IMPLEMENTATION OF THE SYMBOLIC SIMPLIFICATION FOR THE CALCULATION OF ACCELERATIONS OF MULTIBODIES

Dmitry Vlasenko, Roland Kasper
*Institute of Mobile Systems (IMS), Otto-von-Guericke-University Magdeburg
Universitätsplatz 2, D-39016 Magdeburg, Germany
e-mails: Dmitri.Vlasenko@ovgu.de, Roland.Kasper@ovgu.de

KEYWORDS

Symbolic decomposition, multibody dynamics

ABSTRACT

This paper presents a method of simplification of multibody dynamics equations by preprocessing based on the symbolic decomposition and multiplication of sparse matrices. The method was implemented in the **V**irtual **S**ystem **D**esigner (VSD) software for the simulation of dynamics of CAD systems. Simulation tests show that the symbolical preprocessing greatly increases the numerical efficiency of the simulation.

INTRODUCTION

Nowadays there are many of methods performing the simulation of multibody systems. If simulated models are described using absolute coordinates, the equations of motion include large sparse matrices. Decompositions and multiplications of the matrices are the most numerically costly procedures in the simulation process.

The numerical efficiency of simulation methods can be significantly reduced if the sparse structure of matrices is taken into account. In the last years we developed a method for symbolic simplification of equations of motion based on preprocessing, which performs the object-oriented multibodies with complex structures and redundant constraints (Vlasenko and Kasper 2007:2). The symbolic simplification of decompositions and multiplications of matrices has several advantages in comparison with standard sparse solvers:

- Sparse structure of matrices is used completely without any run time overhead.
- The numerical operations with numerical elements of matrices are performed already during the translation.
- Additional operations with arrays of indexes (like in usual sparse solvers) are not needed.

In this article we show the results of the implementation of the method for the calculation of accelerations of multibodies. We developed in Maple a preprocessing module, which performs the symbolical simplification, and integrated it with our tool **V**irtual **S**ystem **D**esigner (VSD) for the object-oriented simulation of dynamics of CAD systems. Tests show that the integration of the preprocessing module with VSD greatly reduces the simulation time and the number of computations.

SYMBOLICAL SIMPLIFICATION OF THE DECOMPOSITION OF MATRICES

Let us consider a multibody system, consisting of rigid bodies, connected by holonomical constrains. The equations of constraints can be written as:

$$\mathbf{g}(\mathbf{q}) = 0 \tag{1}$$

where $\mathbf{q}$ is the vector of coordinates.

Differentiating this equation, we get the equations of constraints on the velocity level:

$$\mathbf{G}(\mathbf{q})\mathbf{v} = 0 \tag{2}$$

where $\mathbf{G}$ is the constraint Jacobian matrix, $\mathbf{v}$ is the vector of velocity variables.

Differentiating (1) twice, we obtain the equations of constraints on the acceleration level:

$$\mathbf{G}(\mathbf{q})\dot{\mathbf{v}} = \mathbf{u}(\mathbf{q},\mathbf{v}) \tag{3}$$

where

$$\mathbf{u} = -\dot{\mathbf{G}} \cdot \mathbf{v} \tag{4}$$

Combining (3) with the equations of motion in descriptor form:

$$\mathbf{M}\dot{\mathbf{v}} + \mathbf{G}(\mathbf{q})^T \boldsymbol{\lambda} = \mathbf{f}(\mathbf{q}) \tag{5}$$

we get the index-one formulation of the equations of motion (Eich-Soellner and Führer 1998; von Schwerin 1999) which can be used for the calculation of $\dot{\mathbf{v}}, \boldsymbol{\lambda}$

$$\dot{\mathbf{q}} = \mathbf{v} \tag{6}$$

$$\begin{pmatrix} \mathbf{M} & \mathbf{G}(\mathbf{q})^T \\ \mathbf{G}(\mathbf{q}) & 0 \end{pmatrix} \begin{pmatrix} \dot{\mathbf{v}} \\ \boldsymbol{\lambda} \end{pmatrix} = \begin{pmatrix} \mathbf{f}(\mathbf{q}) \\ \mathbf{u}(\mathbf{q},\mathbf{v}) \end{pmatrix} \tag{7}$$

where $\mathbf{f}$ is the vector of external forces, $\mathbf{M}$ is the mass matrix, $\boldsymbol{\lambda}$ is the vector of Lagrange multipliers.

This linear system can be efficiently solved by sparse solvers, exploiting a block-sparse structure of matrices $\mathbf{M}$ and $\mathbf{G}$ (e.g. null space methods, range space methods) (von Schwerin 1999; Lubich et al., 1995)

We developed an algorithm, based on the QR-decomposition of matrices, which can be used for the simulation of mechanical systems with complex structure, including closed loops and redundant constraints. Let us consider it more precisely.

From (2) we get the system of equations

$$\dot{\mathbf{v}} = \mathbf{M}^{-1}(\mathbf{f} - \mathbf{G}^T\boldsymbol{\lambda}) \tag{8}$$

$$\mathbf{G}\mathbf{M}^{-1}\mathbf{G}^T\boldsymbol{\lambda} = \mathbf{G}\mathbf{M}^{-1}\mathbf{f} - \mathbf{u} \tag{9}$$

Computing the Choleski decomposition of $\mathbf{M} = \mathbf{L}\mathbf{L}^T$, we obtain from (7) the matrix equation for $\boldsymbol{\lambda}$

$$\mathbf{A}^T\mathbf{A}\boldsymbol{\lambda} = \mathbf{b} \tag{10}$$

where

$$\mathbf{A} = \mathbf{L}^{-1}\mathbf{G}^T \tag{11}$$

$$\mathbf{b} = \mathbf{G}\mathbf{M}^{-1}\mathbf{f} - \mathbf{u} \tag{12}$$

If the matrix **A** is linearly independent (e.g. all rows of the Jacobian matrix **G** are independent), then, using the QR-decomposition of $\mathbf{A} = \mathbf{Q}\begin{pmatrix}\mathbf{R}\\ \mathbf{0}\end{pmatrix}$, we can calculate the value of $\boldsymbol{\lambda}$ as

$$\boldsymbol{\lambda} = \mathbf{R}^{-1}(\mathbf{R}^T)^{-1}\mathbf{b} \tag{13}$$

However, in the case of redundant constraints **G** has dependent rows! The presence of redundant constraints in CAD models is not unusual. In many cases design engineers develop CAD models, using more constraints than it is needful from the mechanical point of view. The redesign of CAD models and the elimination of redundant constraints by engineer is very costly procedure.

From (11) follows that if **G** has dependent rows, then **A** has dependent columns. Consider now the calculation of solution of (10) in this case. Clearly, if **A** has dependent columns then the product $\mathbf{A}^T\mathbf{A}$ is singular and the solution of (10) is not unique. In our case we need only an arbitrary solution with limited norm. Performing the QR-decomposition with pivoting of **A**, we obtain (Golub and van Loan 1996):

$$\mathbf{A}\boldsymbol{\Pi} = \mathbf{Q}\begin{pmatrix}\mathbf{R}_1 & \mathbf{R}_2\\ 0 & 0\end{pmatrix} \tag{14}$$

Here **Q** is an orthogonal matrix, **Π** is a permutation and $\mathbf{R}_1$ is a non-singular and upper triangular (*r*, *r*) matrix, where *r*=rank(**A**). Then the solution of (10) can be found using the formula (Vlasenko and Kasper 2007:2)

$$\boldsymbol{\lambda} = \boldsymbol{\Pi}_1\mathbf{R}_1^{-1}(\mathbf{R}_1^{-1})^T\boldsymbol{\Pi}_1^T\mathbf{b} \tag{15}$$

where $\boldsymbol{\Pi}_1$ is a part of the permutation matrix **Π**: $\boldsymbol{\Pi}=(\boldsymbol{\Pi}_1, \boldsymbol{\Pi}_2)$. From **(15)** follows that we do not need to calculate the matrix $\mathbf{Q}$, but only $\boldsymbol{\Pi}_1$, $\mathbf{R}_1$. Substituting the value of $\boldsymbol{\lambda}$ in (8), we calculate the absolute accelerations $\dot{\mathbf{v}}$.

Since using absolute coordinates, the matrix **A** usually has a sparse structure and includes both numerical and symbolic elements, e.g. the elements that are constant during the simulation and elements, depending on the coordinates of bodies. Therefore, the QR-decomposition of **A** can be optimized. We have developed a preprocessing module, which symbolically simplifying the QR-decomposition of matrices and for each decomposition generates a corresponding C-code.

This approach has the advantages pointed out in introduction. Generating C-code directly not only avoids calculating with zero elements, but also allows to preprocess all numerical parts of expressions. Indexing of matrices is avoided completely as linear code is generated.

It is well-known that the numerical complexity of the QR-decomposition depends on the order of columns. That is why we decompose not the matrix **A**, but the matrix $\tilde{\mathbf{A}} = \mathbf{A}\tilde{\boldsymbol{\Pi}}$ which is obtained from **A** by the reordering of columns.

We do not identify dependent rows in the matrix $\tilde{\mathbf{A}}$ on the preprocessing level because some elements of $\tilde{\mathbf{A}}$ **a**re not constant. That is why we can get the situation when, substituting in our C-procedure the numerical values of elements of $\tilde{\mathbf{A}}$ on an arbitrary time step, we obtain the matrix $\mathbf{R}_1$ having zero elements on the main diagonal. We propose the following algorithm of the solution of this problem:

1. Using a C-procedure, generated by the preprocessing module, we obtain from the numerical value of the matrix $\tilde{\mathbf{A}}$ the upper triangular matrix **R**, having zero elements on the main diagonal.

2. We permutate rows and columns of **R** in order to maximize the size of the non-singular upper triangular submatrix $\mathbf{R}_{1,1}$ and to minimize the size of the lower submatrix $\mathbf{R}_{2,2}$:

$$\mathbf{P}_r\mathbf{R}\mathbf{P}_c = \begin{pmatrix}\mathbf{R}_{1,1} & \mathbf{R}_{1,2}\\ 0 & \mathbf{R}_{2,2}\end{pmatrix} \tag{16}$$

   where $\mathbf{P}_r$ and $\mathbf{P}_c$ are permutation matrices.

3. We perform the QR-decomposition of the submatrix $\mathbf{R}_{2,2} = \mathbf{Q}_2\begin{pmatrix}\mathbf{U}_1 & \mathbf{U}_2\\ 0 & 0\end{pmatrix}\begin{pmatrix}\boldsymbol{\Pi}^T_{2,1}\\ \boldsymbol{\Pi}^T_{2,2}\end{pmatrix}$. Then the matrix $\mathbf{R}_1$ from (13) can be calculated as

$$\mathbf{R}_1 = \begin{pmatrix}\mathbf{R}_{1,1} & \tilde{\mathbf{R}}_{1,2}\\ 0 & \mathbf{U}_1\end{pmatrix} \tag{17}$$

   where $\tilde{\mathbf{R}}_{1,2} = \mathbf{R}_{1,2}\boldsymbol{\Pi}_{2,1}$. The matrix $\boldsymbol{\Pi}_1$ from (13) is a combination of submatrices of $\boldsymbol{\Pi}$, $\boldsymbol{\Pi}_{2,1}$.

DOUBLE INSULATOR CHAIN EXAMPLE

Consider a double insulator chain example (Hagedorn et al. 1980; Lubich et al., 1995; Vlasenko and Kasper 2007:1), shown in Figure 1. Each chain consists of insulators, connected by revolute joints. The first end of each chain is coupled with the triangular distance holder; the second is coupled with the ground. The holder is connected with the high voltage line, which is modeled as a force $\mathbf{f}_c$, acting on the holder.

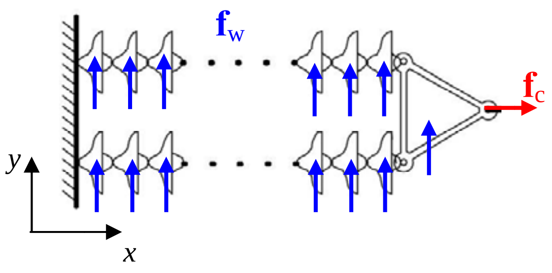

Figure 1. Double Insulator Chain Model

We modified the model, proposed by Hagedorn (Hagedorn et al. 1980) and added the wind force $\mathbf{f}_w$=10*N*, acting on the insulators and on the holder in the *y*-direction. Let *m* denote the number of insulators in each chain. Figure 2 shows the changes of the *y*-coordinate of the holder when *m*=8.

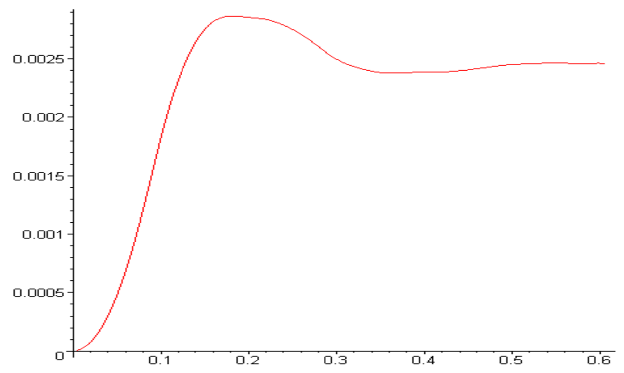

Figure 2. *y*-Coordinate of the Triangular Distance Holder

We simulated the dynamics of the example for different *m*, using the preprocessing module and the standard dense solver during the QR-decomposition of the matrix **A**. The results of the simulation are summarized in Table 1.

Table 1 Comparison of Simulation Effort of Double Insulator Chain Model

Chain length(m)	Flops (VSD)	Flops (dense solver)
4	5899	24400
8	11651	151632
16	23158	1058896

From Table 1 follows that we get a linear increase of the numerical operations in the case of the preprocessing module implementation vs. a cubic increase in the case of the dense solver implementation. This result corresponds to the linear increase of the simulation time for sparse solvers during the simulation of a one-chain insulator model.

INTEGRATION WITH VSD

Possibilities of the optimization of simulation

The theoretical results, shown in the previous section, shows that the simbolical simplification of the decomposition of matrices can significantly reduce the number of numerical operations during the simulation.

Moreover, not only the procedure of the decomposition of matrices can be simplified. If we consider precisely the equations (4), (11) and (12), we can see that the matrices $\mathbf{L}^{-1}$, $\mathbf{G}$, $\mathbf{M}^{-1}$ and $\dot{\mathbf{G}}$ are sparse. Therefore, the calculations of the matrix **A** and of the vectors **b** and **u** need the calculationss of products of sparse matrices. Taking into account the sparsity of the matrices during the calculation of the products, we can significantly improve the numerical efficiency of the simulation.

Since the matrices $\mathbf{G}$, $\dot{\mathbf{G}}$, etc. are sparse, we can write them down in a compressed form, including only the symbolical parts of matrices. The compressed form of matrices can be calculated, using the following procedure. Consider an arbitrary matrix **K**:

$$\mathbf{K} = \begin{pmatrix} 0 & k_{1,2} & 5 \\ k_{2,1} & k_{2,2} & 0 \\ 0 & 0 & k_{3,3} \end{pmatrix}$$

Then the dense form ${}^D\mathbf{K}$ of the matrix **K** can be calculated as union of rows of **K**, excluding all non-symbolycal elements:

$${}^D\mathbf{K} := (k_{1,2} \quad k_{2,1} \quad k_{2,2} \quad k_{3,3})$$

The use of the dense form of matrices reduces the number of memory, needed for the simulation.

Preprocessing module

We developed in Maple a preprocessing module which performs the symbolic optimisation of calculations. Starting from the mechanical parameters of a simulated system (e.g. masses of bodies, types and places of connections, etc.), the module generates a set of optimised C-procedures, which are then compiled in a .dll library. The library is called from VSD during the simulation of the mechanical system.

The module generates the following procedures:

1. The constraint Jacobian matrix ${}^D\mathbf{G}$ in a compressed form as a function of coordinates **q**.
2. The matrix $\mathbf{A} = \mathbf{L}^{-1}\mathbf{G}^T$ as a function of **q** and ${}^D\mathbf{G}$.
3. The product $\mathbf{M}^{-1}\mathbf{s}$ as a function of **q** and an arbitrary vector **s** (we call this procedure during the calculation of **b** from (12) and during the calculation of $\dot{\mathbf{v}}$ from (8))
4. The product $\mathbf{Gs}$ as a function of ${}^D\mathbf{G}$ and an arbitrary vector **s** (the procedure is called during the calculation of **b** from (12))
5. The vector $\mathbf{u} = -\dot{\mathbf{G}} \cdot \mathbf{v}$ as a function of coordinates **q** and velocities $\mathbf{v}$.
6. The matrix **R** as a function of matrix **A,** where **R** is an upper-triangular matrix, obtained by the QR-decomposition of **A**: $\mathbf{A} = \mathbf{Q}\begin{pmatrix} \mathbf{R} \\ \mathbf{0} \end{pmatrix}$.

The functions, generated by the preprocessing module, are much faster than the standard procedures of multiplications and decompositions and need less RAM because they use the sparse structure of matrices.

MANIPULATOR MODEL

We tested our preprocessing module, simulating the dynamics of a CAD model of a spatial manipulator, shown in Figure 3. Each stiff connection between bodies is defined by three plane-to-plane joints like it is usually defined by design engineers during the development of CAD models. This leads to the redundancy of constraints, which should be taken into account during the simulation of the manipulator. The complete model includes 8 bodies connected by 3 revolute joints and by 12 plane-plane joints.

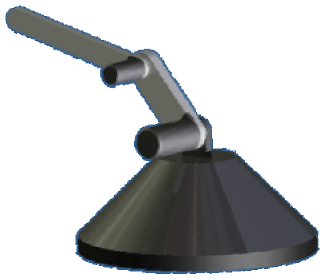

Figure 3. Manipulator Model

The matrix **A** matrix from (10), corresponding to the model, is shown in Figure 4, where symbolic elements are colored black, zero elements are blank and numerical elements are non-blank. Its length is 42 and the width is 51. The use of sparse methods for the decomposition of **A** is very effectively because its density is 17.6%. The QR-decomposition of **A** using the preprocessing module needs 34916 flops, which is 83% less than the number of operations needed for the decomposition of **A** using a standard dense solver.

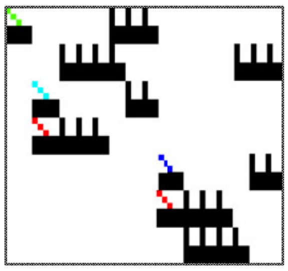

Figure 4. The matrix **A** of the model

The simulation of the dynamics of the model is carried out for the first 2 seconds using the Runge-Kutta integrator with the fixed timestep of 0.01 seconds. The use of the .dll library, generated by the preprocessing module, allows us to perform the simulation in 0.52 seconds, which is 5.5 times quicker than the simulation without preprocessing analysis (here we show the time of the simulation of the unstabilized model. In VSD is used also the second preprocessing module for the stabilization of constraints in the projection methods, but its description is very extensive and sophisticated and will be considered in other articles). This result shows that the use of the preprocessing module enables in future the real-time simulation of complex multibody systems.

CONCLUSION

The method of symbolical simplification of decomposition of sparse matrices can significantly increase the numerical efficiency of the calculation of multibodies' accelerations. The proposed method can be used for the simulation of multibodies with complex structure and redundant constraints.

The method was tested during the simulation of double chain examples. The tests results of the decomposition of matrices shows linear increase of the numerical operations using the preprocessing module vs. a cubic dependency of the dense solver.

The preprocessing module, simplifying numerical procedures with sparse matrices, was developed in Maple and integrated with the simulation software VSD. The results of the simulation of a CAD model of a manipulator show the high efficiency of the method.

REFERENCES

Eich-Soellner, E., Führer, C.: *Numerical Methods in Multibody Dynamics*, B. G. Teubner, Stuttgart (1998).

Golub, G. H.; Charles van Loan, F.: *Matrix Computations*, 3rd ed., Johns Hopkins UP, 1996.

Hagedorn, P., Idelberger, H., Mocks, L: Dynamische Vorgänge bei Lastumlagerung in Abspannketten von Freileitungen. etz Archiv 2, p 109-119 (1980).

Lubich, Ch., Nowak, U., Pöhle, U., Engstler, Ch.: *MEXX - Numerical software for the integration of constrained mechanical multibody systems*. Mech. Struct. Mach. 23, p 473-495 (1995).

von Schwerin, Reinhold: *Multibody System Simulation. Numerical Methods, Algorithms and Software*. Springer, 1999

Vlasenko, D.; Kasper, R.; (2007): *A New Software Approach for the Simulation of Multibody Dynamics.* ASME Journal of Computational and Nonlinear Dynamics, Volume 2, Issue 3, 2007, pp. 274-278.

Vlasenko, D.; Kasper, R. (2007): *Sparse Matrix Method for Component-Oriented Dynamic Simulation of Multibodies in VSD Software*. Proceedings of Multibody Dynamics 2007 (ECCOMAS Thematic Conference), Milan, Italy, June 25-28 June, 2007

BIOGRAPHY

Dmitry Vlasenko (born 1977) graduated from the Novosibirsk State University, Russia with a B.A. in 1998 and from St. Petersburg State University, Russia with M.A. in 19200. He obtained his PhD degree in Mechanical Engineering from Otto-von-Guericke-University Magdeburg, Germany in 2006. From 2006 he works as a research engineer at University of Magdeburg, Institute of Mobile Systems.