New Methods for Hardware-in-the-loop Simulation of Stiff Models

Hilding Elmqvist, Sven Erik Mattsson and Hans Olsson
Dynasim, Lund, Sweden, E-mail: {elmqvist, svenerik, hans}@dynasim.se

Abstract

The possibilities of multi-domain hierarchical modeling in Dymola often lead to models with both fast and slow parts and the simulation problems become stiff. The usual use of the explicit Euler method for hardware-in-the-loop simulations is not appropriate, because it requires very small step sizes and thus too large computational efforts. The implicit Euler method allows larger step sizes to be used. However, a non-linear system of equations needs to be solved at each step. Reducing the size of the non-linear problem is advantageous. The method of inline integration was introduced to support this. The discretization formulas of the integration method are combined with the model equations and structural analysis and computer algebra methods are applied on the augmented system of equations. This paper describes and illustrates some very important improvements in Dymola's support of the inline integration method. The symbolic analysis and manipulation have been improved and it reduces, in many cases the, size of the non-linear system drastically. Analytic Jacobians for the nonlinear system also increase efficiency and robustness. Support of inline integration of higher order leads to better accuracy for larger steps.

Introduction

Real-time simulation of physical models is a growing field of applications for simulation software. One goal is to be able to simulate more and more complex models in real-time with fast sampling rates. Many of those models are multi-engineering models, which means, that they contain components from more than one engineering domain. Mechanic, electric, hydraulic or thermodynamic components are often coupled together in one model. This leads to a large span of time-constants in the model. The usual use of the explicit Euler method is not appropriate because the fastest time-constant determines the computational effort (step size) for the simulation. In order to maintain stability of the integration method the step size must be less than the smallest time constant. Typically, the fastest modes are not excited to a degree that it is necessay to resolve them for the intended purpose. In such cases the problem is referred as stiff. The implicit Euler method solves the numerical *stability* problem and allows larger step sizes to be used. It is the *accuracy* required that restricts how large step sizes that can be used. Using the implicit Euler method, on the other hand, implies that a nonlinear system of equations needs to be solved at every step. The size of this system is at least as large as the size of the state vector, n. Solving large nonlinear systems of equations in real-time somewhat problematic because the number of operations is $O(n^3)$ and the number of iterations might vary for different steps. Reducing the size of the nonlinear problem is advantageous. Due to the hybrid nature of the system the Jacobian of the nonlinear system can change drastically between steps. This makes it difficult to apply methods relying on Jacobian updating.

The method of inline integration [3] was introduced to handle such cases. The discretization formulas of the integration method are combined with the model equations and structural analysis and computer algebra methods are applied on the augmented system of equations. For a robotics model with 66 states, the size of the nonlinear system of equations could be reduced to only 6. This method has had little practical use, because certain pragmas about the structure of the model equations had to be put into the model by the user.

Another method, "mixed-mode integration", of reducing the size of the system of nonlinear equations is to use explicit discretization on slow states and implicit on fast states. The problem is then to find the partitioning of the state vector into slow and fast states. A method based on linearization and eigenvalue analysis was presented in [6]. Since the partitioning is based on linearization, special care is needed for highly non-linear and partly discrete model such as friction. In addition it requires a pre-processing step that includes off-line simulation and "suitable" inputs. It is thus not straightforward to use this method.

This paper describes and illustrates some important improvements in Dymola's [1,2] support of the inline integration method.

1. The symbolic analysis and manipulation have been improved and it reduces, in many cases the, size of the non-linear system drastically.
2. The generation of analytical Jacobians has been improved.
3. Inline integration of higher order methods are supported.

The large possible reduction of the size of the implicit non-linear system of equations is due to the fact that certain subsystems might be linear even after ammendment of the corresponding discretization formulas. Dymola is now able to automatically detect such structures during the structural analysis of the equations. Furthermore, in certain cases the corresponding linear subproblem is sparse. This is, for example, the case for discretized PDE's. For a one dimensional PDE, a band structure is obtained. The usual technique of tearing then implies a reduction of the size of the problem. For a PDE model with 10 elements, the size of the nonlinear problem, i.e. the number of iteration variables, can often be reduced to one when a first order spatial discretization is used.

The implicit inline Euler technique solves the numerical stability problem. However, the step size need to be chosen small enough to get desired accuracy. Dymola support of inline integration has been extended with higher order methods to meet the accuracy requirements. The use of higher order methods is necessary for e.g. hydraulics systems where one can have oscillations in the kHz-range and want to use step-sizes for external sampling in the same range.

Exploiting sparse structures

Consider a system of differential algebraic equations (DAE)

$$F(t, x, \dot{x}, y) = 0$$

where t is time, x and y are vectors of unknown variables. The elements of x are called dynamic variables since their time derivatives, $\dot{x}$, appear in the equations and the elements of y are called algebraic variables since none of its derivatives appear in the equations.

When making inline discretization, the model equations are combined with the discretization formulas of the integration method. For implicit Euler we get the nonlinear problem

$$F(t_i, x_i, \dot{x}_i, y_i) = 0$$
$$\dot{x}_i = (x_i - x_{i-1})/h$$

to solve for x_n and y_n at each step. Also for an ODE on explicit state space form,

$$\dot{x} = f(t, x)$$

the inlined integration method using implicit Euler gives a non-linear problem. The size of the problem is the size of the state vector.

The non-linear systems obtained when combining the discretization formulas of implicit integration methods with model equations are sparse, because typically a model has hundreds or thousands of unknowns, while each equation refer to very few, say ten, variables. There is much structure to exploit.

Let us represent the structure of a system by a structure Jacobian, J, where each row represents a scalar equation and each column represents an unknown variable of the system. If variable j does not appear in equation i then $J_{ij} = 0$. Otherwise it is one. The representation can be extended to indicate how it appears, for example, whether it appears linear or not.

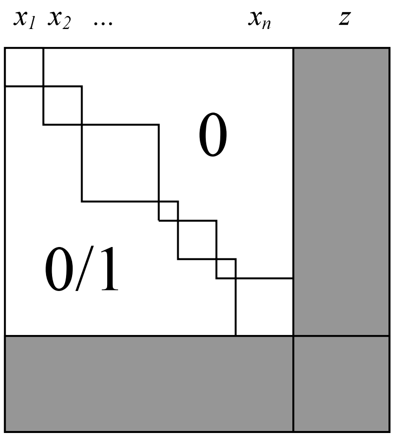

Figure 1: A desired structure for the Jacobian.

Consider a structure Jacobian of the form as shown in Figure 1. The elements of the right and lower borders (the grey part) can have any values. It is the structure of the upper left part (the white part) that is important. It shall be block lower triangular (BLT) and each diagonal block shall be non-singular.

If the z variables are assumed known, the problem of solving for the x variables is decomposed into a sequence of smaller problems that be solved in turn giving $x_1, x_2, ..., x_n$.

It means that when using a numerical solver to solve the total problem, the numerical solver needs only iterate over the z variables which is a smaller problem. A numerical solver needs residuals to be calculated, when it provides a value for the z. The residual is calculated in the following way

1. Solve in turn the sequence of problems for the x_i values using the given z value and the x_j ($j < i$) values already calculated.
2. Use the z value and the calculated x_i values to calculate the residuals of the remaining equations at the bottom.

To obtain efficient simulation, the aim is to obtain a small number of z variables while keeping the sequence of problems to solve x_i simple. It is favourable if the calculations of the x variables are just a sequence of assignment involving no numerical solvers. Small linear systems of equations are also acceptable. It is very important that the subproblems to solve for the x_i variables are nonsingular. If the original problem is non-singular, then the manipulation must not introduce singularities or divisions by zero. Unfortunately, it is not only a question avoiding divisions by zero, but also divisions by too small numbers. When solving linear equations this is commonly solved by pivoting in order to avoid large condition numbers of the factorized matrices.

When solving the outer nonlinear problem, it is favourable to use Newton methods. Fixed point iteration cannot be used for stiff problems. Newton methods need the Jacobian of the problem. Let n_z denote the number of elements of z. The Jacobian can be calculated numerically, by performing additional n_z residual calculations, which may be costly. By generating code for analytic calculation of the Jacobian the effort to calculate all non-zero elements of the Jacobian typically is of the same magnitude as one residual calculation, which is a considerably less effort. This reduction is due to common subexpression elimination.

Higher order methods

In order to get sufficient accuracy for large steps it was necessary to extend the basic method to higher order methods. Higher order methods indicate that they have order greater than one, and the ones considered have orders 2 to 4.

The higher order methods implemented for the new method are L-stable singly diagonally implicit Runge-Kutta methods [4]. The L-stability implies that they are stable for all stable linear systems and do not exhibit oscillations for very stiff systems. The class of methods, singly diagonally implicit Runge-Kutta methods, require the solution to the same equation systems as implicit Euler.

Other high order methods

More general implicit Runge-Kutta methods can often be made more efficient in off-line simulations. However, this requires more costly factorizations that can be shared between many steps and is thus not suitable for realtime simulations. Multi-steps methods and other methods that propagate more information from one step to the next are not suited for real-time simulations of hybrid systems.

Example: One dimensional PDE

Discretized partial differential equations (PDE) have special sparse structures because each unknown appears only in a few equations. For a one dimensional PDE, a band structure is obtained.

Consider the following PDE, modeling one-dimensional heat diffusion.

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2}$$

with boundary conditions

$$u(x, t=0) = 20\sin(\frac{\pi}{2}\frac{x}{L}) + 300$$

$$u(x=0, t) = 20\sin(\frac{\pi}{12}t) + 300$$

$$\frac{\partial u}{\partial x}(x=L, t) = 0$$

where L is the length of the object. By discretizing in space

$$\frac{\partial^2 u}{\partial x^2}(x,t) \cong \frac{u(x+\Delta x,t) - 2u(x,t) + u(x-\Delta x,t)}{\Delta x^2}$$

where $\Delta x = L/n$ with n being the number of discrete elements, the PDE can be transformed into a set of ODEs. The matrix notation allows convenient description of the discretized model.

```
model PDE
  parameter Real L = 1;
  parameter Integer n = 50;
  Real Dx = L/n;
  constant Real Pi=3.14159265;
  Real u[n+1];
equation
  u[1] = 20*sin(Pi/12*time) + 300;
  der(u[2:n]) = (u[3:n + 1] -
    2*u[2:n] + u[1:n - 1])/(Dx*Dx);
  u[n + 1] = u[n - 1];
initial equation
  u[2:n] = 20*sin(Pi/2*(1:n-1)*Dx) +
         300*ones(n-1);
end PDE;
```

The discretized ODE is conveniently written by use of shifted sub-ranges of the vector u. The boundary condition at t=0 is given as an initialization equation. The sine function is evaluated elementwise on the sequence. The boundary condition at x=0 is handled by making u[1] an algebraic variable with given time dependency. The boundary condition at x=1 is handled by adding one element to u, namely u[n+1], and the equation u[n+1] = u[n-1].

By discretizing in time using implicit Euler

```
der(u[i]) = (u[i]-old(u[i]))/h
```

where h is the step size and the expression old(u[i]) denotes the value of u[i] at the previous step, the ODE

```
der(u[2:n]) = (u[3:n + 1] -
    2*u[2:n] + u[1:n - 1])/(Dx*Dx)
```

is transformed into

```
u[3:n + 1] = (2+a)*u[2:n] - u[1:n - 1]
             - a*old(u[2:n])
```

where a is the constant

```
a = Dx*Dx/h
```

The first component of the discretized ODE is

```
u[3] = (2+a)*u[2] - u[1] - a*old(u[2])
```

The variable u[1] is known because it simply calculated from the boundary condition given as a pure time dependent expression. Thus the equation has two unknowns, u[2] and u[3], since all old expressions are known quantities when taking a new step. If u[2] is known, it is simple to calculate u[3].

Let us assume u[1:3] to be known and consider the second component of the discretized ODE

```
u[4] = (2+a)*u[3] - u[2] - a*old(u[3])
```

which is simple to use to calculate u[4]. Proceeding in the same way for all components of the discretized ODE, we find equations for calculating u[3:n+1] in a simple way when u[2] is assumed to be known.

The remaining equation is

```
u[n + 1] = u[n - 1];
```

which now is used to give the residual u[n+1]-u[n-1] for calculating u[2] iteratively. In other words the numerical solver need only iterate over one variable.

Since this problem is linear, Dymola continues the symbolic manipulation and uses the explicit expressions for u[3:n+1] to back-substitute the residual equation to get an equation for u[2] and solves this equation symbolically. Dymola has transformed the model to a simple sequence of assignments and there is no need for a numerical solver.

This model for heat diffusion is not stiff, but it illustrates very well how the sparse structures of discretized PDEs can be exploited. Moreover, such a model can be part of a model that is stiff. Dymola is then able to find and treat these equations as described.

Models of hydraulics systems are stiff. Models to describe pressure wave oscillations in the kHz range in long lines have the same banded structure as discussed above and Dymola is able to find and to reduce the size of the non-linear system of equations automatically.

Example: Multi-body systems

Consider modeling of multi-body systems. The equation of motion can be written as

$$M(q)\ddot{q} = F(q, \dot{q})$$

where q is a vector of generalized coordinates representing the system's position (distances or angles), M is the non-singular mass matrix, and F represents applied forces. Let n denote the number of elements of q or in other words the degree of freedom for the mechanical systems. The states are q and $\dot{q}$. Thus the number of states is $2n$.

When simulating this using an explicit ODE solver, it is a major task to invert the mass matrix to solve for the accelerations. When using implicit inlining, inverting the mass matrix can be avoided and the size of the non-linear system to be solved can be reduced from $3n$ to n. The approach is to iterate over the accelerations $\ddot{q}$ and use the the discretization formulas to calculate q and $\dot{q}$, and use $M(q)\ddot{q} - F(q, \dot{q})$ as the residual. This approach was presented in [1]. However, this method has had little practical use, because certain pragmas about the structure of the model equations had to be put into the model by the user.

The new structural analysis methods of Dymola automatically rediscovers well-known O(n) method by Luh, Walker, and Paul for calculating the joint forces and torques from the motion of the joints (q, $\dot{q}$ and $\ddot{q}$). Dymola is able to find this approach automatically without no hints or exploiting facts that it is a multi-body model. Dymola makes it by only analyzing the structure of the equations and manipulate them properly. The component models of the library ModelicAdditions.MultBody result typically in a hundred unknowns for each degree of freedom. Thus, it is far from trivial to transform an inlined model to this efficient form for numeric solution. Moreover, Dymola is able to find the core problem in more complex settings such as for a robot with drivelines and controllers. This is illustrated in the following application.

Application: Robotics model

Consider the model r3.robot in the Modelica [5] library ModelicaAdditions.MultiBody.Examples.Robots as shown in Figures 2 and 3.

The model describes an industrial robot with six degrees of freedom. The model is composed of basic mechanical components such as joints and bars as shown in part 3 of Figure 2. At every joint, a drive train as shown in part 4 of Figure 2 is present. Each drive train contains a motor, a gearbox and an actuator as well as a control system. The elasticity of the gears of the first three joints is modelled by one spring for each gearbox. The elasticity of the last three joints is neglected. In part 5 of Figure 2, the model of the motor and the actuator of one joint is shown. This component is defined, most naturally, as an electrical circuit. Finally, in Figure 3, the control system with tacho filters for one drive train is defined in block diagram format. To simplify the discussion, we omit potential locking in the joints due to bearing friction.

The model consists of 12 states for the mechanical part of the robot, two states for every gearbox with modeled elasticity, two states for every motor/actuator component, three states for every tacho filter, and three

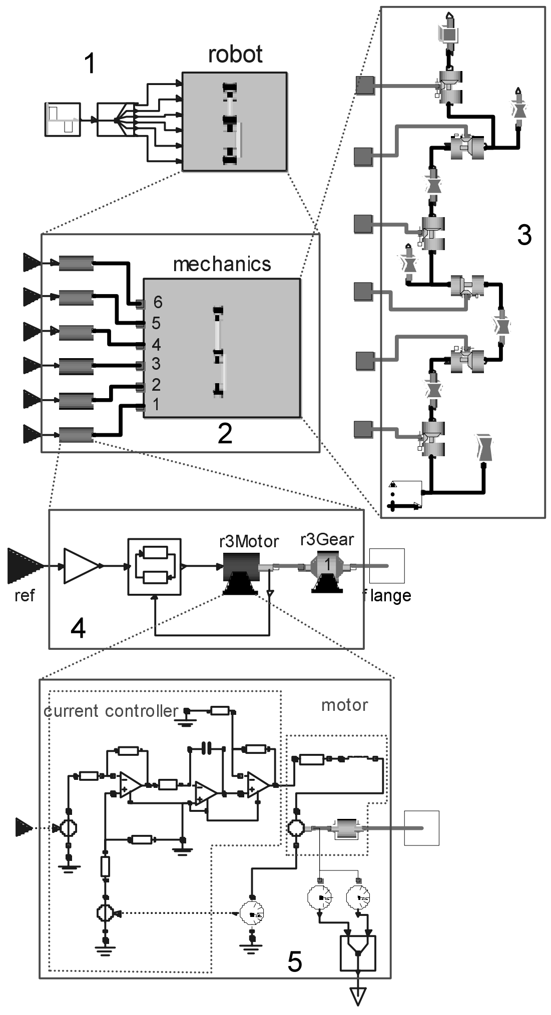

Figure 2: The robot model r3.

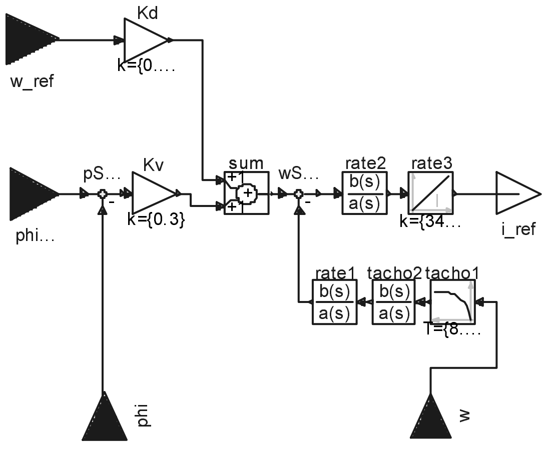

Figure 3: The controller of one joint.

states for every controller. The overall model has thus 12+3·2+6·(2+3+3) = 66 states. The simulation problem has additional 12 states for generating the reference path.

The model has 5963 unknowns. After Dymola's elimination of constant and alias variables at translation, 932 nontrivial and time-varying variables remain. For explicit methods there is one linear equation system to solve. It is of size six. It corresponds to the inversion of the mass matrix. Dymola has solved all other equation systems symbolically.

When using inlined explicit Euler, a step size of 0.05 ms has to be selected to achieve stable behavior. The paper [6] reports that the fastest eigenvalues of the linearization of the system are about 7000 in magnitude.

For the inlined implicit Euler, Dymola translates the simulation problem to a non-linear system of size 6 with no additional local equation systems. The equation systems from discretizing the drive trains and their controllers are linear and Dymola is able to solve them symbolically.

Table 1: Performance of the methods for the robot problem that is simulated for 1 s using a Pentium IV 1.6 GHz processor.

	Inl. Expl. Euler	**Inl. Impl. Euler**	**Inl. Impl. RK 3**	**Inl. Impl. RK 3**
Step size [ms]	0.05	1	5	10
Pos. error [mm]	0.1	3	0.1	0.4
Vel. error [mm/s]	5	20	5	20
CPU time [s]	1.97	0.16	0.11	0.06

The resulting execution times and maximum position and velocity errors compared to a reference solution calculated using DASSL are shown in Table 1. When judging the errors it may be of interest to know that the robot is of meter size and the maximum speeds are 2-4 m/s.

For easy interpretation of the execution times the problem was simulated for one second. It means that if the CPU time is less than one second, the simulation runs faster than real-time.

When using explicit Euler the simulation runs slower than real-time. The solution has good accuracy, but the computational burden is high. It is very interesting to see that the inlined third order implicit Runge-Kutta method gives a solution with the same accuracy only needing 6% of the effort for the explicit Euler method.

The implicit methods run all faster than real-time. As reported above Dymola is able to reduce the size of the non-linear system to six. Before the new improvements Dymola reduced the size of the non-linear system to 39 giving a CPU of 1.1 s for the simulation. Using the new approach the implicit Euler method needs only 0.16 s for the simulation. The simulation is speeded up more than six times and it runs faster than real-time.

The table shows that high order methods pay off. The third order Runge-Kutta method gives with less effort a better result than implicit Euler does.

If we let the robot model allow potential locking in the joints due to bearing friction when using the inlined third order implicit Runge-Kutta method with a step size of 5 ms, the CPU time needed is 0.16 s. Thus, this model runs also much faster then real-time.

Conclusions

This paper has described and illustrated Dymola's new approach to inlined implicit integration. The new features include more advanced analysis and manipulation of the inlined problem giving in many cases a drastic reduction of the non-linear problem that has to been solved numerically. Generation of analytic Jacobians also increases performance. Support of inline integration of higher order methods leads to better accuracy for larger steps. Thus allowing faster simulation.

Reported experiences of applying the new approach to simulation of an industrial robot have shown very promising results. The method has also been applied successfully to simulating hydraulic systems with long pipes exhibiting pressure wave oscillations.

Acknowledgements

This work was in parts supported by the European Commission under contract IST-199-11979 with Dynasim AB under the Information Societies Technology as the project entitled "Real-time simulation for design of multi-physics systems".

References

[1] D. Brück, H. Elmqvist, S.E. Mattsson, H. Olsson: *Dymola for Multi-Engineering Modeling and Simulation*, Proceedings of Modelica 2002. Modelica homepage: http://www.Modelica.org,

[2] Dymola. *Dynamic Modeling Laboratory*, Dynasim AB, Lund, Sweden, http://www.Dynasim.se

[3] H. Elmqvist, F. Cellier, M. Otter Inline Integration: A new mixed symbolic/numeric approach for solving differential-algebraic equation systems. Proceedings: European Simulation Multiconference June 1995 Prague, pp: XXIII-XXXIV.

[4] Hairer, Wanner: Solving Ordinary Differential Equations II, Springer Verlag

[5] Modelica. Modelica homepage: http://www.Modelica.org,

[6] Schiela, Olsson: Mixed-mode Integration for Real-Time Simulation, Modelica Workshop 2000, October 23-24, 2000, Lund University, Lund, Sweden