Numerical Simulation of Dynamic Systems XIX

Prof. Dr. François E. Cellier
Department of Computer Science
ETH Zurich

April 30, 2013

RK Algorithms

Let us now discuss, how we may implement *DAE versions* of *RK algorithms*.

For example, we can write the classical RK4 algorithm in DAE form:

$$\mathbf{f}\left(\mathbf{x_k}, \mathbf{k_1}, \mathbf{u}(t_k), t_k\right) = 0.0$$

$$\mathbf{f}\left(\mathbf{x_k} + \frac{h}{2}\mathbf{k_1}, \mathbf{k_2}, \mathbf{u}(t_k + \frac{h}{2}), t_k + \frac{h}{2}\right) = 0.0$$

$$\mathbf{f}\left(\mathbf{x_k} + \frac{h}{2}\mathbf{k_2}, \mathbf{k_3}, \mathbf{u}(t_k + \frac{h}{2}), t_k + \frac{h}{2}\right) = 0.0$$

$$\mathbf{f}\left(\mathbf{x_k} + h\mathbf{k_3}, \mathbf{k_4}, \mathbf{u}(t_k + h), t_k + h\right) = 0.0$$

$$\mathbf{x_{k+1}} = \mathbf{x_k} + \frac{h}{6}\left(\mathbf{k_1} + 2\mathbf{k_2} + 2\mathbf{k_3} + \mathbf{k_4}\right)$$

RK Algorithms II

- Every stage of the RK4 algorithm is converted to a Newton iteration.
- In the case of an implicit model, the ODE and DAE formulations of the algorithm are practically identical.
- In the case of an explicit model, the DAE formulation turns out to be excessively expensive.
- **In general, it often pays off to convert an implicit non-stiff model symbolically to an explicit form in the process of model compilation to avoid iterations. If this is not possible due to the non-linear nature of some of the algebraic loops, we should at least minimize the sizes of the algebraic loops.**

The Radau IIA Algorithms

There exist other classes of *stiffly-stable implicit RK algorithms* that we haven't considered yet.

One such class of algorithms is the class of the *Radau IIA algorithms*.

The 3^{rd}-order accurate Radau IIA algorithm is characterized by the Butcher tableau:

1/3	5/12	-1/12
1	3/4	1/4
x	3/4	1/4

It is a *3^{rd}-order accurate implicit RK algorithm in two stages*. The *Radau IIA(3) algorithm* can be formulated as an *ODE algorithm* in the form:

$$\mathbf{k_1} = \mathbf{f}\left(\mathbf{x_k} + \frac{5h}{12}\mathbf{k_1} - \frac{h}{12}\mathbf{k_2}, \mathbf{u}(t_k + \frac{h}{3}), t_k + \frac{h}{3}\right)$$

$$\mathbf{k_2} = \mathbf{f}\left(\mathbf{x_k} + \frac{3h}{4}\mathbf{k_1} + \frac{h}{4}\mathbf{k_2}, \mathbf{u}(t_k + h), t_k + h\right)$$

$$\mathbf{x_{k+1}} = \mathbf{x_k} + \frac{h}{4}\left(3\mathbf{k_1} + \mathbf{k_2}\right)$$

The Radau IIA Algorithms II

The *Radau IIA(3) algorithm* can be formulated as a *DAE algorithm* in the form:

$$\mathbf{f}\left(\mathbf{x_k} + \frac{5h}{12}\mathbf{k_1} - \frac{h}{12}\mathbf{k_2}, \mathbf{k_1}, \mathbf{u}(t_k + \frac{h}{3}), t_k + \frac{h}{3}\right) = 0.0$$
$$\mathbf{f}\left(\mathbf{x_k} + \frac{3h}{4}\mathbf{k_1} + \frac{h}{4}\mathbf{k_2}, \mathbf{k_2}, \mathbf{u}(t_k + h), t_k + h\right) = 0.0$$
$$\mathbf{x_{k+1}} = \mathbf{x_k} + \frac{h}{4}\left(3\mathbf{k_1} + \mathbf{k_2}\right)$$

- The two formulations are practically identical. In both cases, we require a single Newton iteration that spans over both stages of the algorithm. The iteration variables are the two vectors $\mathbf{k_1}$ and $\mathbf{k_2}$.
- In order for the Newton iteration to converge, we need good initial guesses of $\mathbf{k_1}$ and $\mathbf{k_2}$. We use: $\mathbf{k_1} = \mathbf{k_2} = \dot{\mathbf{x}}(t_0)$, i.e., the user needs to provide at least a *guess for the state derivatives at time zero* beside from providing the *initial conditions on the state variables* themselves.
- The same is also true for the DAE versions of the AM algorithms, although we failed to mention this fact in the previous presentation.

The Radau IIA Algorithms III

Let us now find the *numerical stability domain* of the *Radau IIA(3) algorithm*. Since it doesn't matter, whether we use the ODE or the DAE version, I prefer to go with the ODE version, as this turns out to be a bit simpler.

We can write:

$$\mathbf{k_1} = \mathbf{A}\left(\mathbf{x_k} + \frac{5h}{12}\mathbf{k_1} - \frac{h}{12}\mathbf{k_2}\right)$$
$$\mathbf{k_2} = \mathbf{A}\left(\mathbf{x_k} + \frac{3h}{4}\mathbf{k_1} + \frac{h}{4}\mathbf{k_2}\right)$$
$$\mathbf{x_{k+1}} = \mathbf{x_k} + \frac{h}{4}\left(3\mathbf{k_1} + \mathbf{k_2}\right)$$

Solving for the unknowns $\mathbf{k_1}$ and $\mathbf{k_2}$:

$$\mathbf{k_1} = \left[\mathbf{I^{(n)}} - \frac{2\mathbf{A}h}{3} + \frac{(\mathbf{A}h)^2}{6}\right]^{-1} \cdot \left(\mathbf{I^{(n)}} - \frac{\mathbf{A}h}{3}\right) \cdot \mathbf{A} \cdot \mathbf{x_k}$$
$$\mathbf{k_2} = \left[\mathbf{I^{(n)}} - \frac{2\mathbf{A}h}{3} + \frac{(\mathbf{A}h)^2}{6}\right]^{-1} \cdot \left(\mathbf{I^{(n)}} + \frac{\mathbf{A}h}{3}\right) \cdot \mathbf{A} \cdot \mathbf{x_k}$$
$$\mathbf{x_{k+1}} = \mathbf{x_k} + \frac{h}{4}\left(3\mathbf{k_1} + \mathbf{k_2}\right)$$

The Radau IIA Algorithms IV

Therefore:

$$\mathbf{F} = \mathbf{I^{(n)}} + \left[\mathbf{I^{(n)}} - \frac{2\mathbf{A}h}{3} + \frac{(\mathbf{A}h)^2}{6}\right]^{-1} \cdot \left(\mathbf{I^{(n)}} - \frac{\mathbf{A}h}{6}\right) \cdot (\mathbf{A}h)$$

Developing into a Taylor series around $h = 0.0$, we obtain:

$$\mathbf{F} \approx \mathbf{I^{(n)}} + \mathbf{A}h + \frac{(\mathbf{A}h)^2}{2} + \frac{(\mathbf{A}h)^3}{6} + \frac{(\mathbf{A}h)^4}{36}$$

Consequently, the *error coefficient* is:

$$\varepsilon = \frac{1}{72}(\mathbf{A}h)^4$$

The Radau IIA Algorithms V

There exists another Radau IIA algorithm characterized by the Butcher tableau:

$\frac{4-\sqrt{6}}{10}$	$\frac{88-7\sqrt{6}}{360}$	$\frac{296-169\sqrt{6}}{1800}$	$\frac{-2+3\sqrt{6}}{225}$
$\frac{4+\sqrt{6}}{10}$	$\frac{296+169\sqrt{6}}{1800}$	$\frac{88+7\sqrt{6}}{360}$	$\frac{-2-3\sqrt{6}}{225}$
1	$\frac{16-\sqrt{6}}{36}$	$\frac{16+\sqrt{6}}{36}$	$\frac{1}{9}$
x	$\frac{16-\sqrt{6}}{36}$	$\frac{16+\sqrt{6}}{36}$	$\frac{1}{9}$

This is a *stiffly-stable 5^{th}-order accurate implicit RK algorithm in three stages* with the $\mathbf{F}$-matrix:

$$\mathbf{F} = \mathbf{I^{(n)}} + \left[\mathbf{I^{(n)}} - \frac{3\mathbf{A}h}{5} + \frac{3(\mathbf{A}h)^2}{20} - \frac{(\mathbf{A}h)^3}{60}\right]^{-1} \cdot \left(\mathbf{I^{(n)}} - \frac{\mathbf{A}h}{10} + \frac{(\mathbf{A}h)^2}{60}\right) \cdot \mathbf{A}h$$

Developing into a Taylor series around $h = 0.0$, we obtain:

$$\mathbf{F} \approx \mathbf{I^{(n)}} + \mathbf{A}h + \frac{(\mathbf{A}h)^2}{2} + \frac{(\mathbf{A}h)^3}{6} + \frac{(\mathbf{A}h)^4}{24} + \frac{(\mathbf{A}h)^5}{120} + \frac{11(\mathbf{A}h)^6}{7200}$$

Consequently, the *error coefficient* is:

$$\varepsilon = \frac{1}{7200}(\mathbf{A}h)^6$$

The Lobatto IIIC Algorithm

There exists yet another IRK algorithm of a different class characterized by the Butcher tableau:

0	1/6	-1/3	1/6
1/2	1/6	5/12	-1/12
1	1/6	2/3	1/6
x	1/6	2/3	1/6

The *Lobatto IIIC algorithm* is a *stiffly-stable 4^{th}-order accurate implicit RK algorithm in three stages* with the **F**-matrix:

$$\mathbf{F} = \mathbf{I}^{(n)} + \left[\mathbf{I}^{(n)} - \frac{3\mathbf{A}h}{4} + \frac{(\mathbf{A}h)^2}{4} - \frac{(\mathbf{A}h)^3}{24}\right]^{-1} \cdot \left(\mathbf{I}^{(n)} - \frac{\mathbf{A}h}{4} + \frac{(\mathbf{A}h)^2}{24}\right) \cdot \mathbf{A}h$$

Developing into a Taylor series around $h = 0.0$, we obtain:

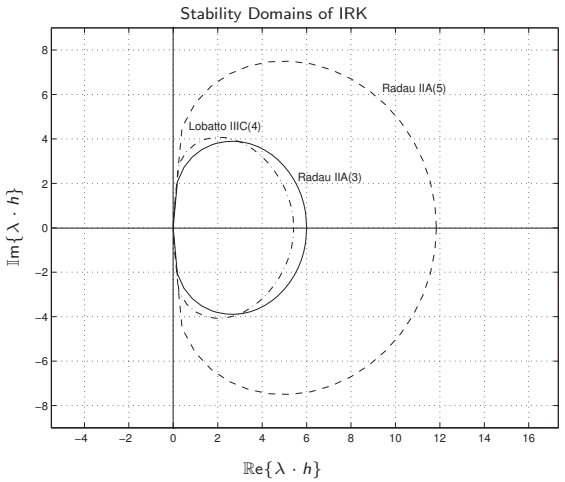
$$\mathbf{F} \approx \mathbf{I}^{(n)} + \mathbf{A}h + \frac{(\mathbf{A}h)^2}{2} + \frac{(\mathbf{A}h)^3}{6} + \frac{(\mathbf{A}h)^4}{24} + \frac{(\mathbf{A}h)^5}{96}$$

Consequently, the *error coefficient* is:

$$\varepsilon = \frac{1}{480}(\mathbf{A}h)^5$$

Numerical Stability Domains of the IRK Algorithms

We can now plot the *numerical stability domains* of the three algorithms *Radau IIA(3)*, *Lobatto IIIC(4)*, and *Radau IIA(5)*.

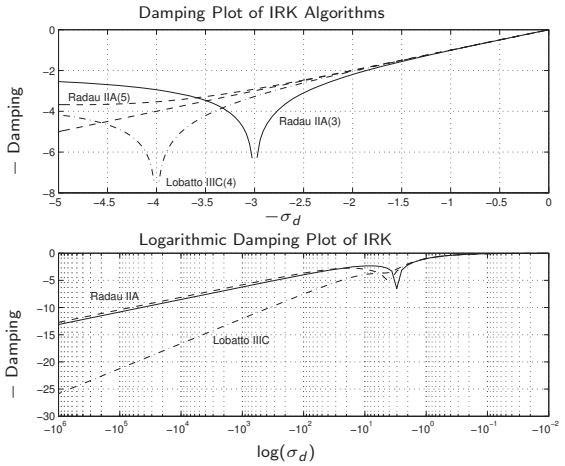


Numerical Stability Domains of the IRK Algorithms II

- All three methods are *A-stable*, a feature that we have not been able to get for higher-order stiffly-stable linear multi-step methods.
- As we expect from Runge-Kutta methods, the unstable regions in the right-half plane grow rather than shrink for higher orders, i.e., not only does the border of stability approximate the imaginary axis better in the vicinity of the origin, but it does so also further up and down the imaginary axis.
- The stability domains of all three algorithms look quite promising.

Damping Plots of the IRK Algorithms

Let us now draw the *damping plots* of the three algorithms *Radau IIA(3)*, *Lobatto IIIC(4)*, and *Radau IIA(5)*.



Damping Plots of the IRK Algorithms II

- All three methods are *L-stable*. The Lobatto IIIC algorithm has slightly better damping characteristics than the two Radau IIA algorithms for poles far out in the left-half complex plane.
- All three methods exhibit large asymptotic regions, therefore allowing large step sizes to be used.
- Radau IIA(3) calls for a Newton iteration spanning over two stages of the algorithm, whereas both Radau IIA(5) and Lobatto IIIC call for a Newton iteration spanning over three stages of the algorithm. Consequently, they will require two and three times more function evaluations per step than the BDF algorithms. Yet, the much larger step sizes than these algorithms allow us to use may well offset the additional cost. Consequently, the IRK algorithms are expected to be quite competitive.
- The IRK algorithms are single-step methods, i.e., they are *self-starting*, and *step-size control* in these methods is much cheaper than in the multi-step methods. We can therefore expect that these algorithms will beat the BDF methods by leaps and bounds when dealing with highly non-linear problems that call for frequent step-size adjustments, or when dealing with problems with frequent discontinuities.

The DASSL Code

DASSL is *one of the most successfull simulation codes* on the market today.

DASSL implements the BDF formulae of orders 1..5, i.e., it is a *variable-step, variable-order BDF code*.

DASSL was developed for the simulation of implicit models. It offers two user interfaces, an *ODE interface*, and a *DAE interface*.

DASSL is the *default simulator* used by the modeling and simulation (M&S) environment **Dymola**, i.e., if the user doesn't select another simulation code explicitly, Dymola chooses DASSL for the simulation of the model.

Dymola makes always use of the ODE interface of DASSL, i.e., it converts the model symbolically to explicit ODE form before calling upon the simulation engine.

The DASSL Code II

Dymola has a preference for DASSL in spite of the code's *inefficiency in handling non-stiff models*.

Dymola was developed for the *simulation of large-scale models*. Almost all large-scale models are inherently stiff, as they focus simultaneously on fast and slow phenomena.

Dymola was developed for users who are not knowledgeable of numerical algorithms. The large majority of Dymola users knows little if anything about numerical ODE solvers. Also, they wouldn't usually know whether their models are stiff or not. As stiff system solvers can deal (albeit less efficiently) with non-stiff models, whereas non-stiff system solvers cannot deal at all with stiff models, the use of DASSL promotes *robustness of the M&S environment*.

Today's computers are so powerful that consideration of optimal simulation efficiency has lost much of its former importance when dealing with typical engineering problems. An exception are, of course, systems with distributed parameters.

The DASSL Code III

Dymola has a preference for DASSL in spite of the code's *inefficiency in handling non-linear models*.

Non-linear models and in particular *models with discontinuities* require *frequent changes in the integration step size*. Since step-size changes are expensive in multi-step algorithms, DASSL isn't optimally suited for the simulation of systems with discontinuities.

Almost all engineering systems are described by models with heavy and frequent discontinuities.

It would therefore be preferable to use a simulation engine that is based on a *Radau IIA* algorithm.

A fairly robust Radau IIA code has become available recently. Dymola offers that code as one of its simulation engines, but the producers of Dymola haven't made this the default simulation engine yet. Radau IIA turns out to be often more efficient than DASSL, but the Dymola developers don't trust the code's robustness sufficiently yet to elevate it to the level of their default algorithm. This will probably happen in one of the next releases.

The DASSL Code IV

Let us compare once more, in a little more detail, the *difference between the ODE and DAE formulations* of the BDF algorithms.

We begin with the linear explicit system:

$$\dot{\mathbf{x}} = \mathbf{A} \cdot \mathbf{x} + \mathbf{B} \cdot \mathbf{u}$$

Using the BDF3 algorithm in its ODE form:

$$\mathbf{x}_{k+1}^{\mathbf{BDF3}} = \frac{6}{11} h \cdot \dot{\mathbf{x}}_{k+1} + \frac{18}{11}\mathbf{x}_k - \frac{9}{11}\mathbf{x}_{k-1} + \frac{2}{11}\mathbf{x}_{k-2}$$

we eliminate the derivatives by inserting the model equations into the solver equations:

$$\mathbf{x}_{k+1}^{\mathbf{BDF3}} = \frac{6}{11} \cdot \mathbf{A} \cdot h \cdot \mathbf{x}_{k+1} + \frac{6}{11} \cdot \mathbf{B} \cdot h \cdot \mathbf{u}_{k+1} + \frac{18}{11}\mathbf{x}_k - \frac{9}{11}\mathbf{x}_{k-1} + \frac{2}{11}\mathbf{x}_{k-2}$$

We formulate the Newton iteration in the following form:

$$\mathcal{F}(\mathbf{x}_{k+1})^{\mathbf{ODE}} = \mathbf{x}_{k+1} - \mathbf{x}_{k+1}^{\mathbf{BDF3}} = 0.0$$

where $\mathbf{x}_{k+1}^{\mathbf{BDF3}}$ is the known solution approximated by the solver, whereas $\mathbf{x}_{k+1}$ is the unknown analytical solution.

The DASSL Code V

With:

$$\mathcal{F}(\mathbf{x}_{k+1})^{\mathbf{ODE}} = \mathbf{x}_{k+1} - \mathbf{x}_{k+1}^{\mathbf{BDF3}}$$

we can calculate the Hessian of the Newton iteration:

$$\mathcal{H}(\mathbf{x}_{k+1})^{\mathbf{ODE}} = \frac{\partial \mathcal{F}(\mathbf{x}_{k+1})^{\mathbf{ODE}}}{\partial \mathbf{x}_{k+1}} = \mathbf{I}^{(\mathbf{n})} - \frac{6}{11} \cdot \mathbf{A} \cdot h$$

In the case of a non-linear system:

$$\mathcal{H}(\mathbf{x}_{k+1})^{\mathbf{ODE}} = \mathbf{I}^{(\mathbf{n})} - \frac{6}{11} \cdot \mathcal{J} \cdot h$$

where $\mathcal{J}$ is the Jacobian matrix of the system:

$$\mathcal{J}(\mathbf{x}_{k+1}) = \frac{\partial \mathbf{f}(\mathbf{x}_{k+1})}{\partial \mathbf{x}_{k+1}}$$

The DASSL Code VI

What happens in the case of the DAE formulation?

We use the BDF algorithm solved for the derivative:

$$\dot{\mathbf{x}}_{k+1}^{\mathbf{BDF3}} = \frac{1}{h}\left[\frac{11}{6} \cdot \mathbf{x}_{k+1} - 3\mathbf{x}_k + \frac{3}{2}\mathbf{x}_{k-1} - \frac{1}{3}\mathbf{x}_{k-2}\right]$$

We now formulate the Newton iteration in the following form:

$$\mathcal{F}(\mathbf{x}_{k+1})^{\mathbf{DAE}} = \dot{\mathbf{x}}_{k+1}^{\mathbf{model}} - \dot{\mathbf{x}}_{k+1}^{\mathbf{BDF3}} = 0.0$$

where $\dot{\mathbf{x}}_{k+1}^{\mathbf{BDF3}}$ is the derivative approximated by the solver, whereas $\dot{\mathbf{x}}_{k+1}^{\mathbf{model}}$ is the derivative obtained using the model equations.

Therefore:

$$\mathcal{H}(\mathbf{x}_{k+1})^{\mathbf{DAE}} = \frac{\partial \mathcal{F}(\mathbf{x}_{k+1})^{\mathbf{DAE}}}{\partial \mathbf{x}_{k+1}} = \mathbf{A} - \frac{11}{6h} \cdot \mathbf{I}^{(\mathbf{n})}$$

In the case of a non-linear system:

$$\mathcal{H}(\mathbf{x}_{k+1})^{\mathbf{DAE}} = \mathcal{J} - \frac{11}{6h} \cdot \mathbf{I}^{(\mathbf{n})}$$

The DASSL Code VII

What happens if we use very small integration steps?

in the case of the *ODE formulation*:

$$\lim_{h \to 0} \mathcal{H}(\mathbf{x}_{k+1})^{\mathbf{ODE}} = \lim_{h \to 0}\left(\mathbf{I}^{(\mathbf{n})} - \frac{6}{11} \cdot \mathcal{J} \cdot h\right) = \mathbf{I}^{(\mathbf{n})}$$

In the case of the *DAE formulation*:

$$\lim_{h \to 0} \mathcal{H}(\mathbf{x}_{k+1})^{\mathbf{DAE}} = \lim_{h \to 0}\left(\mathcal{J} - \frac{11}{6h} \cdot \mathbf{I}^{(\mathbf{n})}\right) \to \infty$$

In the DAE formulation with very small integration step sizes, the Hessian of the Newton iteration is very sensitive to step-size changes. This makes us suspect that we might have numerical difficulties with the algorithm.

The DASSL Code VIII

What happens in the case of an implicit model?

Let us consider once more the various BDF formulae:

$$\mathbf{x_{k+1}} = h \cdot \mathbf{f_{k+1}} + \mathbf{x_k}$$
$$\mathbf{x_{k+1}} = \frac{2}{3} \cdot h \cdot \mathbf{f_{k+1}} + \frac{4}{3} \cdot \mathbf{x_k} - \frac{1}{3} \cdot \mathbf{x_{k-1}}$$
$$\mathbf{x_{k+1}} = \frac{6}{11} \cdot h \cdot \mathbf{f_{k+1}} + \frac{18}{11} \cdot \mathbf{x_k} - \frac{9}{11} \cdot \mathbf{x_{k-1}} + \frac{2}{11} \cdot \mathbf{x_{k-2}}$$
etc.

We can generalize these formulae:

$$\mathbf{x_{k+1}} = \bar{h} \cdot \mathbf{f_{k+1}} + \mathrm{pre}(\mathbf{x})$$

where $\bar{h}$ is a *normalized step size* proportional to the real step size h, and $\mathrm{pre}(\mathbf{x})$ is a function of information from the past that doesn't influence the Newton iteration.

Therefore:

$$\mathbf{f_{k+1}} = \frac{\mathbf{x_{k+1}} - \mathrm{pre}(\mathbf{x})}{\bar{h}}$$

The DASSL Code IX

Let us now consider the implicit model:

$$\mathbf{f}(\mathbf{x}, \dot{\mathbf{x}}, \mathbf{u}, t) = 0.0$$

We can eliminate the derivatives by inserting the solver formula into the implicit model:

$$\mathcal{F}(\mathbf{x_{k+1}}) = \mathbf{f}(\mathbf{x_{k+1}}, \frac{\mathbf{x_{k+1}} - \mathrm{pre}(\mathbf{x})}{\bar{h}}, \mathbf{u_{k+1}}, t_{k+1}) = 0.0$$

In reality, it isn't necessary to physically insert the solver formula in the model, as we can analyze very easily the effects of such an insertion.

We can write:

$$\mathcal{H}(\mathbf{x_{k+1}}) = \mathcal{J}_{\mathbf{x}}(\mathbf{x_{k+1}}) + \frac{1}{\bar{h}} \cdot \mathcal{J}_{\dot{\mathbf{x}}}(\mathbf{x_{k+1}})$$

where:

$$\mathcal{J}_{\mathbf{x}}(\mathbf{x_{k+1}}) = \left.\frac{\partial \mathbf{f}}{\partial \mathbf{x}}\right|_{x=x_{k+1}, \dot{x}=\dot{x}_{k+1}}$$
$$\mathcal{J}_{\dot{\mathbf{x}}}(\mathbf{x_{k+1}}) = \left.\frac{\partial \mathbf{f}}{\partial \dot{\mathbf{x}}}\right|_{x=x_{k+1}, \dot{x}=\dot{x}_{k+1}}$$

The DASSL Code X

We can implement the Newton iteration in the following form:

$$\left(\mathcal{J}_{\mathbf{x}} + \frac{1}{\bar{h}} \cdot \mathcal{J}_{\dot{\mathbf{x}}}\right) \cdot \delta^{\ell} = \mathbf{f}(\mathbf{x}^{\ell}, \dot{\mathbf{x}}^{\ell}, \mathbf{u}, t)$$
$$\mathbf{x}^{\ell+1} = \mathbf{x}^{\ell} - \delta^{\ell}$$
$$\dot{\mathbf{x}}^{\ell+1} = \dot{\mathbf{x}}^{\ell} - \frac{1}{\bar{h}} \cdot \delta^{\ell}$$

or a bit better:

$$(\bar{h} \cdot \mathcal{J}_{\mathbf{x}} + \mathcal{J}_{\dot{\mathbf{x}}}) \cdot \delta^{\ell} = \bar{h} \cdot \mathbf{f}(\mathbf{x}^{\ell}, \dot{\mathbf{x}}^{\ell}, \mathbf{u}, t)$$
$$\mathbf{x}^{\ell+1} = \mathbf{x}^{\ell} - \delta^{\ell}$$
$$\dot{\mathbf{x}}^{\ell+1} = \dot{\mathbf{x}}^{\ell} - \frac{1}{\bar{h}} \cdot \delta^{\ell}$$

In every iteration step, we must solve a linear system of equations.

The DASSL Code XI

What happens in the case of very small integration steps?

We notice that:

$$\lim_{\bar{h} \to 0} \left(\bar{h} \cdot \mathcal{J}_{\mathbf{x}} + \mathcal{J}_{\dot{\mathbf{x}}}\right) = \mathcal{J}_{\dot{\mathbf{x}}}$$

However, we already know from the previous presentation that the $\mathcal{J}_{\dot{\mathbf{x}}}$-matrix is *singular* in the case of *higher index models*, because in the linear case:

$$\mathbf{A} \cdot \mathbf{x} + \mathbf{B} \cdot \dot{\mathbf{x}} = 0.0$$

we know that $\mathcal{J}_{\mathbf{x}} = \mathbf{A}$ and $\mathcal{J}_{\dot{\mathbf{x}}} = \mathbf{B}$.

DASSL can in general not be used for the simulation of higher-index DAE models.

The DASSL Code XII

Let us now assume that we are dealing with a model consisting of n differential equations and k algebraic equations. The model:

$$\mathbf{f}(\mathbf{x}, \dot{\mathbf{x}}, \mathbf{u}, t) = 0.0$$

contains $n + k$ equations in $n + k$ variables.

This is not a higher-index model, and yet, the $\mathcal{J}_{\dot{\mathbf{x}}}$-matrix is singular.

DASSL has also problems with the simulation of index-1 DAE systems with small integration step sizes.

The DASSL Code XIII

For these reasons:

- **Dymola** doesn't employ DASSL as *DAE algorithm*, but rather as *ODE algorithm*.
- During the model compilation, Dymola first reduces the perturbation index to 1 using the *Pantelides algorithm*.
- Thereafter, Dymola identifies the iteration variables necessary to solve all *algebraic loops*.
- To this end, Dymola uses the *Tarjan algorithm* to isolate the individual algebraic loops and minimize their sizes.
- Afterwards, Dymola uses the *tearing algorithm* to determine a small number of iteration variables.
- During the simulation, Dymola uses DASSL as default *ODE algorithm*. DASSL iterates over the n state variables.
- Inside each iteration step, Dymola evaluates the model. In the model description, we may encounter small *algebraic loops*. Every one of them invokes a Newton iteration to determine the current values of the *tearing variables*.

Inline Integration

Until now, we have always distinguished carefully between the *model equations* on the one hand, and the *solver equations* on the other. The reasons for this distinction are historical. Most users of simulation software know little if anything about numerical methods, and consequently wish to use the simulation engine as a black box. The less they need to know about the solver, the happier they are.

Yet, it turns out that this separation can lead to inefficient simulations. For this reason, we shall now let go of this artificial constraint. Modern model compilers are able to protect the user from having to know much about the solver and yet, allow to generate more efficient simulation code.

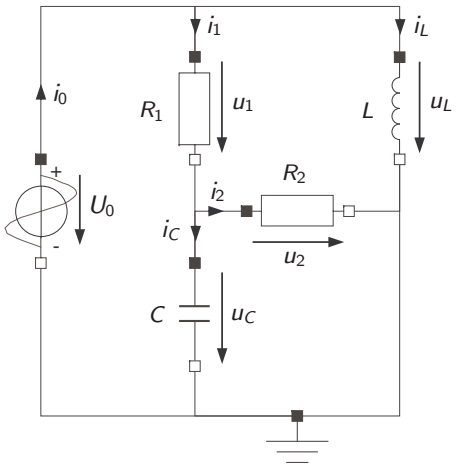
Inline integration merges the two types of equations. The solver equations are simply added to the model equations before causalization is attempted.

In this way, the *system of differential and algebraic equations* is converted to a *system of difference and algebraic equations*.

We shall start by *inlining BDF algorithms*.

Inline Integration II

We start with an example. To this end, we return once more to our already standard electrical circuit.



$$\begin{aligned}
u_0 &= f(t)\\
u_1 &= R_1 \cdot i_1\\
u_2 &= R_2 \cdot i_2\\
u_L &= L \cdot di_L\\
i_C &= C \cdot du_C\\
u_0 &= u_1 + u_C\\
u_L &= u_1 + u_2\\
u_C &= u_2\\
i_0 &= i_1 + i_L\\
i_1 &= i_2 + i_C\\
i_L &= \mathrm{pre}(i_L) + \bar{h} \cdot di_L\\
u_C &= \mathrm{pre}(u_C) + \bar{h} \cdot du_C
\end{aligned}$$

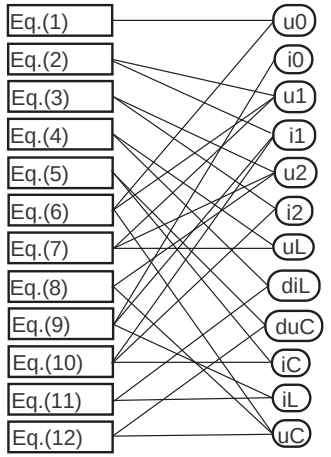
Inline Integration III

- We inlined the BDF formulae in their generic form using the normalized step size $\bar{h}$.
- The two state derivatives have become algebraic variables.
- Instead of dealing with 10 differential and algebraic equations in 10 unknowns, we now deal with 12 purely algebraic equations in 12 unknowns.
- The former formulation allowed us in this example to causalize the set of 10 equations directly. The new formulation contains an algebraic loop.

Inline Integration IV

The model is characterized by the following structure digraph:

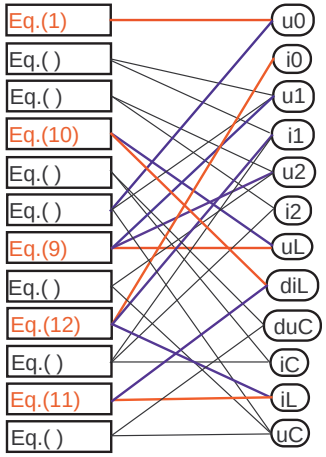
$$
\begin{aligned}
u_0 &= f(t)\\
u_1 &= R_1 \cdot i_1\\
u_2 &= R_2 \cdot i_2\\
u_L &= L \cdot di_L\\
i_C &= C \cdot du_C\\
u_0 &= u_1 + u_C\\
u_L &= u_1 + u_2\\
u_C &= u_2\\
i_0 &= i_1 + i_L\\
i_1 &= i_2 + i_C\\
i_L &= \mathrm{pre}(i_L) + \bar{h} \cdot di_L\\
u_C &= \mathrm{pre}(u_C) + \bar{h} \cdot du_C
\end{aligned}
$$



Inline Integration V

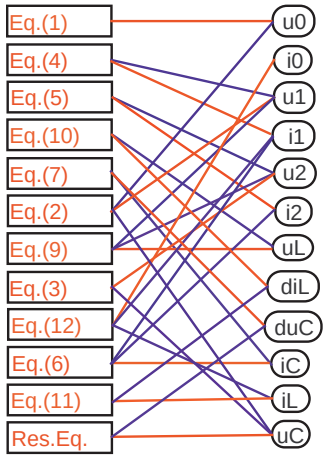
We causalize as much as we can:

$$
\begin{aligned}
u_0 &= f(t)\\
u_1 &= R_1 \cdot i_1\\
u_2 &= R_2 \cdot i_2\\
u_L &= L \cdot di_L\\
i_C &= C \cdot du_C\\
u_0 &= u_1 + u_C\\
u_L &= u_1 + u_2\\
u_C &= u_2\\
i_0 &= i_1 + i_L\\
i_1 &= i_2 + i_C\\
i_L &= \mathrm{pre}(i_L) + \bar{h} \cdot di_L\\
u_C &= \mathrm{pre}(u_C) + \bar{h} \cdot du_C
\end{aligned}
$$



Inline Integration VI

We choose a tearing variable and finalize the causalization:



$$
\begin{aligned}
1:&\quad u_0 = f(t)\\
2:&\quad u_1 = u_0 - u_C\\
3:&\quad u_2 = u_C\\
4:&\quad i_1 = \tfrac{1}{R_1} \cdot u_1\\
5:&\quad i_2 = \tfrac{1}{R_2} \cdot u_2\\
6:&\quad i_C = i_1 - i_2\\
7:&\quad du_C = \tfrac{1}{C} \cdot i_C\\
8:&\quad u_C = \mathrm{pre}(u_C) + h \cdot du_C\\
9:&\quad u_L = u_1 + u_2\\
10:&\quad di_L = \tfrac{1}{L} \cdot u_L\\
11:&\quad i_L = \mathrm{pre}(i_L) + h \cdot di_L\\
12:&\quad i_0 = i_1 + i_L
\end{aligned}
$$

Eq.(2-8) form an algebraic loop in seven equations with u_C serving as tearing variable.

Inline Integration VII

Let us apply variable substitution to find a completely causal set of equations:

$$\begin{aligned} u_C &= \mathrm{pre}(u_C) + h \cdot du_C \\ &= \mathrm{pre}(u_C) + \frac{h}{C} \cdot i_C \\ &= \mathrm{pre}(u_C) + \frac{h}{C} \cdot i_1 - \frac{h}{C} \cdot i_2 \\ &= \mathrm{pre}(u_C) + \frac{h}{R_1 \cdot C} \cdot u_1 - \frac{h}{R_2 \cdot C} \cdot u_2 \\ &= \mathrm{pre}(u_C) + \frac{h}{R_1 \cdot C} \cdot u_0 - \frac{h}{R_1 \cdot C} \cdot u_C - \frac{h}{R_2 \cdot C} \cdot u_C \end{aligned}$$

and therefore:

$$\left[1 + \frac{h}{R_1 \cdot C} + \frac{h}{R_2 \cdot C}\right] \cdot u_C = \mathrm{pre}(u_C) + \frac{h}{R_1 \cdot C} \cdot u_0$$

or:

$$[R_1 \cdot R_2 \cdot C + h \cdot (R_1 + R_2)] \cdot u_C = R_1 \cdot R_2 \cdot C \cdot \mathrm{pre}(u_C) + h \cdot R_2 \cdot u_0$$

Inline Integration VIII

This equation can be solved for u_C:

$$u_C = \frac{R_1 \cdot R_2 \cdot C}{R_1 \cdot R_2 \cdot C + h \cdot (R_1 + R_2)} \cdot \mathrm{pre}(u_C) + \frac{h \cdot R_2}{R_1 \cdot R_2 \cdot C + h \cdot (R_1 + R_2)} \cdot u_0$$

If we let the step size go to zero, we find:

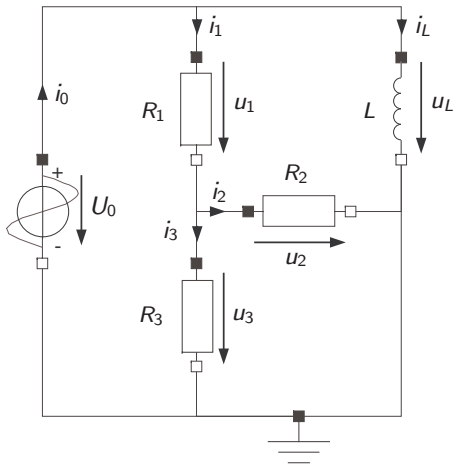
$$\lim_{h \to 0} u_C = \mathrm{pre}(u_C)$$

which is non-singular.

Since the original DAE problem had been of index 0, this is not further surprising.

Inline Integration IX

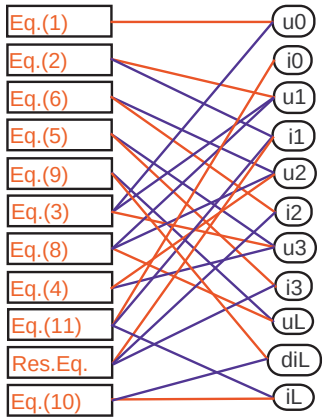
Let us now try an *index-1 problem*:



$$\begin{aligned} u_0 &= f(t) \\ u_1 &= R_1 \cdot i_1 \\ u_2 &= R_2 \cdot i_2 \\ u_3 &= R_3 \cdot i_3 \\ u_L &= L \cdot di_L \\ u_0 &= u_1 + u_3 \\ u_L &= u_1 + u_2 \\ u_3 &= u_2 \\ i_0 &= i_1 + i_L \\ i_1 &= i_2 + i_3 \\ i_L &= \mathrm{pre}(i_L) + h \cdot di_L \end{aligned}$$

Inline Integration X

After complete causalization:



$$\begin{array}{lrcl} 1: & u_0 & = & f(t) \\ 2: & u_1 & = & R_1 \cdot i_1 \\ 3: & u_3 & = & u_0 - u_1 \\ 4: & u_2 & = & u_3 \\ 5: & i_3 & = & \frac{1}{R_3} \cdot u_3 \\ 6: & i_2 & = & \frac{1}{R_2} \cdot u_2 \\ 7: & i_1 & = & i_2 + i_3 \\ 8: & u_L & = & u_1 + u_2 \\ 9: & di_L & = & \frac{1}{L} \cdot u_L \\ 10: & i_L & = & \mathrm{pre}(i_L) + h \cdot di_L \\ 11: & i_0 & = & i_1 + i_L \end{array}$$

Eq.(2-7) form an algebraic loop in six equations with i_1 serving as tearing variable.

Inline Integration XI

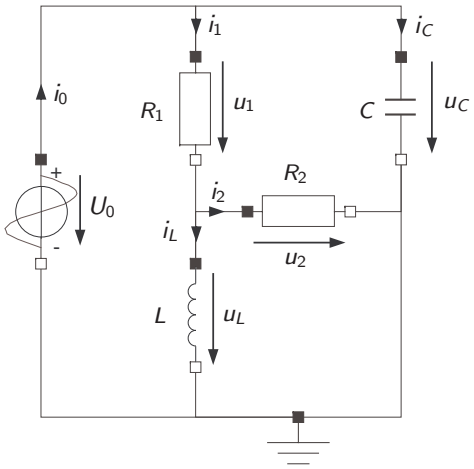
Using the variable substitution technique, we can find a closed-form expression for the tearing variable:

$$i_1 = \frac{R_2 + R_3}{R_1 \cdot R_2 + R_1 \cdot R_3 + R_2 \cdot R_3} \cdot u_0$$

The expression for i_1 is not even a function of the normalized step size $\bar{h}$, i.e., it is non-singular for any value of $\bar{h}$.

Inline Integration XII

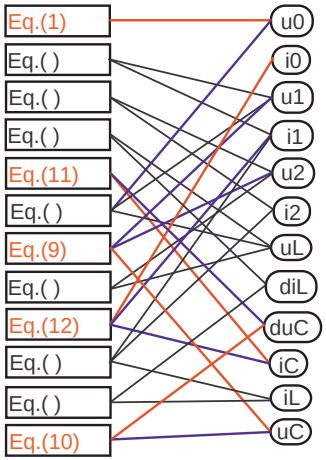
Let us now try an *index-2 problem*:



$$\begin{aligned}
u_0 &= f(t)\\
u_1 &= R_1 \cdot i_1\\
u_2 &= R_2 \cdot i_2\\
u_L &= L \cdot di_L\\
i_C &= C \cdot du_C\\
u_0 &= u_1 + u_L\\
u_C &= u_1 + u_2\\
u_L &= u_2\\
i_0 &= i_1 + i_C\\
i_1 &= i_2 + i_L\\
i_L &= \mathrm{pre}(i_L) + h \cdot di_L\\
u_C &= \mathrm{pre}(u_C) + h \cdot du_C
\end{aligned}$$

Inline Integration XIII

After partial causalization (without tearing the algebraic loop):



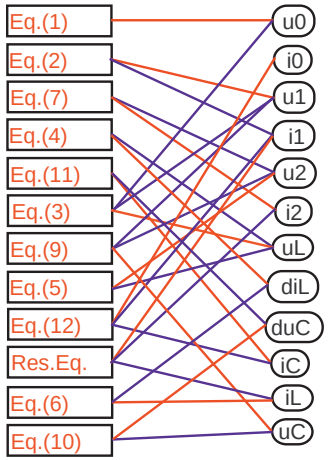
- We found an algebraic loop in seven equations and seven unknowns.
- Unfortunately, we already got ourselves into troubles, as Eq.(10) needs to be solved for du_C:

$$du_C = \frac{u_C - \mathrm{pre}(u_C)}{h}$$

i.e., we ended up with the step size, h, in the denominator, which invariably will cause numerical difficulties, when we try to simulate the system using a small step size. We had no choice in the matter, as the derivative causality on the capacitor was dictated upon us.

Inline Integration XIV

After complete causalization:



$$\begin{array}{lllll}
1: & u_0 & = & f(t)\\
2: & u_1 & = & R_1 \cdot i_1\\
3: & u_L & = & u_0 - u_1\\
4: & di_L & = & \frac{1}{L} \cdot u_L\\
5: & u_2 & = & u_L\\
6: & i_L & = & \mathrm{pre}(i_L) + h \cdot di_L\\
7: & i_2 & = & \frac{1}{R_2} \cdot u_2\\
8: & i_1 & = & i_2 + i_L\\
9: & u_C & = & u_1 + u_2\\
10: & du_C & = & \frac{u_C - \mathrm{pre}(u_C)}{h}\\
11: & i_C & = & C \cdot du_C\\
12: & i_0 & = & i_1 + i_C
\end{array}$$

Inline Integration XV

Using the variable substitution technique, we can find a closed-form expression for the tearing variable:

$$i_1 = \frac{L + h \cdot R_2}{L \cdot (R_1 + R_2) + h \cdot R_2} \cdot u_0 + \frac{R_2 \cdot L}{L \cdot (R_1 + R_2) + h \cdot R_2} \cdot \mathrm{pre}(i_L)$$

If we let the step size go to zero, we find:

$$\lim_{h \to 0} i_1 = \frac{1}{R_1 + R_2} \cdot u_0 + \frac{R_2}{R_1 + R_2} \cdot \mathrm{pre}(i_L)$$

- At least in the given example, inlining was able to solve also the higher-index problem directly.
- Also DASSL is able to sometimes simulate index-2 problems directly.
- Yet, inlining the higher-index problem directly came at a price, as we ended up with the step size, h, in the denominator of one of the model equations.
- Thus, it is usually preferred to first apply the index reduction algorithm by Pantelides.

Conclusions

- In this second presentation on DAE solvers, we looked at *single-step DAE solvers* based on new classes of *implicit Runge-Kutta algorithms* not previously discussed.
- We then discussed **DASSL**, the most successful DAE code currently on the market and shed some light on the limitations of that code.
- A third topic concerned a new way of looking at DAE solution, called *inline integration*, that merges the model equations with the solver equations. We demonstrated some of the features of inline integration by discussing the inlining of BDF algorithms.

References

1. Elmqvist, H., M. Otter, and F.E. Cellier (1995), "Inline Integration: A New Mixed Symbolic/Numeric Approach for Solving Differential-Algebraic Equation Systems," *Proc. ESM'95, SCS European Simulation Multi-Conference*, Prague, Czech Republic, pp.xxiii-xxxiv.

2. Treeaporn, Vicha (2005), *Efficient Simulation of Physical System Models Using Inlined Implicit Runge-Kutta Algorithms*, MS Thesis, Dept. of Electr. & Comp. Engr., University of Arizona, Tucson, AZ.