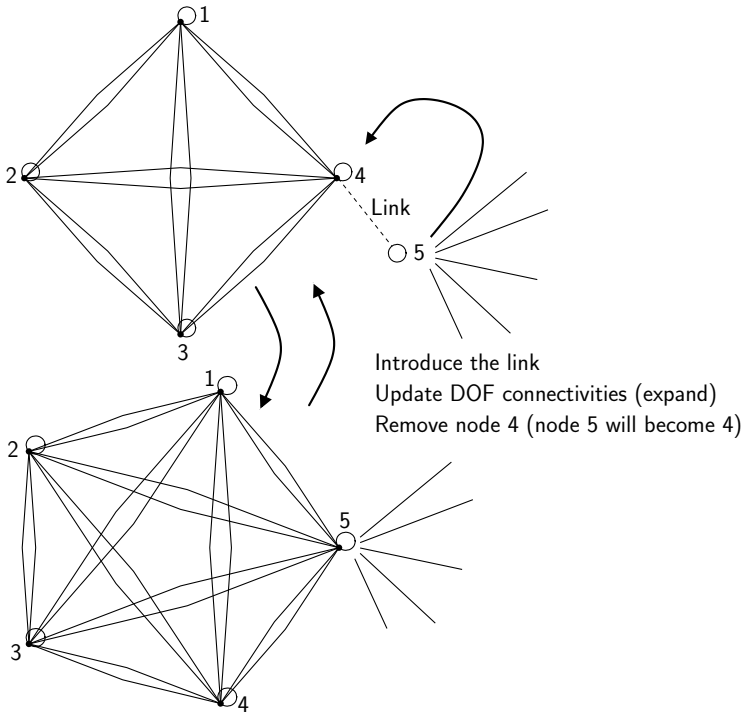


Figure 5: Example of a local graph update from the introduction of a rigid link. The slave node (here 4) is removed from the graph.

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. [12]).

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 F. G. Gustavson [15] explores this aspect by means of the transposition algorithm). 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.

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: `DOF 1` is related to local number `5`, `DOF 2` 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).

4 Assembling

4.1 Symbolic and numeric assembling

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 [4].

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 [4] for further details.

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.* [14].

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
   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
```

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
```

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
...
```

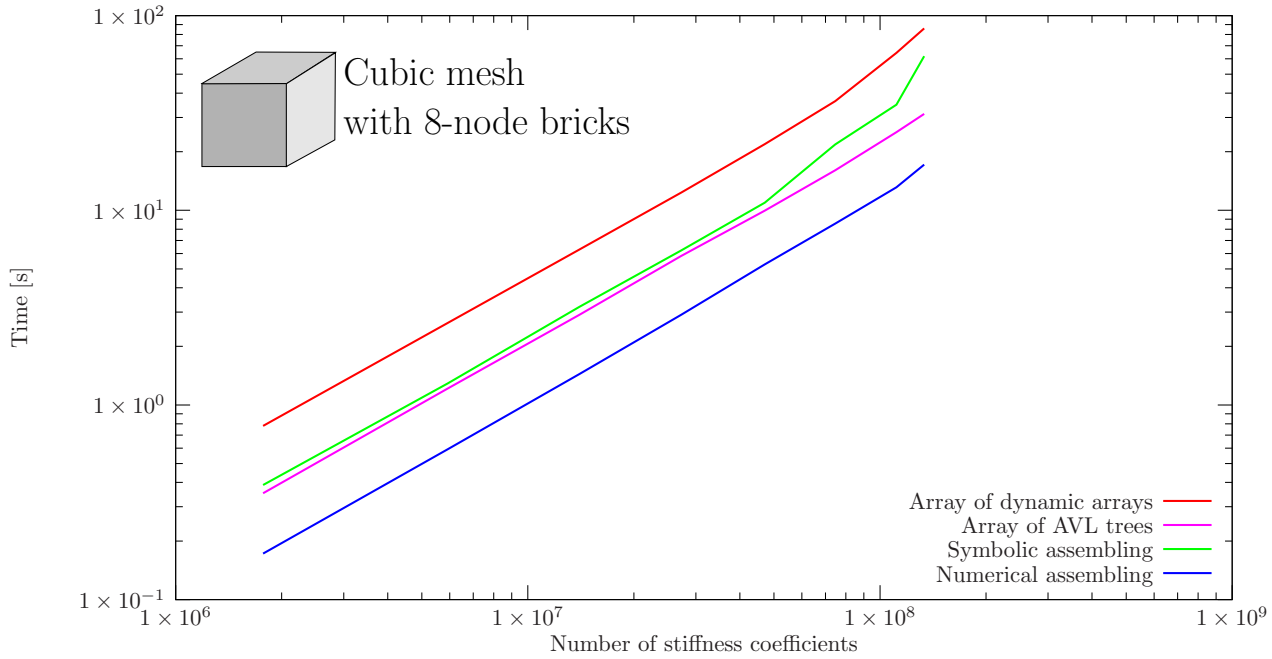


Figure 6: Assembling times as a function of number of stiffness matrix coefficients. Machine: Apple MacBook Pro 2.66 GHz Intel Core i7, 8 Gb RAM. Compiler: gfortran (GCC 4.5.0) with -O3 option.

- Array of AVL trees [18], allowing binary search but requiring branch balancing.
- Our clique/adjacency structure, allowing direct access with symbolic pre-processing required.

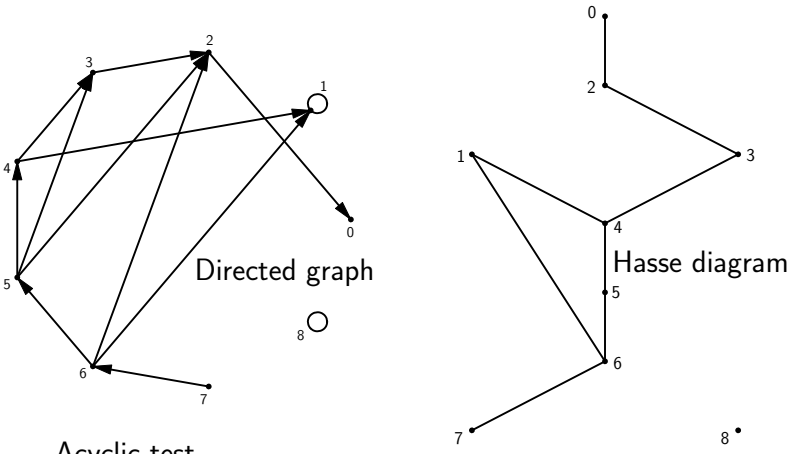
Figure 6 shows the results. Some conclusions are:

- 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 degree-of-freedom. Slave degrees-of-freedom can have any non-negative en_i (for example Dirichlet conditions result in a zero



	Unroll of nested DOFs	Collapse (and sum)
$1:1$	$1:\mathbf{1}$	$1:1$
$2:0$	$2:\mathbf{0}$	$2:0$
$3:2$	$3:2,\mathbf{0}$	$3:0$
$4:1,3$	$4:1,\mathbf{1}\|3,\mathbf{0}$	$4:1,0$
$5:3,4,2$	$5:3,\mathbf{0}\|4,\mathbf{1}\|4,\mathbf{0}\|2,\mathbf{0}$	$5:0,1$
$6:2,1,5$	$6:2,\mathbf{0}\|1,\mathbf{1}\|5,\mathbf{0}\|5,\mathbf{1}\|5,\mathbf{0}\|5,\mathbf{0}$	$6:0,1$
$7:6$	$7:6,\mathbf{0}\|6,\mathbf{1}\|6,\mathbf{0}\|6,\mathbf{1}\|6,\mathbf{0}\|6,\mathbf{0}$	$7:0,1$
$8:8$	$8:\mathbf{8}$	$8:8$

Degree-of-freedom:list of masters

Figure 7: Specific DOF distribution: directed graph and Hasse diagram. Collapse of DOF destinations.

en_i). Consider the DOF arrangement of Figure 7where the directed graph and the Hasse diagram for this arrangement are shown. Traversing from the top the Hasse diagram we obtain the 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 [16]. 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 Figure 7, 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 [16] 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.

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 [16]. Algorithm 6 shows this operation. A solvable problem results in a Direct Acyclic Graph (DAG). The scheduling of DOF processing is required to avoid repetitions. As can be observed in Figure 7, there are no repetitions[4] 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.

4.5 Further details concerning the algorithm

The specific problem data, both element and MPC information is 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. [8]) which has 4 outer nodes and 1 inner node, we identify the degrees-of-freedom as: 3 displacement degrees-of-freedom (types 1,2,3) and 1 pressure (here identified as type 7) degree-of-freedom per *outer* node and one internal node with 3 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.
  - `mtyps` is the global type of a slave degree-of-freedom.
  - `nmast` is the number of master DOFs corresponding to `mnods`.

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

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
 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
```

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

- `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.

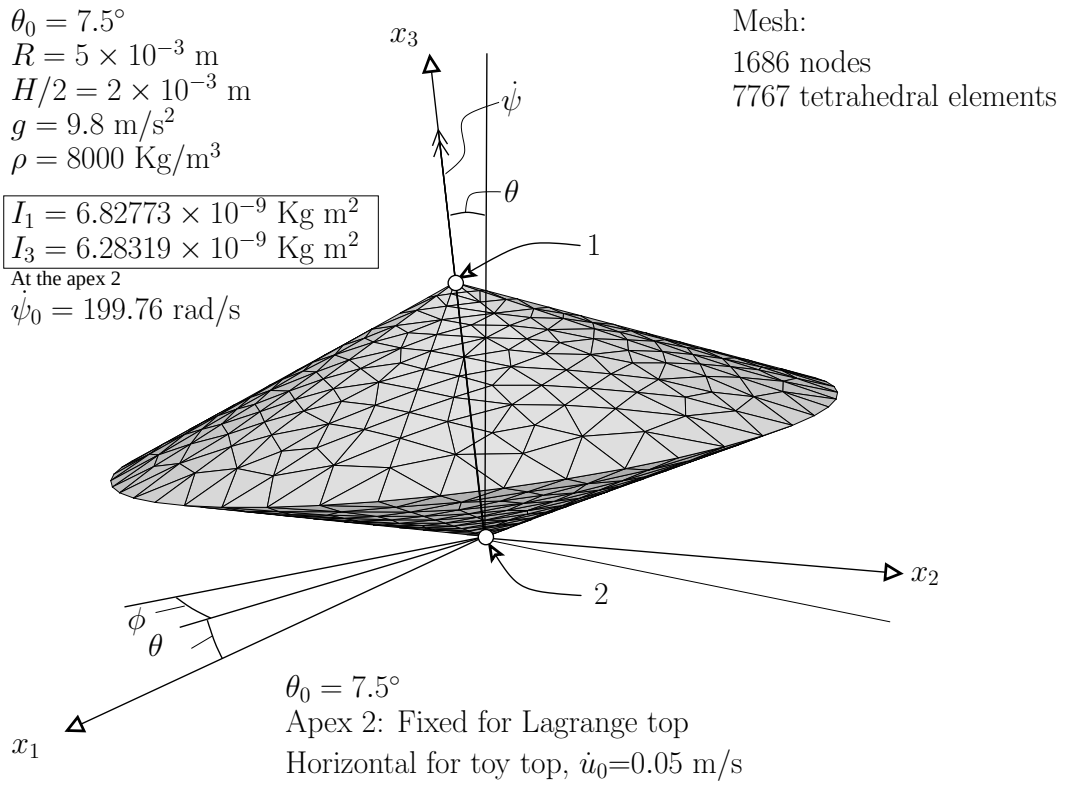


Figure 8: Lagrange and toy tops: relevant geometrical data and mass properties.

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 subsection 2.2) 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. Figure 8 summarizes the relevant data for this problem. For the Lagrange top, results are shown in Figure (9) 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. Figure 10) with good agreement between the results of the two time steps for both apices 1 and 2.

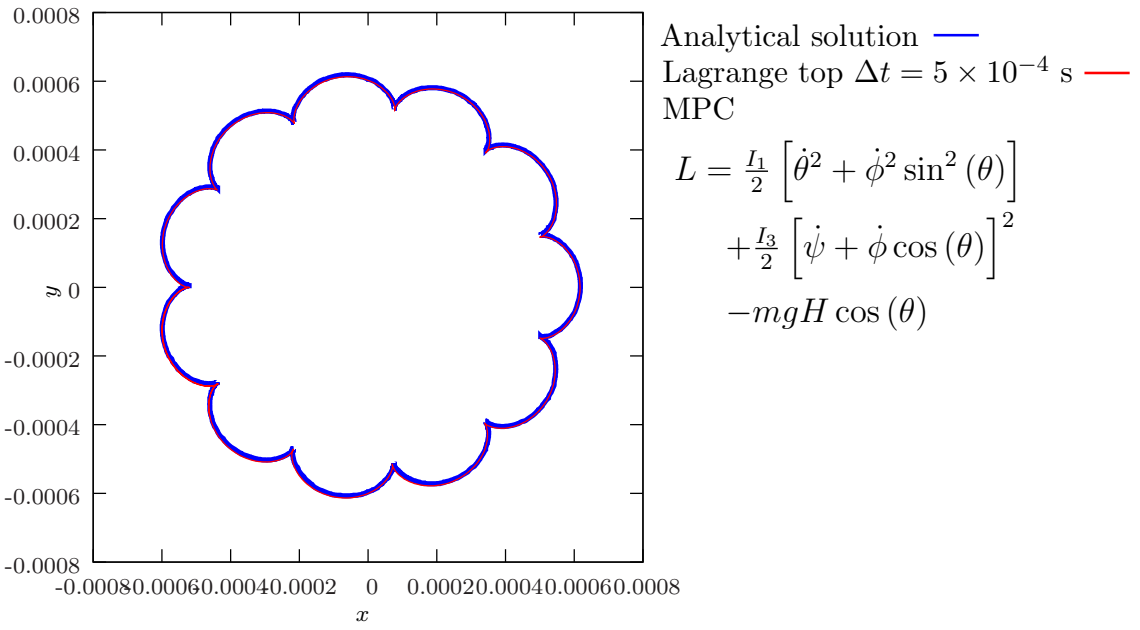


Figure 9: Apex 1 trajectory for the Lagrange top.

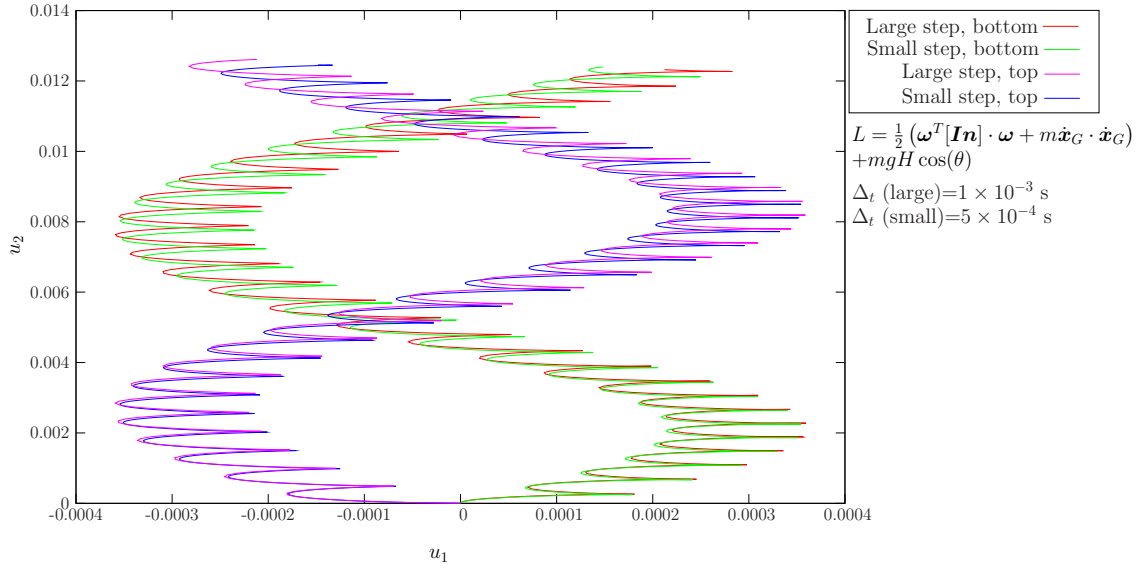


Figure 10: Apex 1 and 2 displacement components for the toy top.

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 Figure 11 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. Figure 12 shows this difference. Excellent agreement between the theoretical output angular velocity and the measured one can be seen in Figure 13.

5.3 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. [5] 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.* [11], with relevant data shown in Figure 14. 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 Figure 15. A comparison with the measurements of Bocca *et al.* [11] is shown in Figure 16 along with the results by the cracking particle method of Rabczuk and Belytschko [22]. Note that geometrical elements are used to retain mesh quality after element splitting.

In the following fracture example, we test the control algorithm with the *quasi-brittle* shell fracture algorithm recently presented in CFRAC 2011 [7]. Relevant data for this problem is shown in Figure 19. A Rankine-based criterion is adopted (coupled with isotropic damage - here represented by the void fraction variable f) as recently discussed in [6]. Note that constrained geometrical elements are used to retain mesh quality after element splitting (see [5, 7] for further details). Two initial meshes are employed: one containing 5440 and another with 10270 triangular elements. A sequence of deformed meshes of the shell is shown in Figure 18 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 Figure 19 for both meshes.

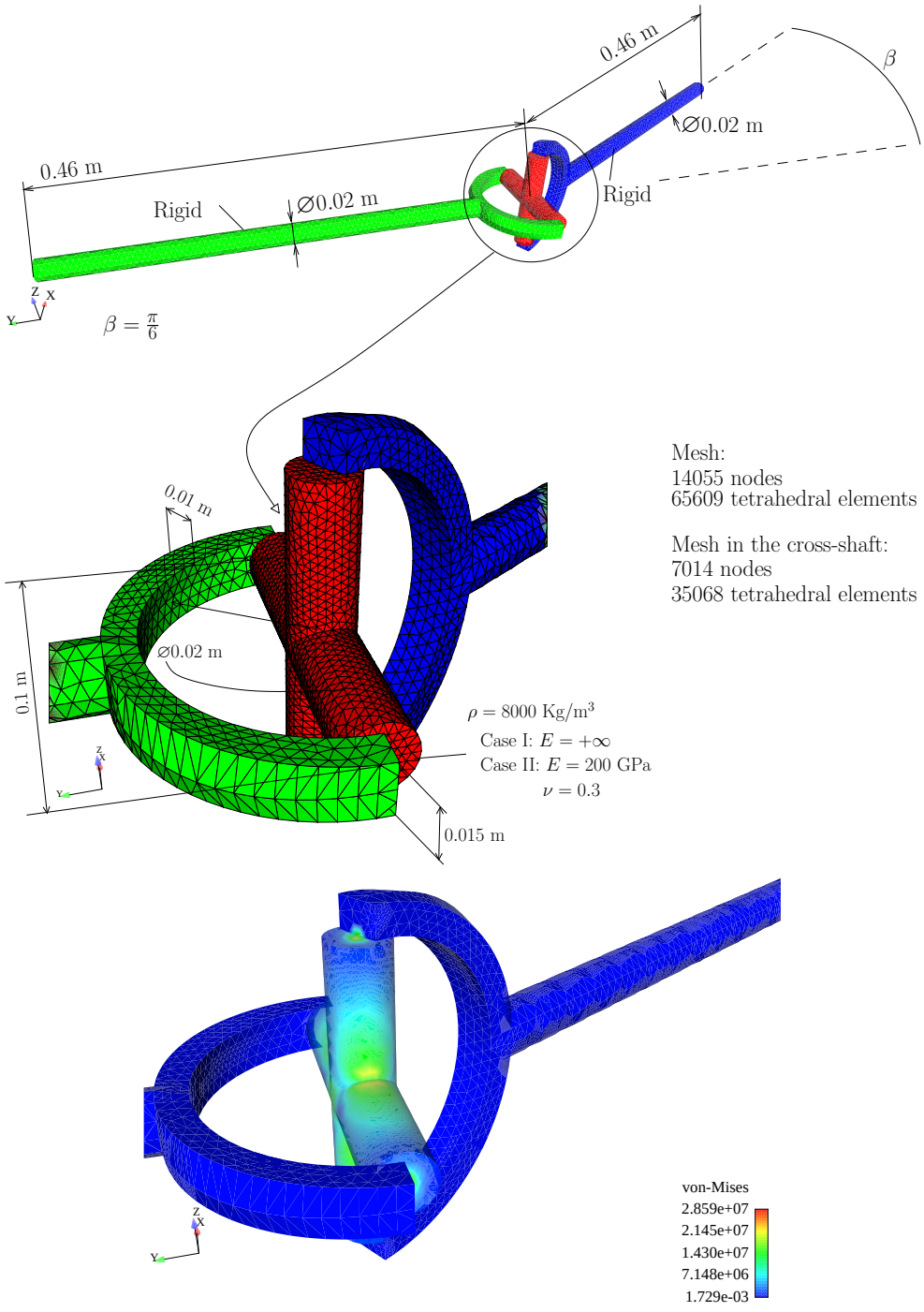


Figure 11: Universal joint: geometry and relevant problem data. The von-Mises equivalent stress at the cross-shaft (the only deformable part) is shown.

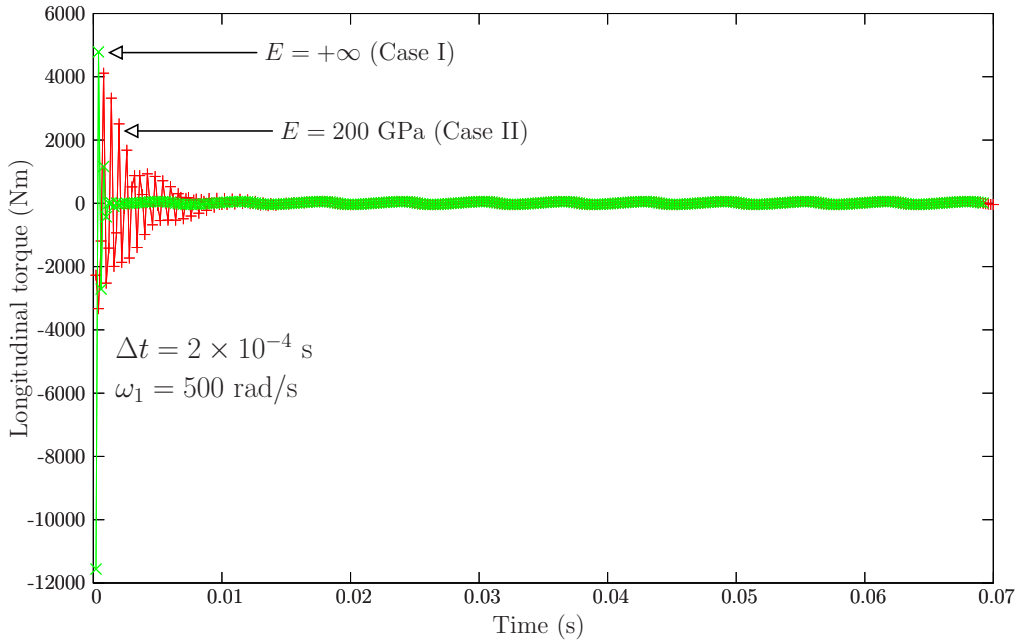


Figure 12: Universal joint: response to constant angular velocity at the input shaft.

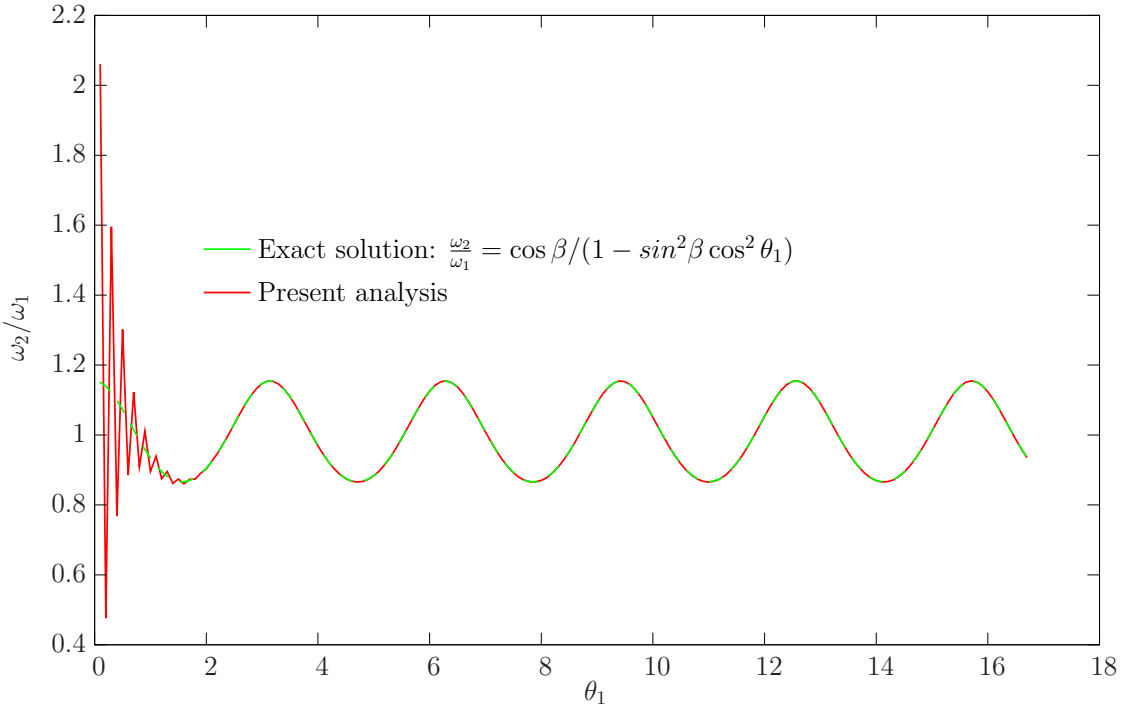


Figure 13: Universal joint: ratio between angular velocities, comparison with exact solution.

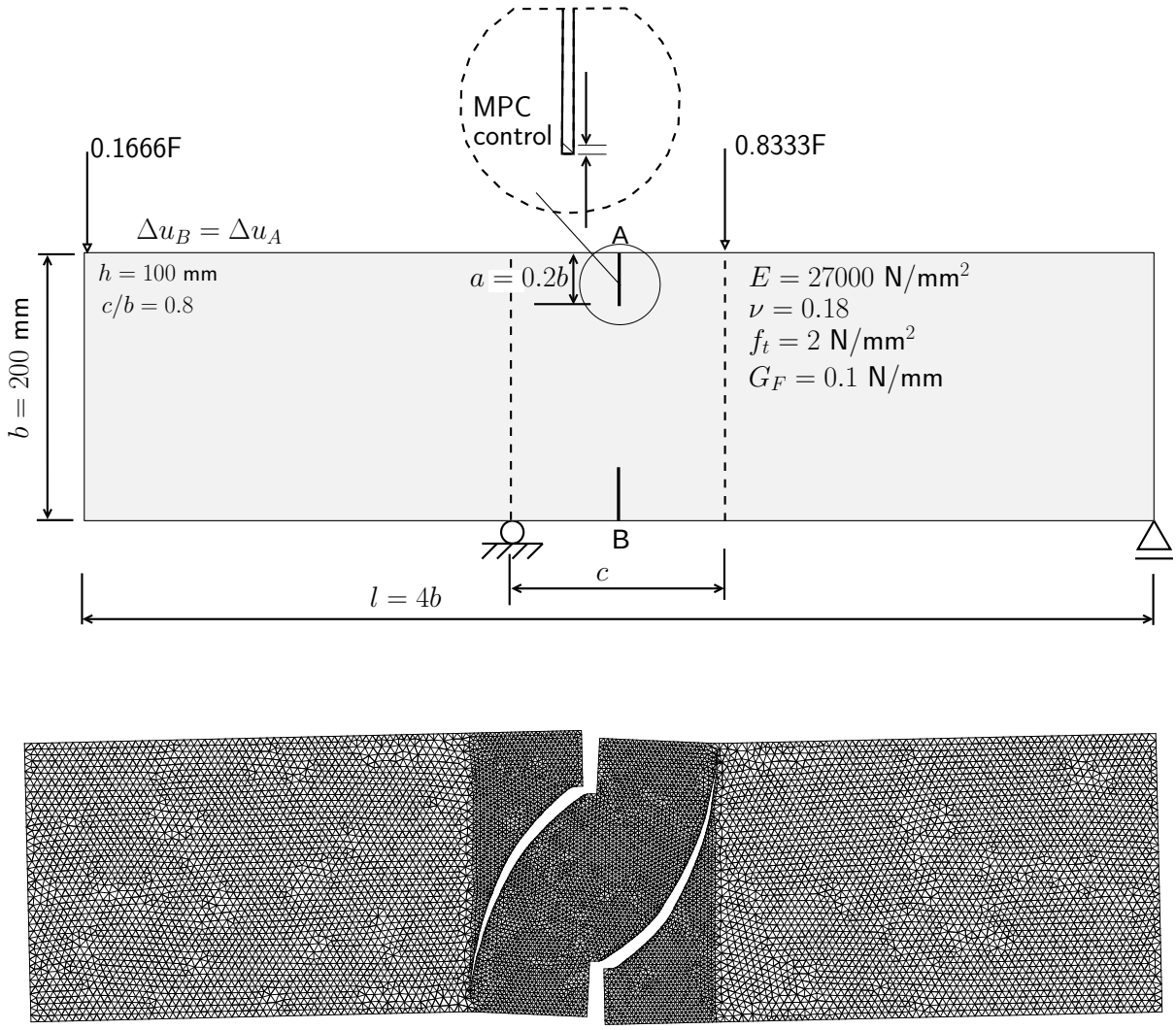


Figure 14: Four-point bending of a concrete beam: geometry, boundary conditions, multiple-point constraints ($\Delta u_B = \Delta u_A$) and material properties. Also shown is the final deformed mesh 10× magnified.

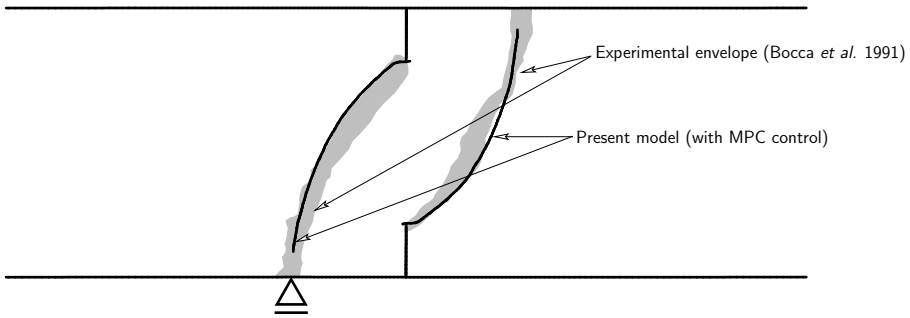


Figure 15: Four-point bending of a concrete beam: crack paths compared with the envelope of experimental results by Bocca, Carpintieri and Valente [11].

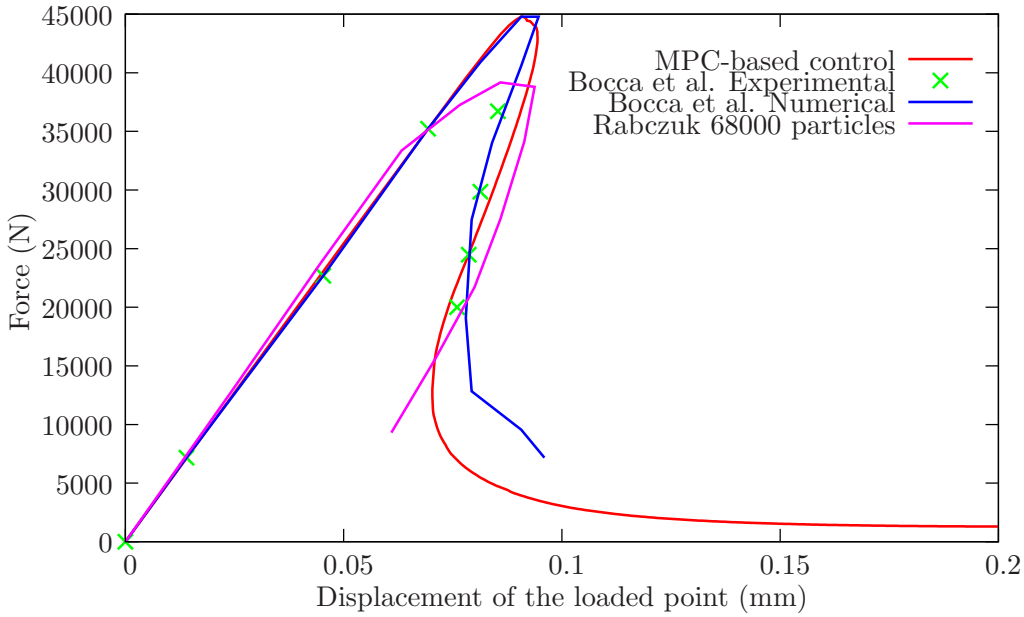


Figure 16: Four-point bending of a concrete beam: load-displacement results, compared with the results of Bocca, Carpintieri and Valente [11] and the cracking particle method of Rabczuk and Belytschko [22] with their 68000 particle analysis.

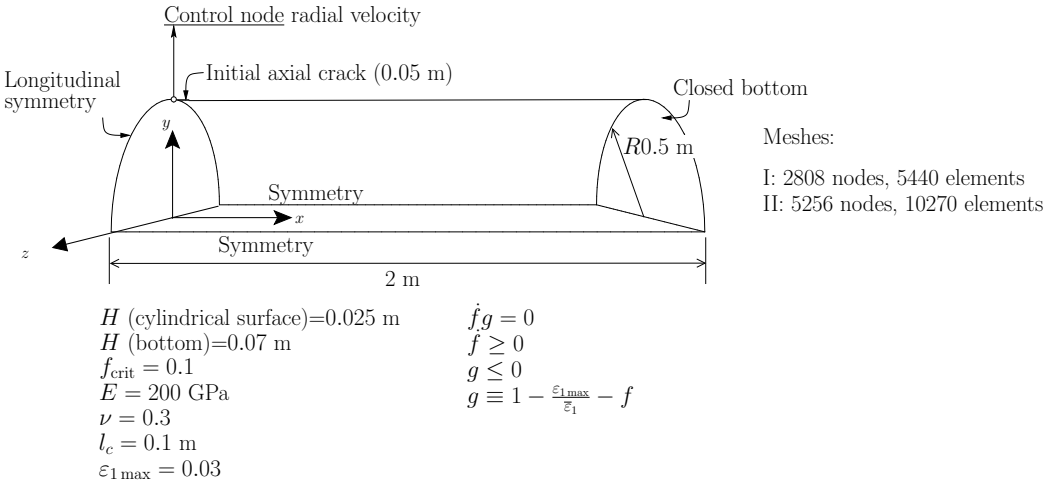


Figure 17: Quasi-brittle fracture of a cylindrical shell: relevant data and discretization.

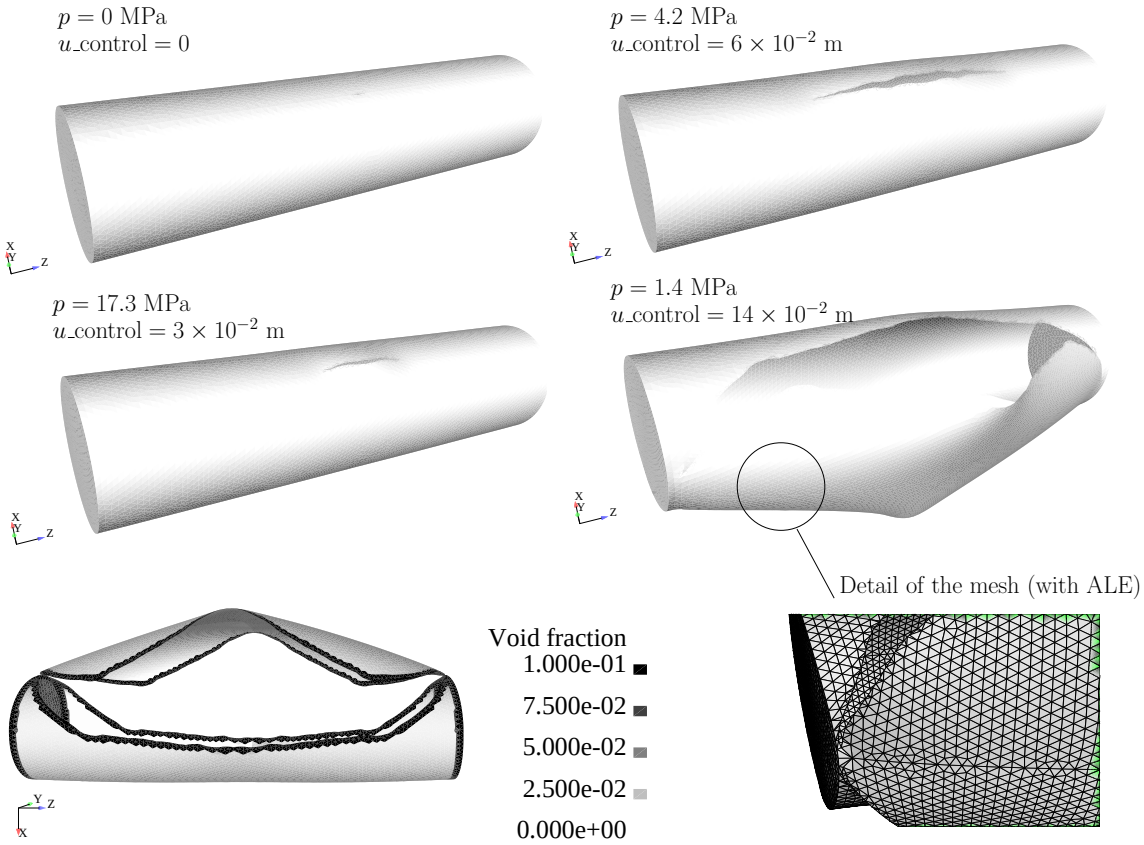


Figure 18: Quasi-brittle fracture of a cylindrical shell: sequence of deformed meshes.

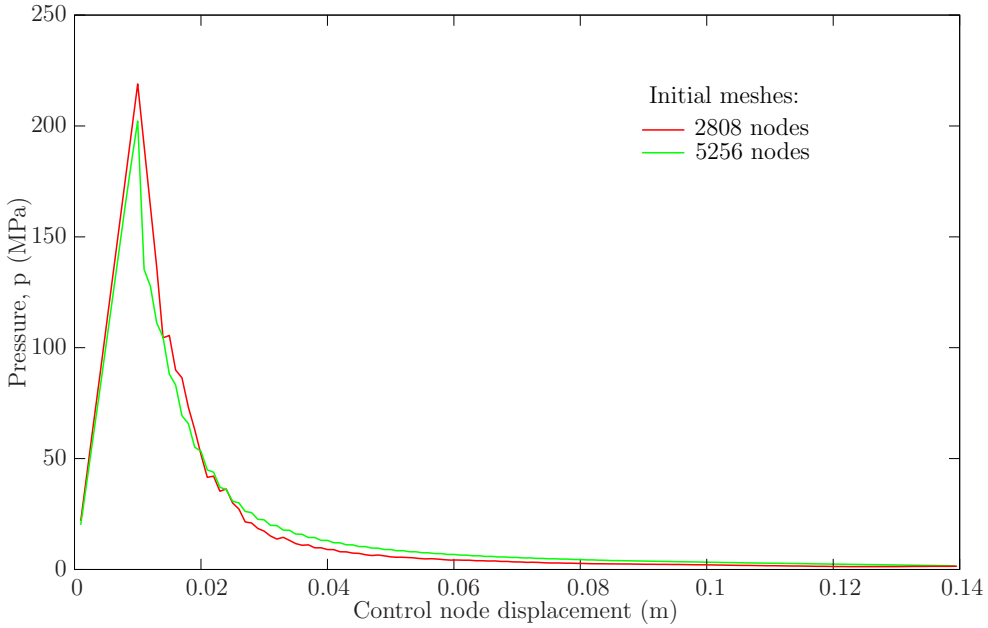


Figure 19: Quasi-brittle fracture of a cylindrical shell: control displacement/pressure results.

6 Concluding remarks

In this work a new algorithm and corresponding code to process both additive and multiplicative components in an 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 [4] under the LGPL license. It requires a Fortran 2003 compatible compiler.

Acknowledgments

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

REFERENCES

[1] F. Amirouche. *Fundamentals of Multibody Dynamics Theory and Applications*. Birkhäuser, 2006.

[2] S.S. Antman. *Nonlinear Problems of Elasticity*. Springer, Second edition, 2005.

[3] S.S. Antman and R.S. Marlow. Material constraints, lagrange multipliers, and compatibility. *Archive for Rational Mechanics and Analysis*, 116:257–299, 1991.

[4] P. Areias. Simplasmpc. http://code.google.com/p/simplasmpc/.

[5] P. Areias, D. Dias-da-Costa, J. Alfaiate, and E. Júlio. Arbitrary bi-dimensional finite strain cohesive crack propagation. *Computational Mechanics*, 45(1):61–75, 2009.

[6] P. Areias, J. Garção, E.B. Pires, and J. Infante Barbosa. Exact corotational shell for finite strains and fracture. *Computational Mechanics*, 48:385–406, 2011.

[7] P. Areias, N. Van Goethem, and 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.

[8] V.I. Arnold. *Mathematical Methods of Classical Mechanics*. Springer, 1989.

[9] K.-J. Bathe. Conserving energy and momentum in nonlinear dynamics: A simple implicit time integration scheme. *Computers and Structures*, 85:437–445, 2006.

[10] T. Belytschko, W.K. Liu, and B. Moran. *Nonlinear Finite Elements for Continua and Structures*. John Wiley & Sons, 2000.

[11] P. Bocca, A. Carpinteri, and S. Valente. Mixed mode fracture of concrete. *International Journal of Solids and Structures*, 27(9):1139–1153, 1991.

[12] J.I. Curiskis and S. Valliappan. A solution algorithm for linear constraint equations in finite element analysis. *Computers and Structures*, 8:117–124, 1978.

[13] T.A. Davis. *Direct Methods for Sparse Linear Systems*. SIAM, 2006.

[14] I.S. Duff, A.M. Erisman, and J.K. Reid. *Direct Methods for Sparse Matrices*. Clarendon Press, Oxford, 1986.

[15] F.G. Gustavson. Two fast algorithms for sparse matrices: multiplication and permuted transposition. *ACM Transactions of Mathematical Software*, 4(3):250–269, 1978.

[16] D. Jungnickel. *Graphs, Networks and Algorithms*, volume 5 of *Algorithms and Computation in Mathematics*. Springer, Second edition, 2005.

[17] A. Klarbring. *Models of Mechanics*. Springer, 2006.

[18] D. Knuth. *The Art of Computer Programming*, volume 3. Addison-Wesley, Third edition, 1997.

[19] S. Nash and A. Sofer. *Linear and Nonlinear Programming*. McGraw-Hill International Editions, 1996.

[20] P.E. Nikravesh. *Computer-Aided Analysis of Mechanical Systems*. Prentice Hall, 1988.

[21] J. Nocedal and S. Wright. *Numerical Optimization*. Series Operations Research. Springer, Second edition, 2006.

[22] T. Rabczuk and T. Belytschko. Cracking particles: a simplified meshfree method for arbitrary evolving cracks. *International Journal for Numerical Methods in Engineering*, 61:2316–2343, 2004.

[23] W.C. Rheinboldt. Geometric notes on optimization with equality constraints. *Applied Mathematical letters*, 9(3):83–87, 1996.