MULTIBODY DYNAMICS 2011, ECCOMAS Thematic Conference
J.C. Samin, P. Fisette (eds.)
Brussels, Belgium, 4-7 July 2011

SYMBOLIC COMPUTATION TECHNIQUES FOR MULTIBODY MODEL DEVELOPMENT AND CODE GENERATION

Chad Schmitke and Paul Goossens

Maplesoft
Waterloo, Ontario, Canada
e-mail: cschmitke@maplesoft.com,
web page: http://www.maplesoft.com

Keywords: symbolic computation, multibody, maplesim, maple, optimized code generation

Abstract. *This paper presents some of the benefits of a general purpose symbolic computation environment when constructing and generating simulation code for multibody, multi-domain systems. Specifically, it considers how tools provided by these environments can be harnessed to generate highly efficient simulation code. MapleSim™, a modeling and simulation platform that is based on the Maple™ symbolic computation engine is used as the investigative tool. As a case study, different approaches to an inverse dynamics solution of a Stewart-Gough platform are modeled and exported to C-code for simulation and timing.*

1 INTRODUCTION

One of the goals of multibody dynamics research is to advance techniques for developing efficient simulation code. To this end, symbolic computation has been shown to be beneficial for many classes of problems [1]. Many of the straightforward benefits of symbolic computation in multibody formalisms are well known: automatic removal of multiplications by 1s and 0s, cancellation of common terms, trigonometric simplifications, etc. As well, most of these operations can be performed without implementing a general purpose symbolic computation engine (GPSCE), that is, they can be coded into routines that run while simulation code is being processed (or during the formulation stage), without the need to provide the user with any ability to directly view or manipulate the underlying equations. The equations are only seen as text in the software's target code.

The goal of this paper is to present some additional advantages of having access to a GPSCE during the model development and code generation phases. The first section discuses how coordinate selection can impact both the form and complexity of system's governing equations. The second section looks at how being able to view and manipulate these equations (and their structure) in a GPSCE can lead to further simplifications (and perhaps inform the coordinate selection). Generation of optimized simulation code is considered in the third section, with a brief discussion of how a specific GPSCE, Maple, leverages its internal representation of the equations to generate efficient code. This is followed by example timings of a Stewart-Gough manipulator using some of the aforementioned techniques. The paper concludes with some summary thoughts on the use of GPSCE's for formulating and generating code for multibody systems.

2 COORDINATE SELECTION

Although different multibody formalisms may be more or less efficient than one another, the efficiency of the final set of equations is strongly dependent on the chosen coordinates. As Mitiguy and Kane [2] point out, "Generally, the choice of variables made by the analyst has a profound effect on the efficiency of the resulting equations".

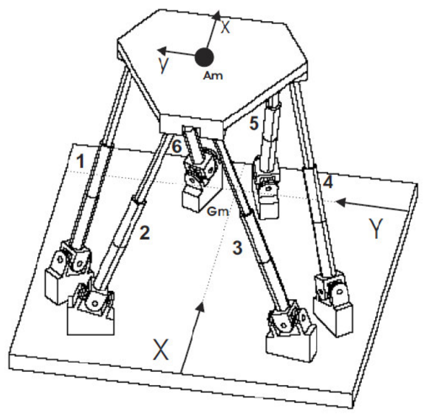


Figure 1: Stewart-Gough Manipulator

Consider the Stewart-Gough manipulator shown in Figure 1. The 6 DoF (Degree of Freedom) mechanism consists of 6 identical legs fixed to ground via universal joints, and attached to a common platform via spherical joints. The motion of the platform is controlled by the 6 actuators which drive the length of each leg.

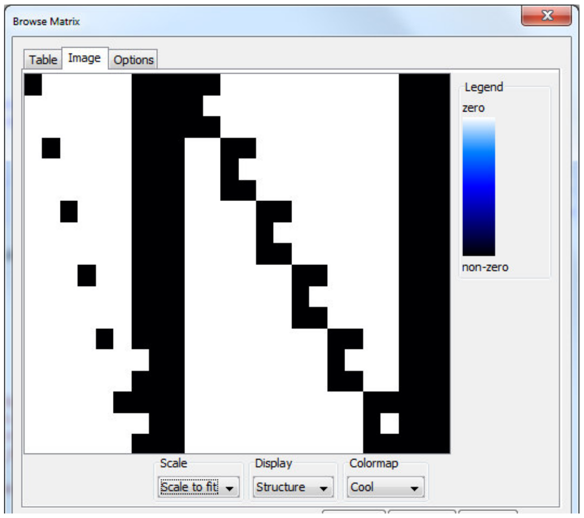
Using a selection of absolute coordinates (the *x*-,*y*-,*z*-*translatio*n and *roll*, *pitch*, *yaw* orientation of each rigid body) to model this system results in 78 coordinates – 6 for the platform motion and 12 for each of the 6 legs (each leg having 2 rigid bodies). These coordinates necessitate 78 ordinary differential equations (ODEs) to govern their motion. Since the platform only has 6 DoF, 72 additional constraint equations are required to account for the lack of independence between the coordinates. This gives a total of 150 differential-algebraic equations (DAEs) for this model. The large number of constraints can be particularly costly during simulation since they will need to be handled using some type of iterative approach.

In contrast, using a set of hybrid coordinates (a combination of absolute and joint coordinates), the platform can be modeled using 24 variables – 6 for the platform motion and 3 for each of the 6 legs (2 for the universal joint at the base and 1 for the prismatic joint). This requires 24 ODEs and 18 constraint equations, resulting in only 42 DAEs.

Using pure joint coordinates (replacing the platform coordinates with a spherical joint in one of the legs), we could reduce the coordinate count to 21, but the structure of the equations is not as efficient. In general, fewer coordinates do not always lead to more efficient simulations (3). Being able to control the coordinate selection is important as it allows expert users to apply domain knowledge to a given model.

3 SYMBOLIC MANIPULATION

This control over the coordinates is particularly advantageous when one can then view and manipulate these equations using a GPSCE. The Stewart Platform shown in Figure 1 and described in [4] was created in MapleSim, and the equations were formulated and retrieved via the multibody analysis package. One of the commands in this library returns the symbolic constraint Jacobian for the system (the partial derivative of the constraint equations with respect to the modeling coordinates).



Scale: Scale to fit
Display: Structure
Colormap: Cool

Figure 2: Graphical Depiction Of Symbolic Jacobian

Although it is difficult to glean any insight from the fully expanded Jacobian, looking at the symbolic structure of the Jacobian can lead to a better understanding of the system's make-up. This structure is shown in Figure 2. Here, white squares corresponds to symbolically 0 terms (0 for all time, not just at a single instant in time) and black squares corresponds to potentially non-zero terms. The two vertical black columns correspond to the platform's three translations and rotational coordinates, respectively. Clearly these coordinates are involved in all of the constraint equations. However, and as direct result of using the previously mentioned hybrid coordinate selection, the joint coordinates for each leg only appear in a cluster of three equations – the constraints related to the *xyz*-constraint of each leg's

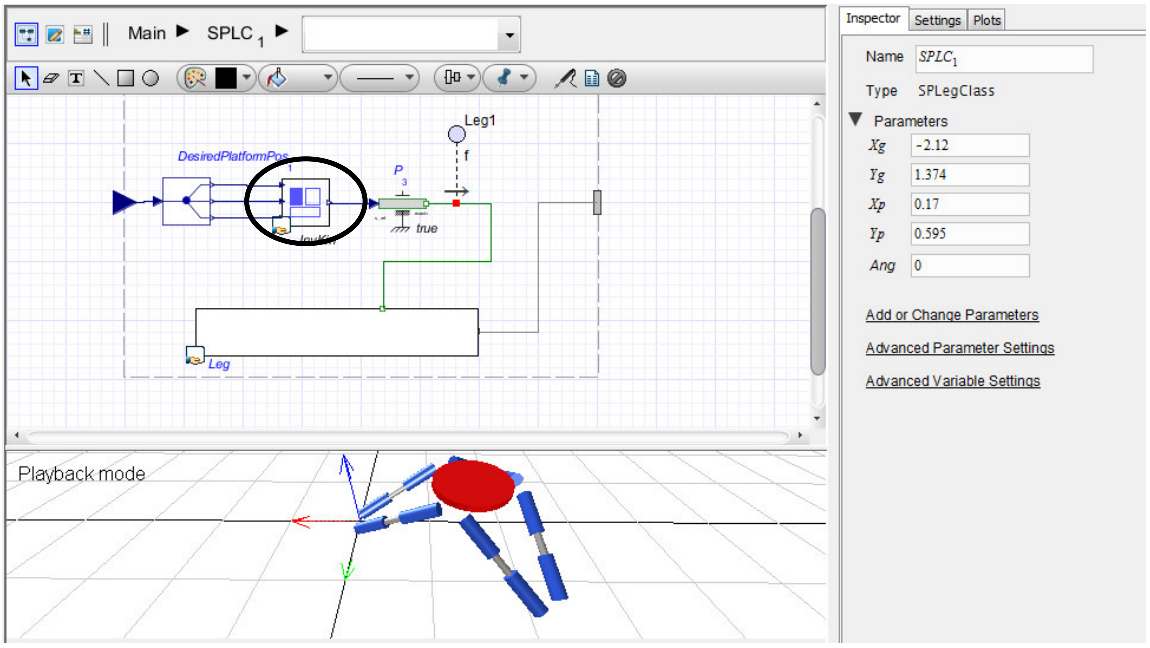
$$
\begin{aligned}
\Big[\Big[&-1. - 1.\,s + X\cos(ang)\cos(\beta) + Y\sin(ang)\cos(\beta) \\
&+ 1.\,Xp\cos(ang)\cos(\beta) + 1.\,Yp\sin(ang)\cos(\beta) \\
&- 1.\,Yg\sin(ang)\cos(\beta) + \cos(\alpha)\sin(\beta)\,Z \\
&- 1.\,Xg\cos(ang)\cos(\beta) - 1.\,Xp\sin(ang)\sin(\alpha)\sin(\beta) \\
&+ 1.\,Yp\cos(ang)\sin(\alpha)\sin(\beta) + Xg\sin(ang)\sin(\alpha)\sin(\beta) \\
&- 1.\,Yg\cos(ang)\sin(\alpha)\sin(\beta) + Y\cos(ang)\sin(\alpha)\sin(\beta) \\
&- 0.4000000000\cos(\alpha)\sin(\beta) - 1.\,X\sin(ang)\sin(\alpha)\sin(\beta)\Big], \\
\Big[&-1.\sin(ang)\cos(\alpha)\,X + \sin(ang)\cos(\alpha)\,Xg \\
&+ \cos(ang)\cos(\alpha)\,Y - 1.\cos(ang)\cos(\alpha)\,Yg \\
&+ 0.4000000000\sin(\alpha) - 1.\sin(ang)\cos(\alpha)\,Xp \\
&+ 1.\cos(ang)\cos(\alpha)\,Yp - 1.\sin(\alpha)\,Z\Big], \\
\Big[&-1.\,X\sin(ang)\sin(\alpha)\cos(\beta) + Y\cos(ang)\sin(\alpha)\cos(\beta) \\
&- 1.\,Xp\sin(ang)\sin(\alpha)\cos(\beta) + 1.\,Yp\cos(ang)\sin(\alpha)\cos(\beta) \\
&+ Xg\sin(ang)\sin(\alpha)\cos(\beta) - 1.\,Yg\cos(ang)\sin(\alpha)\cos(\beta) \\
&- 1.\,Xp\cos(ang)\sin(\beta) + Yg\sin(ang)\sin(\beta) \\
&- 0.4000000000\cos(\alpha)\cos(\beta) + Xg\cos(ang)\sin(\beta) \\
&- 1.\,Y\sin(ang)\sin(\beta) - 1.\,X\cos(ang)\sin(\beta) \\
&+ \cos(\alpha)\cos(\beta)\,Z - 1.\,Yp\sin(ang)\sin(\beta)\Big]\Big]
\end{aligned}
$$

Figure 3: Parameterized Constraint Equations

spherical joint. A quick symbolic comparison confirms that the parameterized constraints of these 6 groups are symbolically identical. A snapshot of the 3 constraint equations (taken from Maple) are shown in Figure 3.

Here *X*, *Y* and *Z* correspond to the desired translation of the platform (orientation is assumed level). *Xp*,*Yp*, *Zp* and *Xg*,*Yg*,*Zg* correspond to the location of the leg with respect to frame *Am* and frame *Gm* in coordinates local to each frame (see Figure 1). The variables α, β and *s* are the universal joint angles and the prismatic joint's displacement, respectively. Finally, *ang* is an offset angle controlling the orientation of the leg with respect to the *Gm* frame.

Even more useful than helping to identify the structure of the equations, a GPSCE can also be used to analytically solve these equations for the desired variables. Using Maple's **solve()** command, and passing in the three equations shown, we can immediately obtain a solution for alpha α, β and *s* as a function of only the platform position and the leg parameters – effectively solving the inverse kinematics problem for each leg without needing to perform any manual manipulation of the equations.



Name SPLC₁

Type SPLegClass

Parameters

Xg	-2.12
Yg	1.374
Xp	0.17
Yp	0.595
Ang	0

Figure 4: Snapshot Of MapleSim Interface High-lighting Inverse Kinematics Block

Once obtained, MapleSim allows for the automatic creation of block components from the derived equations. All that is required is to map the variables appearing in the equations to ports on the block. Shown circled in Figure 4, this block takes the *XYZ* motion of the platform, and the *Xp*,*Yp*, *Zp* and *Xg*,*Yg*,*Zg* parameters relating the leg to the platform and ground, and outputs the required length of variable *s* (the length of the leg). Used in conjunction with a position driver, we can easily measure the system's inverse dynamics, that is, the forces required in each leg to generate the prescribed motion.

To verify the correctness of the solution we compare against the results shown in [4] for a diagonal heaving of the platform. The results, which match those listed by Tsai, are shown in Figure 5.

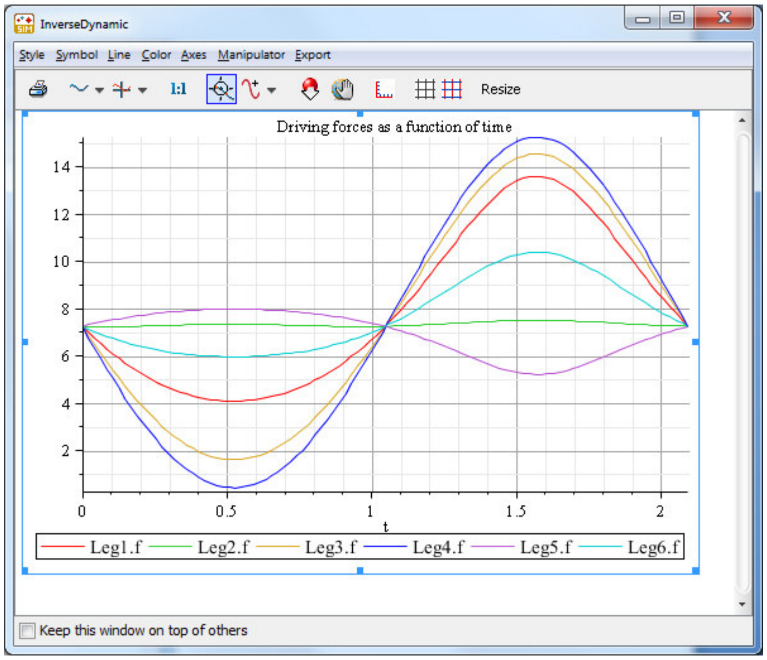


Figure 5: Forces Required In Each Leg During Heaving Motion

4 OPTIMIZED CODE GENERATION

In the previous example, no additional coding was required to transform the solved equations into efficient simulation code. That task was entirely handled by the GPSCE. To talk about this further we need to understand a bit more about how equations are internally stored withing the engine – in this case, Maple.

Maple uses a Directed Acyclic Graph (DAG) to store its internal representation of a given set of equations (see the discussion on repeated subexpressions in [5]). This structure looks for repeated subexpressions of the expression and stores them only once, referencing them elsewhere in the equations. Not only is this efficient for managing a computer's memory resources, but it also provides a significant benefit in the area of generating efficient simulation code.

As a simple example, consider the set of equations shown in Figure 3. Using Maple's **cost()** function, we can calculate the equations to have 35 additions/subtractions, 87 multiplications, and 82 functions (**sin()**,**cos()** calls). Applying optimization, which takes advantage of the internal DAG structure, yields the equations shown in Figure 6:

$$\begin{aligned}&[\,t1=\cos(ang),\ t2=\cos(\beta),\ t3=\sin(ang),\ t4=\cos(\alpha),\ t5=\sin(\beta),\ t6\\&=\sin(\alpha),\ t7=-X+Xg-Xp,\ t8=Y-Yg+Yp,\ t9=t8\,t1\\&+t7\,t3,\ t10=-\frac{2}{5}+Z,\ t11=t6\,t9+t10\,t4,\ t1=t7\,t1,\ t3=t8\,t3,\\&CON1=(-t1+t3)\,t2+t11\,t5-1-s,\ CON2=t4\,t9-t10\,t6,\\&CON3=t11\,t2+(t1-t3)\,t5\,]\end{aligned}$$

Figure 6: Optimized Constraint Equations

Here, *tn* corresponds to temporary variables created for repeated subexpressions. *CON1*, *CON2*, and *CON3* represent the original three constraints in terms of the repeated expressions. A **cost()** of these equations yield 14 additions/subtractions, 12 multiplications, and 6 functions.

Running this optimization on the 42 DAEs for the Stewart Platform gives a similar magnitude of drop in the number of computations. Specifically, the original 14096 additions drops to 2369, and the 35163 multiplications drops to 2766.

5 SIMULATION TIMINGS

Although counting the number of computations in the expressions can be useful, a better comparison is the actual integration time. Using the MapleSim Connector®, two inverse dynamics solutions to the Stewart-Gough manipulator were exported to C-code and wrapped in MATLAB®/Simulink® S-function. In one case the inverse kinematics was left as a set of nonlinear equations (those shown in Figure 3) to be solved iteratively during simulation. In the other case, the analytical solution to the inverse kinematics problem was used.

In order to solve DAEs in MATLAB®/Simulink®, a projection step is used at the end of every major integration step, perturbing the solution slightly in order to project the states back onto the constraint manifold [6]. This projection step is iterative, and MapleSim allows you to loop to a specific error tolerance in the constraint violation, or specify a fixed number of iterations.

A straight running of the two models against one another (fixed-step, Euler solver) showed a 3.5x speed-up when the inverse kinematics were solved analytically. In part, this was due to the additional projection calls made in the unsolved case, as more work was required in order to achieve the same level of accuracy in the constraint residual. When both models were limited to a single projection call per time-step, the solved case was still 2.5x faster.

6 CONCLUSIONS

The goal of the paper has been to show how access to a GPSCE during the model creation as well as the code generation phase can provide additional opportunities for a user to further simplify their system.

Coordinate selection coupled with the ability to manipulate the derived equations provides significant opportunities to generate more efficient code. In this paper we've shown how a set of coordinates can be chosen to yield repeated subsystems of constraint equations. Using the GPSCE, these kinematic constraints can be analytically solved and sent back into the simulation. However, this is just an example of the types of operations that can be performed in a GPSCE – order reduction, symbolic differentiation, etc. are also available to the user in these environments.

The code generation phase can take advantage of a GPSCE by leveraging the internal structures that describe the equations. Since leading GPSCEs (like Maple) have spent decades managing and compacting these structures, being able to automatically re-use them for code optimization can lead to very efficient simulation code – without the need for manual programming.

Finally, it was shown that by using a GPSCE to obtain an analytical solution for a Stewart-Gough platform's inverse kinematics, we were able to speed-up the simulation time of the inverse dynamics solution 2.5x to 3.5x times.

REFERENCES

[1] J.C. Samin and P. Fisette. *Symbolic Modeling of Multibody Systems*. Kluwer Academic Publisher, Dordrecht, 2003.

[2] P.C. Mitiguy and T.R. Kane. Motion variables leading to efficient equations of motion. *International Journal of Robotics Research*, **15**, 522–532, 1996.

[3] M. Leger, and J. McPhee, Selection of Modeling Coordinates for Forward Dynamic Multibody Simulations, *Multibody System Dynamics*, **18**, 277-297, 2007.

[4] Lung-Wen Tsai. Solving the InverseDynamics of a Stewart-Gough Manipulator by the Principle of Virtual Work. *Journal of Mechanical Design*, **122**, 3-9, 2000.

[5] Allan Wittkopf, Automatic Code Generation and Optimization in Maple, *Journal of Numerical Analysis, Industrial and Applied Mathematics*, **1**, 1-13, 2007.

[6] Lawrence F. Shampine, Mark W. Reichelt, Jacek A. Kierzenka, Solving index-1 DAEs in MATLAB® and Simulink®, *SIAM Review*, **41**, 538-552, 1999.