INLINING STEP-SIZE CONTROLLED FULLY IMPLICIT RUNGE-KUTTA ALGORITHMS FOR THE SEMI-ANALYTICAL AND SEMI-NUMERICAL SOLUTION OF STIFF ODES AND DAES

François E. Cellier, Ph.D.
Professor
Department of Electrical and Computer Engineering
University of Arizona
Tucson, AZ 85721-0104
U.S.A.
Cellier@ECE.Arizona.Edu
http://www.ece.arizona.edu/~cellier/

Traditionally, continuous system simulation software was designed for the simulation of dynamical systems described by state-space models, i.e., a set of first-order ordinary differential equations (ODEs). For the numerical solution, explicit integration algorithms, usually 4^{th}-order accurate explicit Runge-Kutta algorithms were used. The goal was to strictly separate the model description from the numerical solution. In this way, no iterations were necessary at all, and the simulation software could toggle between calls to the numerical solver on the one hand, and calls to the program that captured the state-space model on the other.

The approach was suitable as long as scientists and engineers were content to solve low-order models of simple devices. As the models grew in complexity, the range of time constants to be captured by the models spread, i.e., the models became invariably stiff. Explicit ODE solvers were no longer suitable for the task, because now, their step sizes were controlled by the numerical stability requirements rather than by accuracy requirements. Implicit integration algorithms had to be used. Among those, the most widely used algorithms were a class of linear multi-step methods commonly referred to as Backward Difference Formulae (BDF). Of those, the currently most robust code is DASSL. Using such techniques, a Newton iteration has to be performed during each step, whereby the Jacobian matrix to be inverted is of size n^2, where n is the order of the model, i.e., denotes the number of state equations. A full-blown Newton iteration is necessary, because any simplification, such as a fixed-point iteration destroys the numerical stability properties of the BDF method.

Unfortunately, physical systems do not lend themselves easily to state-space descriptions. The state-space approach to modeling is an invention of control engineers, who ensure by design that their models can be described easily in this form. In general, the equations governing physical systems are algebraically coupled, not because of neglected fast time

constants, but simply, because physics is essentially acausal, i.e., the equations describing physical systems are always simultaneous equations.

It is very difficult to manually translate a non-linear circuit containing several transistors into an explicit state-space form. Manually translating the equations that describe a six-degree-of-freedom robot into such a form is totally hopeless. Similarly, a model describing an industrial distillation column contains hundreds or thousands of coupled simultaneous equations that cannot be converted manually to an explicit state-space form. In general, physical systems are governed by a set of implicitly described differential equations with algebraic couplings between the state derivatives and often even between the states themselves, i.e., physical systems are usually governed by higher-index Differential Algebraic Equation (DAE) systems.

The couplings between states are best dealt with at compile time by symbolically differentiating the constraint equations while reducing the number of states, until no constraints exist any longer among the remaining state variables. This process can be automated. There exist numerically efficient algorithms, such as the algorithm by *Pantelides*, that is of linear computational complexity, to analytically reduce the perturbation index of the DAE system to one, i.e., to get rid of the constraints between states.

The resulting index-one DAE system can be numerically solved using DASSL. However, the algebraic coupling equations need to be iterated as well, i.e., each algebraic coupling leads to an additional term in the Jacobian matrix. The resulting Jacobian will now be of size $(n+w)^2$, where n denotes the number of states, and w denotes the number of algebraic couplings.

It is possible to automatically convert the resulting index-one DAE system to an explicit ODE form using the algorithm by *Tarjan*. However, this transition is not always harmless. The reason is that the implicit DAE equations are usually sparse. In the process of inverting the equations, the sparsity may be lost.

Inlining is a technique that merges the model equations with the equations describing the integration algorithm, thereby converting the differential equations to a set of equivalent difference equations. The process of inlining can be automated, i.e., results in additional symbolic formulae manipulations at compile time. The user can still describe his model in terms of a higher-index DAE system. All of the necessary transformations occur in a completely automated fashion at compile time, resulting in a set of implicitly coupled difference equations. Simulation now simply means to apply Newton iteration to the set of difference equations at each time step. The Jacobian is still of size $(n+w)^2$, i.e., nothing has been gained so far.

It may not be necessary to iterate over all these variables. A *tearing* algorithm can be used to find a minimum number of so-called tearing variables, i.e., iteration variables, thereby reducing the size of the Jacobian matrix dramatically. Unfortunately, complete tearing is an np-complete problem, i.e., the algorithm is of exponential computational

complexity. However, there exist good heuristics of linear computational complexity that will find a small set of tearing variables, though not necessarily the smallest one. Notice however that tearing can only be applied after inlining, i.e., the algorithm can only be applied to the algebraically coupled set of difference equations, not to the index-one DAE system. Hence, while inlining by itself does not help generate more efficient simulation equations, inlining with subsequent tearing usually improves the efficiency of the simulation dramatically.

Attractive alternatives to DASSL provide implicit Runge-Kutta (IRK) algorithms, such as Radau-IIa. IRKs are attractive, because they allow larger step sizes to be taken, thereby reducing the number of Newton iterations needed during simulation. Yet, these algorithms have not been widely used, because they are more difficult to implement than the BDF algorithms. The paper will show how inlining leads to elegant implementations of IRK algorithms.

An additional difficulty relates to the step-size control of these algorithms. Explicit RK algorithms embed a lower-order algorithm within the higher-order algorithm to have a second approximation to compare to. For example, Runge-Kutta Fehlberg 4/5 (RKF4/5) embeds a 4^{th}-order accurate algorithm within a 5^{th}-order accurate algorithm, and compares the two approximations obtained in this way for the purpose of step-size control. BDF algorithms have elegant order-control algorithms built in. They also use the embedding technique to embed an entire series of algorithms of different orders, switching between them in order to maximize the efficiency. Step-size control in these algorithms is less elegant and more costly, but can be done adequately using the Nordsieck vector approach.

In contrast, no good step-size control mechanisms have been known for IRKs. Finding embedding IRKs is a very difficult problem, and integrating two completely separate algorithms in parallel is hopelessly inefficient. Hairer proposed an explicit embedding algorithm to accompany the IRK for step-size control purposes, arguing that since the IRK will be propagated to the next step, numerical stability considerations on the companion algorithm are not as stringent. Unfortunately, eigenvalues of the Jacobian with large negative real parts may lead to overly conservative error estimates that would still restrict the step size unnecessarily. Hairer knew about this problem and proposed a "filter" that would dampen out the contribution of these eigenvalues. Essentially, he now has an implicit companion method, but he uses a fixed-point iteration that he applies once in order to improve the error estimate. Unfortunately, fixed-point iteration still destroys the stability properties.

In this paper, a fully-implicit stiffly-stable companion method to Radau-IIa is shown that can be computed almost for free, and that does not share the problems of Hairer's approach. The companion embeds another polynomial approximation, but does so making use of the available information over the last two steps, i.e., the approach is a hybrid approach: the integration algorithm itself is a single-step IRK, whereas the companion method is a two-step linear polynomial approximation of one order higher than the IRK. Whereas the technique will be shown for the 3^{rd}-order and 5^{th}-order

accurate Radau-IIa algorithms, the technique can just as easily be applied to Lobatto-IIIc and other IRK algorithms. Inlining helps to make the implementation of the step-size control algorithm simple and elegant, and tearing can be used on the resulting set of algebraically coupled difference equations to reduce the size of the Jacobian to obtain optimal run-time efficiency.