Numerical Simulation of Dynamic Systems XVIII

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

April 23, 2013

Differential Algebraic Equations

As we have meanwhile understood, a model described by *state equations* is not the normal form, in which a model of a dynamical system presents itself initially.

We almost always obtain a *mixture of differential and algebraic equations*. Furthermore, these equations are formulated implicitly:

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

In the last few presentations, we analyzed how such a DAE model can be converted symbolically to ODE form by the model compiler. In the next few presentations, we shall discuss, how a DAE model can be simulated directly.

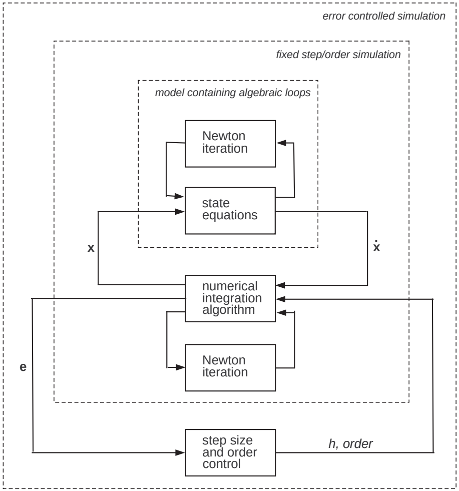
As the derivatives show up in the DAE model implicitly, we shall need to *iterate over the model equations* during each function evaluation.

The Three Iteration Loops

When simulating an implicitly formulated model of a dynamic system using an implicit DAE solver with step size and order control, we encounter *three iteration loops*.

- On the innermost level, we need to iterate over each function evaluation in order to obtain the values of the state derivatives. This iteration is the most expensive one, as it needs to be performed most frequently.
- On the next higher level, we need to iterate over each integration step, as we are using an implicit DAE solver for the simulation.
- On the highest level, it may happen that an entire integration step is rejected and needs to be repeated, because the estimation of the local integration error indicates that we are using either a step size that is too large or an order that is too low.

The Three Iteration Loops II


error controlled simulation
fixed step/order simulation
model containing algebraic loops
Newton iteration
state equations
x
$\dot{x}$
numerical integration algorithm
Newton iteration
e
step size and order control
h, order

The Three Iteration Loops III

Are these three separate iteration loops really necessary?

- Do processes that require iterations, really exist in the physical world? Isn't the physical world *causal*, i.e., isn't it true that each event has one or several causes, and that a strictly sequential ordering is possible between causes and effects? Don't iterations defy the principle of strict causality?
- Mutual causal dependencies do indeed exist in physics and are rather common. The relationship between voltage and current in a resistor is non-causal. It is not true that the potential difference at the two ends of the resistor makes current flow, or that the current flowing through the resistor causes a voltage drop. These are simply two different facets of one and the same physical phenomenon.
- Yet "causal loops" do not truly exist in the physical world. If we place two resistors in series, this will create an algebraic loop in our model. The idea of a loop implies a sequence of execution, i.e., a causes b, which in turn causes c, which is responsible for a. Physics doesn't understand the concept of a "sequence of execution." *Physics is by its very nature completely non-causal.*

All phenomena observed are byproducts of the big balance equations that we call the conservation principles: conservation of energy, conservation of mass, and conservation of momentum.

Simulation of Implicit DAEs Using Implicit DAE Solvers

Let us simulate the model:

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

using the BDF3 algorithm:

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

We can solve the solver equation for the derivative vector:

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

We substitute the solver equation in its derivative form into the model equations:

$$\mathcal{F}(\mathbf{x_{k+1}}) = \mathbf{f}(\mathbf{x_{k+1}}, \frac{1}{h}\left[\frac{11}{6} \cdot \mathbf{x_{k+1}} - 3\mathbf{x_k} + \frac{3}{2}\mathbf{x_{k-1}} - \frac{1}{3}\mathbf{x_{k-2}}\right], \mathbf{u_{k+1}}, t_{k+1}) = 0.0$$

Newton iteration can evidently be applied directly, fusing the two innermost iteration loops.

Simulation of Implicit DAEs Using Implicit DAE Solvers II

Where did the concept of a state-space description of physical systems come from?

- It originated with the desire to separate the process of *modeling* (in a simple-minded way of looking at things, the process of generating a state-space model out of physical observations) from that of *simulation* (the process of translating the state-space model into trajectory behavior).
- In the context of explicit ODE solvers, this separation comes quite naturally. The state-space model computes $\dot{\mathbf{x}}(t_k)$ out of $\mathbf{x}(t_k)$, and the integration algorithm in turn computes $\mathbf{x}(t_{k+1})$ out of $\mathbf{x}(t_k)$ and $\dot{\mathbf{x}}(t_k)$ – a meaningful and clean separation of duties.
- By the time implicit integration algorithms were introduced, this separation was no longer as clean and crisp and beautiful. We now had to deal with causal loops anyway, since the state-space model and the ODE solver now operated on the same time instant, i.e., they had to co-operate to find simultaneously $\mathbf{x}(t_{k+1})$ and $\dot{\mathbf{x}}(t_{k+1})$.
- However, tradition had imprinted this separation so deeply into the brains of the simulation practitioners of that epoch that no one bothered to raise the question whether this separation was still useful, or whether it might not even be detrimental to our task.

Simulation of Implicit DAEs Using Implicit DAE Solvers III

- By combining the *solver equations* and the *model equations*, i.e., by substituting the solver equations into the model equations, we were able to amalgamate the two innermost iteration loops into a single loop. This often pays off in terms of computational efficiency.

- As a by-product, we were able to throw out n equations and unknowns from our model, where n denotes the order of our model, i.e., the number of individual integrators, as we eliminated all of the derivatives from the model. They are no longer computed.

- We shall explore this fruitful idea some more in the context of *inline integration*.

BDF Algorithms

A numerical integration algorithm that is applied directly to an implicit model description is called *numerical differential algebraic equation solver* or shorter *numerical DAE solver*.

We need to analyze the *numerical stability properties* of numerical DAE solvers.

To this end, we start with the linear implicit DAE model:

$$\mathbf{A} \cdot \mathbf{x} + \mathbf{B} \cdot \dot{\mathbf{x}} = 0$$

Let us insert the *BDF3 algorithm* in its differential form:

$$\mathbf{A} \cdot \mathbf{x_{k+1}} + \frac{11\mathbf{B}}{6h} \cdot \left(\mathbf{x_{k+1}} - \frac{18}{11}\mathbf{x_k} + \frac{9}{11}\mathbf{x_{k-1}} - \frac{2}{11}\mathbf{x_{k-2}} \right) = 0.0$$

Thus:

$$\mathbf{x_{k+1}} = \left(-\mathbf{B} - \frac{6\mathbf{A}h}{11} \right)^{-1} \cdot \left(-\frac{18\mathbf{B}}{11}\mathbf{x_k} + \frac{9\mathbf{B}}{11}\mathbf{x_{k-1}} - \frac{2\mathbf{B}}{11}\mathbf{x_{k-2}} \right)$$

BDF Algorithms II

Let us first discuss the simplest case:

$$\mathbf{B} = -\mathbf{I}^{(n)}$$

In this case, the linear implicit DAE model degenerates to the explicit linear ODE model:

$$\dot{\mathbf{x}} = \mathbf{A} \cdot \mathbf{x}$$

and the equation used to compute $\mathbf{x}_{k+1}$ degenerates to:

$$\mathbf{x}_{k+1} = \left(\mathbf{I}^{(n)} - \frac{6\mathbf{A}h}{11} \right)^{-1} \cdot \left(\frac{18}{11}\mathbf{x}_k - \frac{9}{11}\mathbf{x}_{k-1} + \frac{2}{11}\mathbf{x}_{k-2} \right)$$

which is identical to the equation that had been used earlier to determine the stability domain of the BDF3 numerical ODE solver.

- At least in this most simple situation, the stability domain is not at all affected by the DAE formulation.

BDF Algorithms III

Let us assume next that $\mathbf{B}$ is a non-singular matrix. In this case, the implicit DAE model can be made explicit:

$$\dot{\mathbf{x}} = -\mathbf{B}^{-1} \cdot \mathbf{A} \cdot \mathbf{x}$$

Does the inversion of B have an effect on the stability domain?

We can determine the stability domain of the method in the following way. We choose the eigenvalues of $-\mathbf{B}^{-1} \cdot \mathbf{A}$ along the unit circle of the complex plane, then apply the so found $\mathbf{A}$- and $\mathbf{B}$-matrices to the $\mathbf{F}$-matrix:

$$\mathbf{F} = \begin{pmatrix} \mathbf{O}^{(\mathbf{n})} & \mathbf{I}^{(\mathbf{n})} & \mathbf{O}^{(\mathbf{n})} \\ \mathbf{O}^{(\mathbf{n})} & \mathbf{O}^{(\mathbf{n})} & \mathbf{I}^{(\mathbf{n})} \\ \frac{2}{11}\left(\mathbf{B} + \frac{6}{11}\mathbf{A}h\right)^{-1}\mathbf{B} & -\frac{9}{11}\left(\mathbf{B} + \frac{6}{11}\mathbf{A}h\right)^{-1}\mathbf{B} & \frac{18}{11}\left(\mathbf{B} + \frac{6}{11}\mathbf{A}h\right)^{-1}\mathbf{B} \end{pmatrix}$$

and determine h such that the dominant eigenvalues of $\mathbf{F}$ are on the unit circle.

BDF Algorithms IV

- We arbitrarily chose several different non-singular **B**-matrices of dimensions 2×2, and computed the corresponding **A**-matrices using:

$$\mathbf{A} = -\mathbf{B} \cdot \begin{pmatrix} 0 & 1 \\ -1 & 2\cos(\alpha) \end{pmatrix}$$

- We then plugged these **A**-matrices into the **F**-matrix of the numerical DAE solver, and computed the stability domains.

- In every single case, the stability domain was exactly the same as that of the corresponding numerical ODE solver.

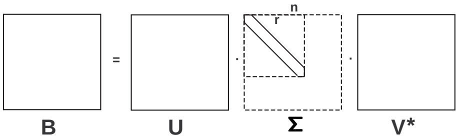
Non-singular B-matrices do not influence the numerical stability properties of the method in any way.

Higher Index Models

With a *singular* **B** *matrix*, the DAE model cannot be reformulated in an explicit form that easily.

We begin by a *singular value decomposition of the* **B** *matrix*:

$$\mathbf{B} = \mathbf{U} \cdot \mathbf{\Sigma} \cdot \mathbf{V}^*$$



with:

$$\mathrm{rank}(\mathbf{U}) = \mathrm{rank}(\mathbf{V}) = n$$
$$\mathrm{rank}(\mathbf{\Sigma}) = \mathrm{rank}(\mathbf{B}) = r < n$$

U and **V** are *unitary matrices*, whereas **Σ** is a *diagonal matrix*.

$\mathbf{V}^*$ is the *Hermitian transpose* of **V**.

Higher Index Models II

Therefore:

$$
\begin{aligned}
& \mathbf{A} \cdot \mathbf{x} + \mathbf{U} \cdot \mathbf{\Sigma} \cdot \mathbf{V}^* \cdot \dot{\mathbf{x}} = 0.0 \\
\Rightarrow \quad & \mathbf{U}^* \cdot \mathbf{A} \cdot \mathbf{x} + \mathbf{\Sigma} \cdot \mathbf{V}^* \cdot \dot{\mathbf{x}} = 0.0
\end{aligned}
$$

and with:

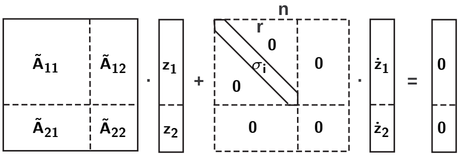
$$\mathbf{z} = \mathbf{V}^* \cdot \mathbf{x}$$

we obtain:

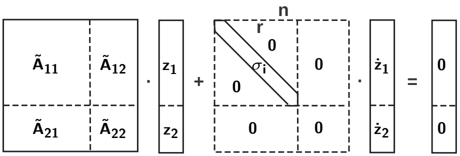
$$\mathbf{U}^* \cdot \mathbf{A} \cdot \mathbf{V} \cdot \mathbf{z} + \mathbf{\Sigma} \cdot \dot{\mathbf{z}} = 0.0$$

or:

$$\tilde{\mathbf{A}} \cdot \mathbf{z} + \mathbf{\Sigma} \cdot \dot{\mathbf{z}} = 0.0$$


$\tilde{\mathbf{A}}_{11}$
$\tilde{\mathbf{A}}_{12}$
$\tilde{\mathbf{A}}_{21}$
$\tilde{\mathbf{A}}_{22}$
·
$\mathbf{z}_1$
$\mathbf{z}_2$
+
n
r
0
σ_i
0
0
0
0
·
$\dot{\mathbf{z}}_1$
$\dot{\mathbf{z}}_2$
=
0
0

Higher Index Models III



$$\tilde{\mathbf{A}}_{11} \cdot \mathbf{z}_1 + \tilde{\mathbf{A}}_{12} \cdot \mathbf{z}_2 + \mathbf{\Sigma}_{11} \cdot \dot{\mathbf{z}}_1 = 0.0$$
$$\tilde{\mathbf{A}}_{21} \cdot \mathbf{z}_1 + \tilde{\mathbf{A}}_{22} \cdot \mathbf{z}_2 = 0.0$$

If the matrix $\tilde{\mathbf{A}}_{22}$ is *singular*, the *model is poorly defined*.

On the other hand, if the matrix $\tilde{\mathbf{A}}_{22}$ is *non-singular*, we can write:

$$\mathbf{z}_2 = -\tilde{\mathbf{A}}_{22}^{-1} \cdot \tilde{\mathbf{A}}_{21} \cdot \mathbf{z}_1$$

and consequently:

$$\dot{\mathbf{z}}_1 = \mathbf{\Sigma}_{11}{}^{-1} \cdot \left(\tilde{\mathbf{A}}_{12} \cdot \tilde{\mathbf{A}}_{22}^{-1} \cdot \tilde{\mathbf{A}}_{21} - \tilde{\mathbf{A}}_{11} \right) \cdot \mathbf{z}_1$$

Higher Index Models IV

- Although the system seemed to be of order n, in reality, it is only of order $r < n$.
- Such a model is called a *higher index model*.
- We have already learnt that there exist *symbolic algorithms* for *automatic index reduction*. One of these algorithms is the *Pantelides algorithm* that was discussed extensively in the previous two presentations.
- The Pantelides algorithm is an algorithm of *linear computational complexity*. Thus, it is very efficient.
- Dealing with higher index models directly, i.e., numerically, without prior index reduction is never worth it. It is always better to first reduce the perturbation index symbolically down to at least index-1 as part of the model compilation.
- **Consequently, the case of singular B matrices is irrelevant**.

AM Algorithms

Let us analyze now what happens when we implement a *DAE algorithm of the AM class*.

Let us start with the AM3 algorithm:

$$\mathbf{x}(t_{k+1}) = \mathbf{x}(t_k) + \frac{h}{12}\left(5\mathbf{f}_{k+1} + 8\mathbf{f_k} - \mathbf{f_{k-1}}\right)$$

We can solve the solver equation for the derivative vector:

$$\dot{\mathbf{x}}(t_{k+1}) = \mathbf{f_{k+1}} = \frac{12}{5h}\left(\mathbf{x}(t_{k+1}) - \mathbf{x}(t_k)\right) - \frac{8}{5}\mathbf{f_k} + \frac{1}{5}\mathbf{f_{k-1}}$$

We can substitute that equation into the DAE model:

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

thereby eliminating the derivative $\dot{\mathbf{x}}_{\mathbf{k+1}}$.

AM Algorithms II

Unfortunately, the "known" quantities $\mathbf{f_k}$ and $\mathbf{f_{k-1}}$ are left in the equation. Yet, we don't have these values available as we just *eliminated the computation of the derivative vector from the model*.

This didn't happen in the case of the BDF algorithm, because those solvers don't make use of the derivatives of the past in their computation.

- Numerical linear multi-step (ODE or DAE) solvers that make use of only a single derivative in their formulae are called *one-legged algorithms*.
- The BDF algorithms are thus one-legged algorithms. In contrast, the AM algorithms are not.

AM Algorithms III

One way to resolve the problem is to *introduce a second Newton iteration*:

$$\mathcal{F}_1\left(\mathbf{x}(t_{k+1})\right) = \mathbf{f}\left(\mathbf{x}(t_{k+1}), \frac{12}{5h}\mathbf{x}(t_{k+1}) - \frac{12}{5h}\mathbf{x}(t_k) - \frac{8}{5}\mathbf{w}(t_k) + \frac{1}{5}\mathbf{w}(t_{k-1}), \mathbf{u}(t_{k+1}), t_{k+1}\right) = 0.0$$

$$\mathcal{F}_2\left(\mathbf{w}(t_{k+1})\right) = \mathbf{f}\left(\mathbf{x}(t_{k+1}), \mathbf{w}(t_{k+1}), \mathbf{u}(t_{k+1}), t_{k+1}\right) = 0.0$$

- The upper equation determines $\mathbf{x}(t_{k+1})$. In this iteration, $\mathbf{u}(t_{k+1})$, $\mathbf{x}(t_k)$, $\mathbf{w}(t_k)$, and $\mathbf{w}(t_{k-1})$ are assumed known.
- The lower equation then evaluates $\mathbf{w}(t_{k+1})$. During that iteration, $\mathbf{x}(t_{k+1})$ can be assumed known as well. Clearly, $\mathbf{w}$ is just another name for $\dot{\mathbf{x}}$.

AM Algorithms IV

Let us now find the *numerical stability domain fo the AM3 algorithm* when used in its DAE form.

Substituting the AM3 formula in its differentiated form into the linear DAE model, we obtain:

$$\begin{aligned}\mathbf{x}(t_{k+1}) &= \left(\mathbf{B} + \frac{5}{12}\mathbf{A}h\right)^{-1} \cdot \left(\mathbf{B}\mathbf{x}(t_k) + \frac{2}{3}\mathbf{B}h\mathbf{w}(t_k) - \frac{1}{12}\mathbf{B}h\mathbf{w}(t_{k-1})\right)\\ \mathbf{w}(t_{k+1}) &= -\mathbf{B}^{-1}\mathbf{A}\mathbf{x}(t_{k+1})\end{aligned}$$

With the state vector:

$$\mathbf{z_k} = \begin{pmatrix}\mathbf{x_k}\\ \mathbf{w_{k-1}}\\ \mathbf{w_k}\end{pmatrix}$$

we find the **F**-matrix:

$$\mathbf{F} = \begin{pmatrix} \left(\mathbf{B} + \frac{5}{12}\mathbf{A}h\right)^{-1}\mathbf{B} & -\frac{1}{12}\left(\mathbf{B} + \frac{5}{12}\mathbf{A}h\right)^{-1}\mathbf{B}h & \frac{2}{3}\left(\mathbf{B} + \frac{5}{12}\mathbf{A}h\right)^{-1}\mathbf{B}h\\ \mathbf{O}^{(\mathbf{n})} & \mathbf{O}^{(\mathbf{n})} & \mathbf{I}^{(\mathbf{n})}\\ -\mathbf{B}^{-1}\mathbf{A}\left(\mathbf{B} + \frac{5}{12}\mathbf{A}h\right)^{-1}\mathbf{B} & \frac{1}{12}\mathbf{B}^{-1}\mathbf{A}\left(\mathbf{B} + \frac{5}{12}\mathbf{A}h\right)^{-1}\mathbf{B}h & -\frac{2}{3}\mathbf{B}^{-1}\mathbf{A}\left(\mathbf{B} + \frac{5}{12}\mathbf{A}h\right)^{-1}\mathbf{B}h \end{pmatrix}$$

AM Algorithms V

- Choosing again an arbitrary non-singular **B**-matrix, we obtain the same numerical stability domain every time as when using the AM3 algorithm in its ODE form.
- This result looks a bit strange, because the size of the **F**-matrix is larger in the DAE formulation than in the ODE formulation. Consequently, the **F**-matrix has more eigenvalues.
- However, $\mathbf{w}(t_{k+1})$ is linear in $\mathbf{x}(t_{k+1})$, and consequently, the **F**-matrix is singular. We simply added a few eigenvalues at the origin. These eigenvalues do not influence the numerical stability domain.

It has become evident that the numerical stability domain of an implicit (index-1) DAE solver is exactly the same as that of the corresponding ODE solver.

AB Algorithms

Let us discuss next what happens if we try to convert an *explicit solver* to DAE form.

We start with the AB3 algorithm:

$$\mathbf{x}(t_{k+1}) = \mathbf{x}(t_k) + \frac{h}{12}\left(23\mathbf{f}_k - 16\mathbf{f}_{k-1} + 5\mathbf{f}_{k-2}\right)$$

We solve that equation for the derivative vector:

$$\dot{\mathbf{x}}(t_{k+1}) = \frac{12}{23h}\left(\mathbf{x}(t_{k+2}) - \mathbf{x}(t_{k+1})\right) + \frac{16}{23}\mathbf{f}_k - \frac{5}{23}\mathbf{f}_{k-1}$$

Unfortunately, *explicit integration formulae* are converted to *over-implicit differentiation formulae*.

These formulae are useless, as they result in a gigantic iteration over the entire simulation rather than being limited to a single integration step.

Over-implicit Numerical Integration Algorithms

This brings us to another idea. How if we were to convert *over-implicit integration algorithms* that, until now, had been totally useless to DAE form?

The *over-implicit 3rd-order Adams formula*:

$$\mathbf{x}(t_{k+1}) = \mathbf{x}(t_k) + \frac{h}{12}\left(-\mathbf{f}(t_{k+2}) + 8\mathbf{f}(t_{k+1}) + 5\mathbf{f}(t_k)\right)$$

can be converted to the explicit derivative form:

$$\dot{\mathbf{x}}(t_{k+1}) = \frac{12}{h}\left(\mathbf{x}(t_{k-1}) - \mathbf{x}(t_k)\right) + 8\mathbf{f}(t_k) + 5\mathbf{f}(t_{k-1})$$

- **Over-implicit numerical integration formulae are converted to explicit numerical differentiation formulae.**
- Unfortunately, the over-implicit integration algorithm is unstable. Consequently, also the explicit differentiation formula is unstable.

Over-implicit Numerical Integration Algorithms II

Let us now consider over-implicit algorithms of the BDF class.

The *over-implicit 3rd-order BDF algorithm*:

$$\mathbf{x}(t_{k+1}) = \frac{57}{26}\mathbf{x}(t_k) - \frac{21}{13}\mathbf{x}(t_{k-1}) + \frac{11}{26}\mathbf{x}(t_{k-2}) + \frac{6h}{26}\mathbf{f}(t_{k+2})$$

can be converted to the explicit derivative form:

$$\dot{\mathbf{x}}(t_{k+1}) = \frac{1}{6h}\left(26\mathbf{x}(t_k) - 57\mathbf{x}(t_{k-1}) + 42\mathbf{x}(t_{k-2}) - 11\mathbf{x}(t_{k-3})\right)$$

- Unfortunately, also this over-implicit integration algorithm is unstable. Consequently, also the explicit differentiation formula is unstable.

Over-implicit Numerical Integration Algorithms III

- In general, explicit differentiation formulae should not be used because of their numerical instability.
- In contrast, implicit differentiation formulae are numerically stable if the corresponding implicit integration formulae are stable as well.
- The numerical stability properties of a linear multi-step algorithm do not change at all when converted from integral to differential form or vice-versa.

Accuracy Properties

We should also investigate the *accuracy properties* of the *DAE algorithms*.

- We know from our earlier discussions that BDFi formulae are inefficient for use in non-stiff ODEs due to their poor accuracy properties. This was documented when we compared the efficiency of different ODE solvers when simulating the wave equation.
- The problem evidently hasn't vanished by reformulating the model in a DAE format. Thus, we may suspect that the AMi formulae will still work better than the BDFi formulae also in non-stiff DAE simulation. However, whether this is true or not will depend on the relative cost to be paid for the second Newton iteration.

Accuracy Properties II

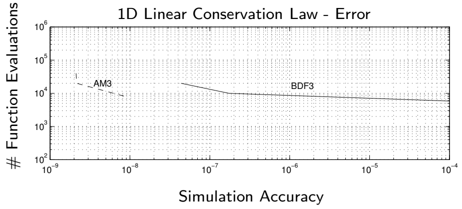
I simulated once more the wave equation using BDF3 and AM3, this time using the DAE formulation of these algorithms:

h	BDF3	AM3
0.1	garbage	unstable
0.05	garbage	unstable
0.02	garbage	unstable
0.01	garbage	unstable
0.005	0.9469e-2	0.8783e-8
0.002	0.1742e-6	0.2149e-8
0.001	0.4363e-7	0.2120e-8

- The entries in the above table, displaying the *step sizes* used, look exactly the same as the corresponding entries in the table shown earlier.
- **This is not surprising, as the DAE formulation doesn't change the numerical properties of the methods**.

Accuracy Properties III

- Yet the cost of the AM3 algorithm in terms of the *number of function evaluations* may be higher in the DAE formulation than in the ODE formulation due to the need for a second Newton iteration.
- On the other hand, the cost of the BDF3 algorithm in terms of the number of function evaluations may be lower in the DAE formulation than in the ODE formulation if the model to be simulated is of index 1, i.e., contains algebraic loops (which is not the case for the wave equation).



- The DAE formulation of the AM3 algorithm turned out to be indeed a bit more expensive than the ODE formulation.

Conclusions

In this presentation, we looked at the DAE formulation of linear multi-step integration algorithms, and have reached a number of interesting conclusions:

- The numerical properties of these algorithms, i.e., their numerical stability and accuracy properties, at least for index-0 and index-1 problems, do not change when switching from an ODE to a DAE formulation.
- It never pays off to simulate a higher-index problem directly. A problem with a perturbation index higher than 1 should always undergo symbolic index reduction first, e.g. using the Pantelides algorithm.
- Whether it pays off to transform an index-1 DAE to explicit ODE form, as we proposed in the previous few presentations, is not evident. It may, in some cases, be more efficient to simulate the index-1 DAE problem directly. We shall talk more about this issue later.

Conclusions II

- We have learnt earlier that linear multi-step algorithms are mostly used for the simulation of stiff ODE problems, as indeed, implementations of BDF algorithms are to this day among the most widely used stiff system solvers on the market.
- It makes little difference whether we use a BDF algorithm in its ODE or in its DAE formulation, and indeed, the most widely used stiff DAE solver today, **DASSL**, is an implementation of a variable-step variable-order BDF algorithm.
- **Dymola** makes use of DASSL, albeit with an ODE interface, as its default simulation engine, i.e., Dymola converts all DAE models symbolically to explicit ODE form as part of its model compilation.
- We shall introduce DASSL in the next presentation.

Conclusions III

- We have seen earlier that linear multi-step algorithms are hardly ever efficient for the simulation of non-stiff ODE problems.
- An exception to the rule may be the class of linear time-invariant systems, where the AB algorithms are indeed cost-effective.
- **Matlab** makes use of AB3 for the simulation of linear time-invariant systems, which is reasonable.
- A DAE formulation of explicit linear multi-step methods makes no sense whatsoever.

Conclusions IV

- In the past, we only looked at the simulation of index-0 problems, i.e., explicitly formulated ODE systems without algebraic loops. For those systems, the AM algorithms were not competitive ever.
- The situation changes, when we look at index-1 problems. Here, the algebraic loops force us to employ a Newton iteration anyway, and if we do, we might just as well use a DAE formulation to start with.
- Thus, whereas the AM algorithms are definitely not competitive for the simulation of non-stiff index-0 problems, they may become competitive for the simulation of non-stiff index-1 problems, especially in their DAE formulation.

References

1. Hu, Luoan (1991), *DBDF: An Implicit Numerical Differentiation Algorithm for Integrated Circuit Simulation*, MS Thesis, Dept. of Electrical & Computer Engineering, University of Arizona, Tucson, AZ.