Numerical Simulation of Dynamic Systems XX

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

April 30, 2013

Inline Integration of IRK Algorithms

Let us now try to apply the same idea to the *Radau IIA(3) algorithm*:

$$\mathbf{x}_{\mathbf{k}+\frac{1}{3}} = \mathbf{x}_{\mathbf{k}} + \frac{5h}{12} \cdot \dot{\mathbf{x}}_{\mathbf{k}+\frac{1}{3}} - \frac{h}{12} \cdot \dot{\mathbf{x}}_{\mathbf{k}+1}$$

$$\mathbf{x}_{\mathbf{k}+1} = \mathbf{x}_{\mathbf{k}} + \frac{3h}{4} \cdot \dot{\mathbf{x}}_{\mathbf{k}+\frac{1}{3}} + \frac{h}{4} \cdot \dot{\mathbf{x}}_{\mathbf{k}+1}$$

Inline Integration of IRK Algorithms

Let us now try to apply the same idea to the *Radau IIA(3) algorithm*:

$$\mathbf{x}_{\mathbf{k}+\frac{1}{3}} = \mathbf{x}_{\mathbf{k}} + \frac{5h}{12} \cdot \dot{\mathbf{x}}_{\mathbf{k}+\frac{1}{3}} - \frac{h}{12} \cdot \dot{\mathbf{x}}_{\mathbf{k}+1}$$
$$\mathbf{x}_{\mathbf{k}+1} = \mathbf{x}_{\mathbf{k}} + \frac{3h}{4} \cdot \dot{\mathbf{x}}_{\mathbf{k}+\frac{1}{3}} + \frac{h}{4} \cdot \dot{\mathbf{x}}_{\mathbf{k}+1}$$

This is an algorithm in two stages. The first stage is evaluated at time $t_{k+\frac{1}{3}}$, whereas the second stage is evaluated at time t_{k+1}.

Inline Integration of IRK Algorithms

Let us now try to apply the same idea to the *Radau IIA(3) algorithm*:

$$\mathbf{x}_{\mathbf{k}+\frac{1}{3}} = \mathbf{x}_{\mathbf{k}} + \frac{5h}{12} \cdot \dot{\mathbf{x}}_{\mathbf{k}+\frac{1}{3}} - \frac{h}{12} \cdot \dot{\mathbf{x}}_{\mathbf{k}+1}$$
$$\mathbf{x}_{\mathbf{k}+1} = \mathbf{x}_{\mathbf{k}} + \frac{3h}{4} \cdot \dot{\mathbf{x}}_{\mathbf{k}+\frac{1}{3}} + \frac{h}{4} \cdot \dot{\mathbf{x}}_{\mathbf{k}+1}$$

This is an algorithm in two stages. The first stage is evaluated at time $t_{k+\frac{1}{3}}$, whereas the second stage is evaluated at time t_{k+1}.

However, we are not dealing here with a *prediction* followed by a *correction*. There is *a single Newton iteration* that extends over all equations of both stages.

Inline Integration of IRK Algorithms

Let us now try to apply the same idea to the *Radau IIA(3) algorithm*:

$$\mathbf{x}_{\mathbf{k}+\frac{1}{3}} = \mathbf{x}_{\mathbf{k}} + \frac{5h}{12} \cdot \dot{\mathbf{x}}_{\mathbf{k}+\frac{1}{3}} - \frac{h}{12} \cdot \dot{\mathbf{x}}_{\mathbf{k}+1}$$
$$\mathbf{x}_{\mathbf{k}+1} = \mathbf{x}_{\mathbf{k}} + \frac{3h}{4} \cdot \dot{\mathbf{x}}_{\mathbf{k}+\frac{1}{3}} + \frac{h}{4} \cdot \dot{\mathbf{x}}_{\mathbf{k}+1}$$

This is an algorithm in two stages. The first stage is evaluated at time $t_{k+\frac{1}{3}}$, whereas the second stage is evaluated at time t_{k+1}.

However, we are not dealing here with a *prediction* followed by a *correction*. There is *a single Newton iteration* that extends over all equations of both stages.

All equations have to be evaluated simultaneously, in spite of the fact that the variables involved in the iteration represent two different time instances.

Inline Integration of IRK Algorithms

Let us now try to apply the same idea to the *Radau IIA(3) algorithm*:

$$\mathbf{x}_{\mathbf{k}+\frac{1}{3}} = \mathbf{x}_{\mathbf{k}} + \frac{5h}{12} \cdot \dot{\mathbf{x}}_{\mathbf{k}+\frac{1}{3}} - \frac{h}{12} \cdot \dot{\mathbf{x}}_{\mathbf{k}+1}$$
$$\mathbf{x}_{\mathbf{k}+1} = \mathbf{x}_{\mathbf{k}} + \frac{3h}{4} \cdot \dot{\mathbf{x}}_{\mathbf{k}+\frac{1}{3}} + \frac{h}{4} \cdot \dot{\mathbf{x}}_{\mathbf{k}+1}$$

This is an algorithm in two stages. The first stage is evaluated at time $t_{k+\frac{1}{3}}$, whereas the second stage is evaluated at time t_{k+1}.

However, we are not dealing here with a *prediction* followed by a *correction*. There is *a single Newton iteration* that extends over all equations of both stages.

All equations have to be evaluated simultaneously, in spite of the fact that the variables involved in the iteration represent two different time instances.

In order to inline this algorithm, we need to *duplicate the equations*.

Inline Integration of IRK Algorithms II

Using the same index-0 example as in the previous presentation, we find:

$$
\begin{aligned}
v_0 &= f(t + \frac{h}{3}) \\
v_1 &= R_1 \cdot j_1 \\
v_2 &= R_2 \cdot j_2 \\
v_L &= L \cdot dj_L \\
j_C &= C \cdot dv_C \\
v_0 &= v_1 + v_C \\
v_L &= v_1 + v_2 \\
v_C &= v_2 \\
j_0 &= j_1 + j_L \\
j_1 &= j_2 + j_C \\
j_L &= \mathrm{pre}(i_L) + \frac{5h}{12} \cdot dj_L - \frac{h}{12} \cdot di_L \\
v_C &= \mathrm{pre}(u_C) + \frac{5h}{12} \cdot dv_C - \frac{h}{12} \cdot du_C
\end{aligned}
$$

$$
\begin{aligned}
u_0 &= f(t + h) \\
u_1 &= R_1 \cdot i_1 \\
u_2 &= R_2 \cdot i_2 \\
u_L &= L \cdot di_L \\
i_C &= C \cdot du_C \\
u_0 &= u_1 + u_C \\
u_L &= u_1 + u_2 \\
u_C &= u_2 \\
i_0 &= i_1 + i_L \\
i_1 &= i_2 + i_C \\
i_L &= \mathrm{pre}(i_L) + \frac{3h}{4} \cdot dj_L + \frac{h}{4} \cdot di_L \\
u_C &= \mathrm{pre}(u_C) + \frac{3h}{4} \cdot dv_C + \frac{h}{4} \cdot du_C
\end{aligned}
$$

Inline Integration of IRK Algorithms III

- Instead of dealing with 12 equations in 12 variables, as in the case of inlining a BDF method, we now are confronted with 24 equations in 24 variables.

Inline Integration of IRK Algorithms III

- Instead of dealing with 12 equations in 12 variables, as in the case of inlining a BDF method, we now are confronted with 24 equations in 24 variables.
- In the duplication of the equations, we simply used variable names starting with v and j to denote voltages and currents at time $t_{k+\frac{1}{3}}$, and we used variable names starting with u and i to refer to voltages and currents at time t_{k+1}.

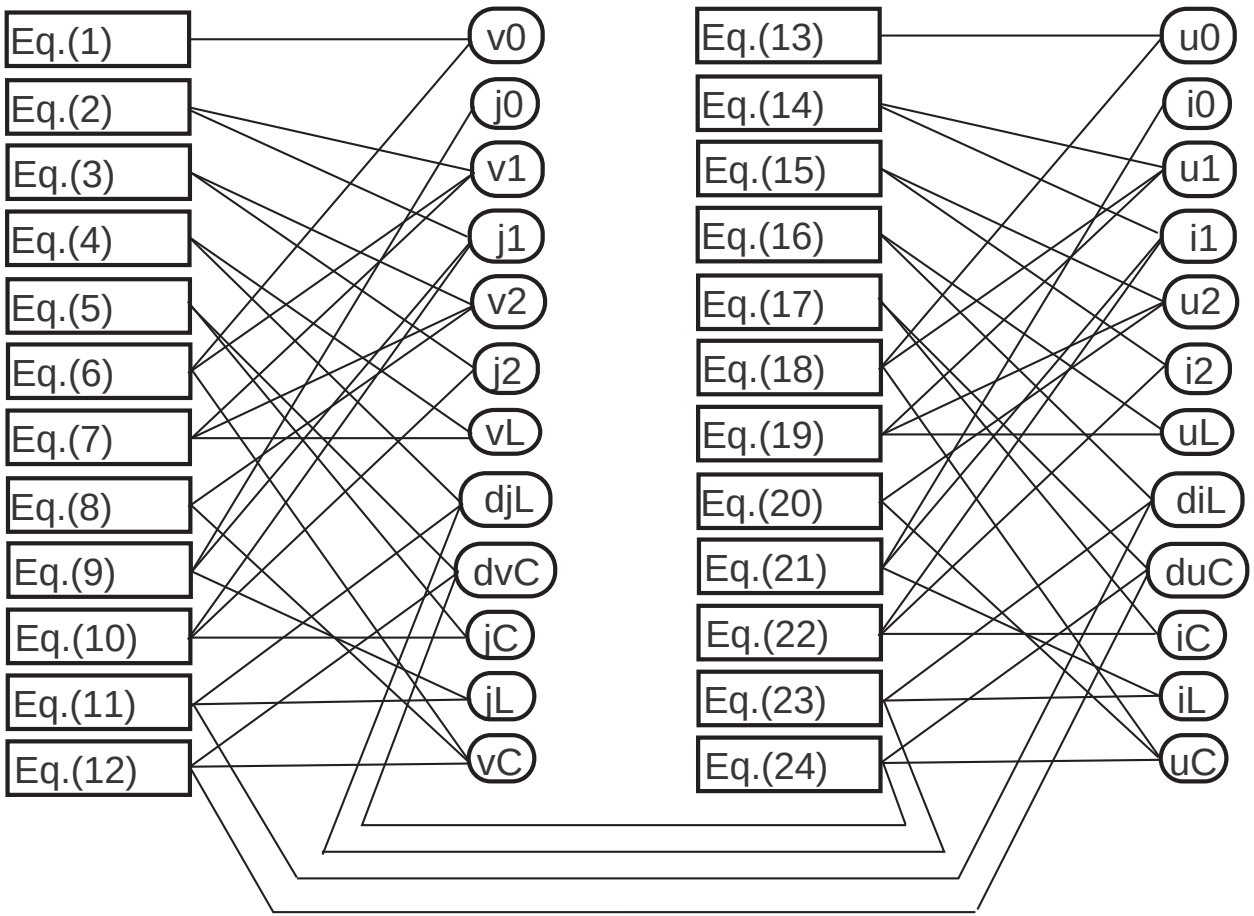
Inline Integration of IRK Algorithms III

- Instead of dealing with 12 equations in 12 variables, as in the case of inlining a BDF method, we now are confronted with 24 equations in 24 variables.
- In the duplication of the equations, we simply used variable names starting with v and j to denote voltages and currents at time $t_{k+\frac{1}{3}}$, and we used variable names starting with u and i to refer to voltages and currents at time t_{k+1}.
- Although the IRK technique will allow us to use much larger step sizes in comparison with the BDF approach, every function evaluation is now more expensive.

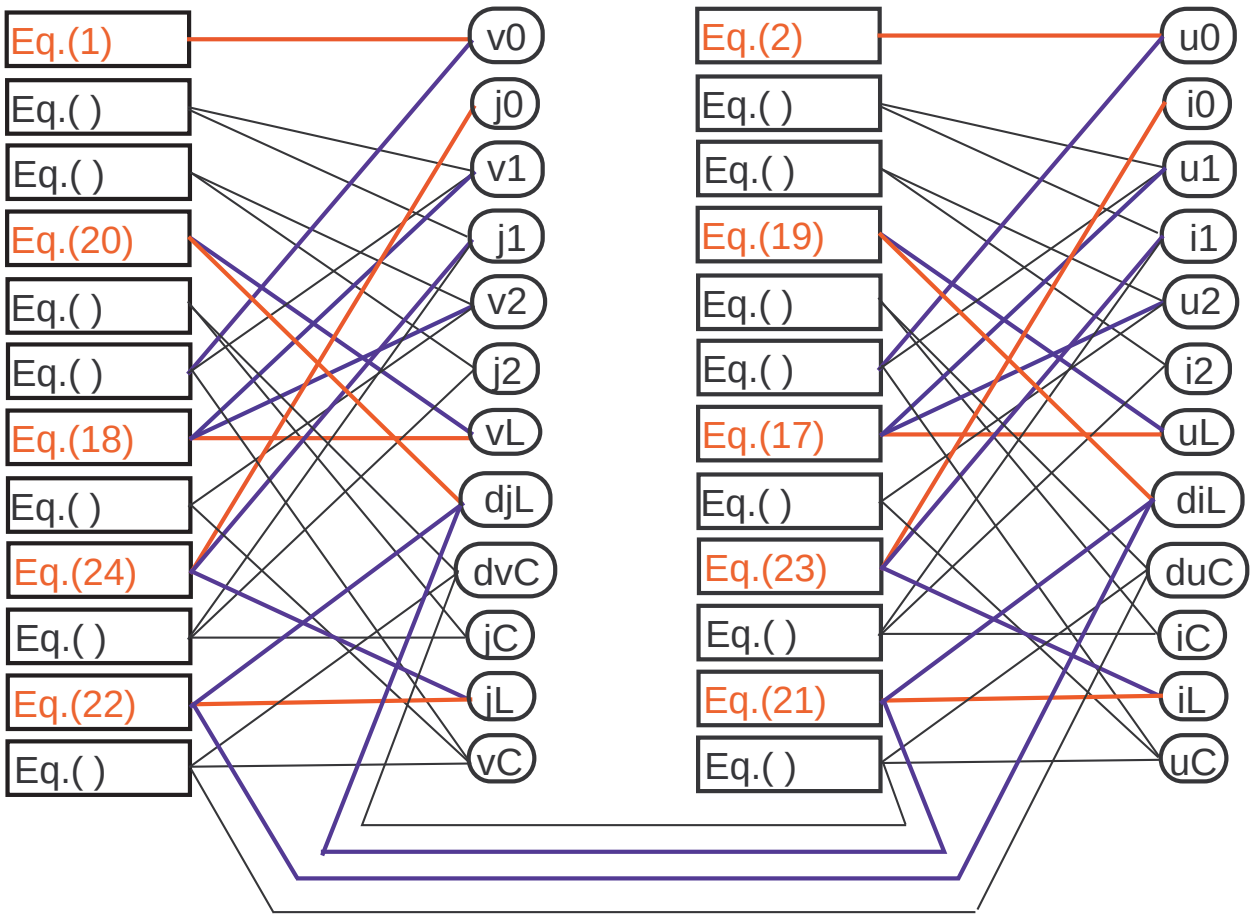
Inline Integration of IRK Algorithms III

- Instead of dealing with 12 equations in 12 variables, as in the case of inlining a BDF method, we now are confronted with 24 equations in 24 variables.
- In the duplication of the equations, we simply used variable names starting with v and j to denote voltages and currents at time $t_{k+\frac{1}{3}}$, and we used variable names starting with u and i to refer to voltages and currents at time t_{k+1}.
- Although the IRK technique will allow us to use much larger step sizes in comparison with the BDF approach, every function evaluation is now more expensive.
- We need to analyze the *number of tearing variables* that result from inlining the IRK method. This will determine the *economy of the algorithm*.

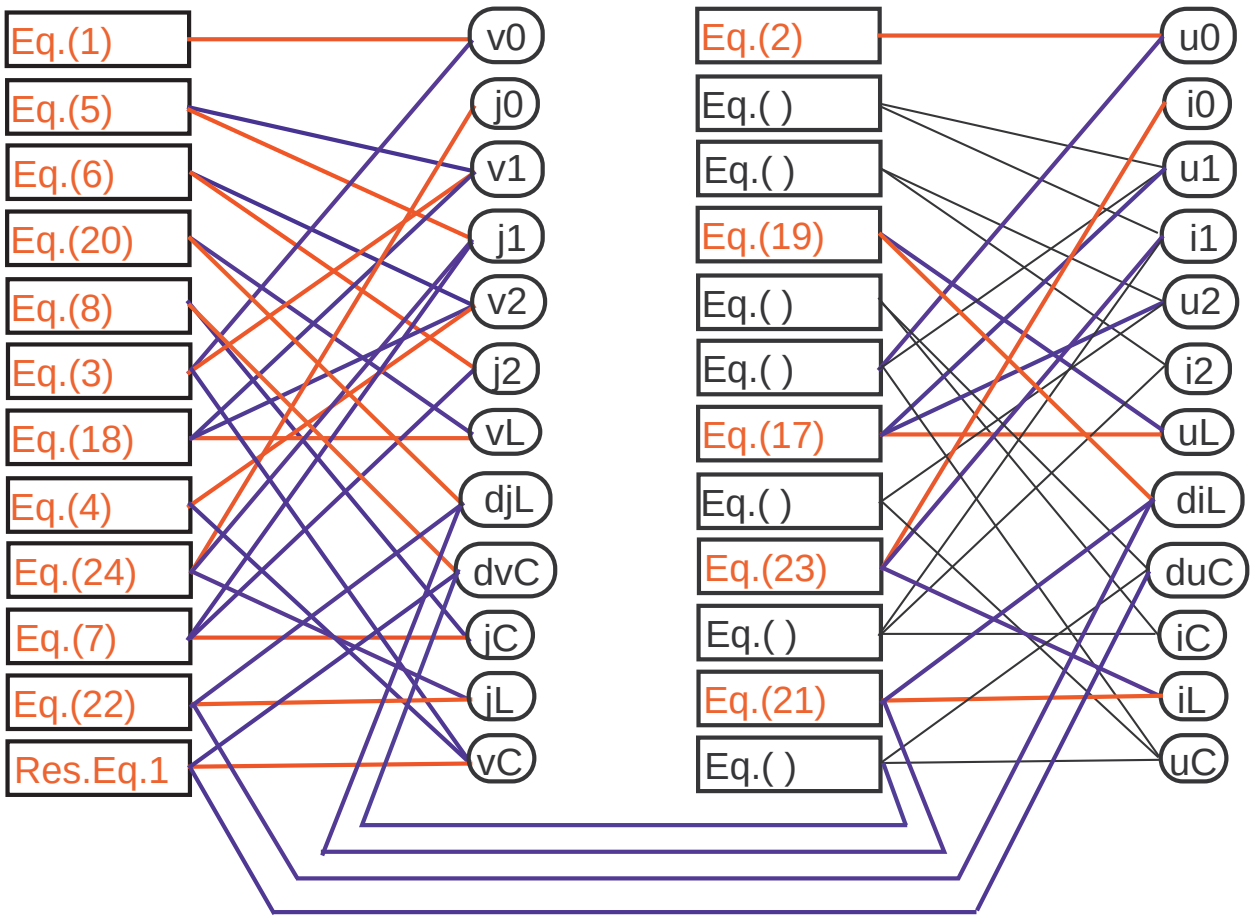
Inline Integration of IRK Algorithms IV



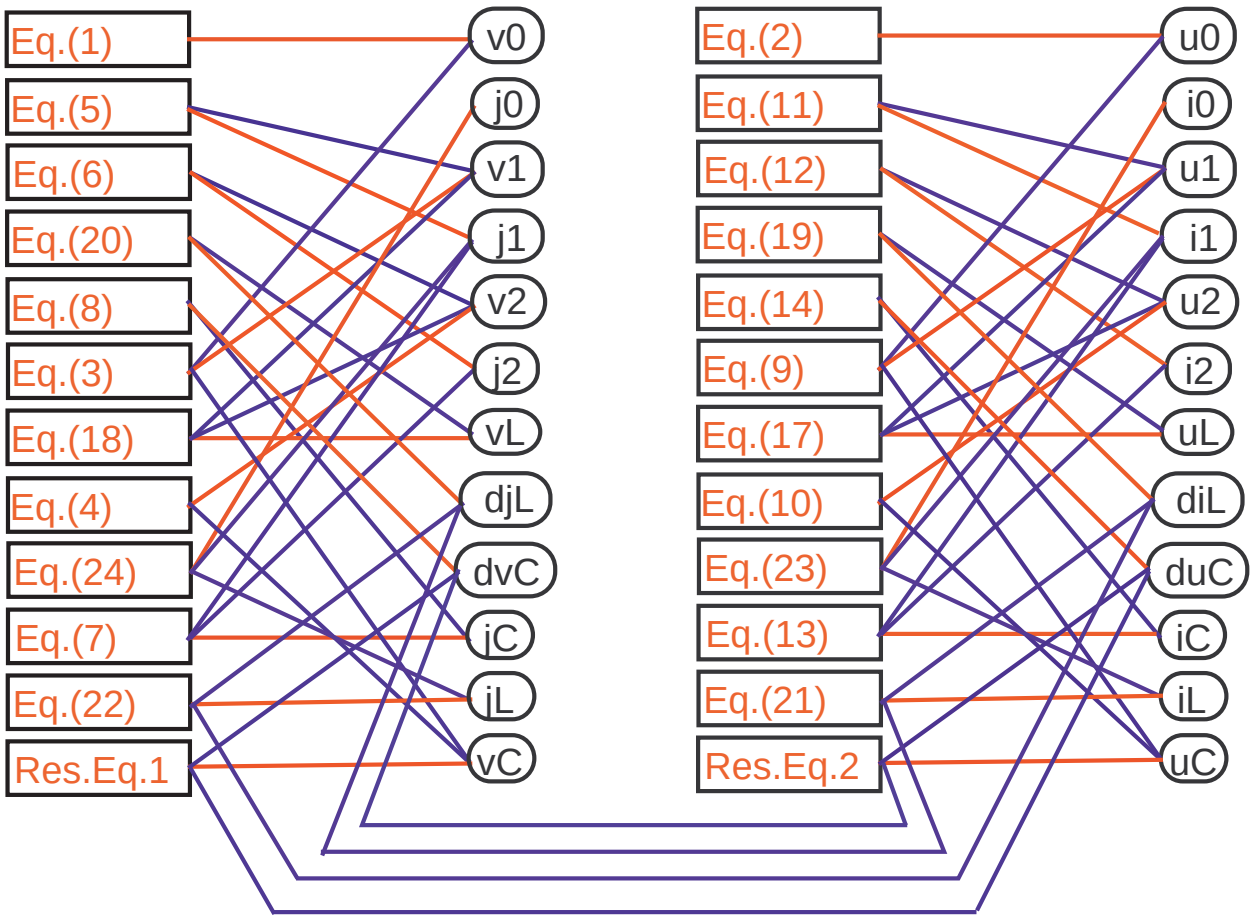
Inline Integration of IRK Algorithms IV



Inline Integration of IRK Algorithms IV



Inline Integration of IRK Algorithms IV



Inline Integration of IRK Algorithms V

- We were able to causalize 10 of the 24 equations at once.

Inline Integration of IRK Algorithms V

- We were able to causalize 10 of the 24 equations at once.
- We then needed to choose a first tearing variable and a first residual equation. We chose v_C as our first tearing variable. This allowed us to causalize seven additional equations.

Inline Integration of IRK Algorithms V

- We were able to causalize 10 of the 24 equations at once.
- We then needed to choose a first tearing variable and a first residual equation. We chose v_C as our first tearing variable. This allowed us to causalize seven additional equations.
- We then needed to select a second tearing variable and a second residual equation. We chose u_C as our second tearing variable. We now were able to causalize the remaining equations.

Inline Integration of IRK Algorithms V

- We were able to causalize 10 of the 24 equations at once.
- We then needed to choose a first tearing variable and a first residual equation. We chose v_C as our first tearing variable. This allowed us to causalize seven additional equations.
- We then needed to select a second tearing variable and a second residual equation. We chose u_C as our second tearing variable. We now were able to causalize the remaining equations.
- Comparing with the BDF algorithm, we notice that we have twice as many model equations. Also, every function evaluation now calls for a Newton iteration in 14 equations and two tearing variables instead of seven equations in a single tearing variable.

Inline Integration of IRK Algorithms V

- We were able to causalize 10 of the 24 equations at once.
- We then needed to choose a first tearing variable and a first residual equation. We chose v_C as our first tearing variable. This allowed us to causalize seven additional equations.
- We then needed to select a second tearing variable and a second residual equation. We chose u_C as our second tearing variable. We now were able to causalize the remaining equations.
- Comparing with the BDF algorithm, we notice that we have twice as many model equations. Also, every function evaluation now calls for a Newton iteration in 14 equations and two tearing variables instead of seven equations in a single tearing variable.
- However, the larger step sizes that we can use in the simulation with Radau IIA together with the much cheaper step-size control that the IRK algorithms offer, make it possible to predict that simulations with the inlined Radau IIA algorithm will likely be more economical than simulations using an inlined BDF method.

Stiffly-stable Step-size Control of Radau IIA(3) Method

- For step-size control, it is necessary to find an *estimate of the integration error*.

Stiffly-stable Step-size Control of Radau IIA(3) Method

- For step-size control, it is necessary to find an *estimate of the integration error*.
- Typically, designers of ODE and/or DAE solvers will look for a second solver of the same or higher order of approximation accuracy to compare it against the solver to be used for simulation.

Stiffly-stable Step-size Control of Radau IIA(3) Method

- For step-size control, it is necessary to find an *estimate of the integration error*.
- Typically, designers of ODE and/or DAE solvers will look for a second solver of the same or higher order of approximation accuracy to compare it against the solver to be used for simulation.
- While it is always possible to run two independent solvers in parallel for the purpose of step-size control, this approach is clearly undesirable, as it makes the solver highly inefficient.

Stiffly-stable Step-size Control of Radau IIA(3) Method

- For step-size control, it is necessary to find an *estimate of the integration error*.
- Typically, designers of ODE and/or DAE solvers will look for a second solver of the same or higher order of approximation accuracy to compare it against the solver to be used for simulation.
- While it is always possible to run two independent solvers in parallel for the purpose of step-size control, this approach is clearly undesirable, as it makes the solver highly inefficient.
- In explicit Runge-Kutta algorithms, it has become customary to search for an *embedding method*, i.e., a second solver that has most of the computations in common with the original solver, such that they share a large portion of the computational load between them.

Stiffly-stable Step-size Control of Radau IIA(3) Method

- For step-size control, it is necessary to find an *estimate of the integration error*.
- Typically, designers of ODE and/or DAE solvers will look for a second solver of the same or higher order of approximation accuracy to compare it against the solver to be used for simulation.
- While it is always possible to run two independent solvers in parallel for the purpose of step-size control, this approach is clearly undesirable, as it makes the solver highly inefficient.
- In explicit Runge-Kutta algorithms, it has become customary to search for an *embedding method*, i.e., a second solver that has most of the computations in common with the original solver, such that they share a large portion of the computational load between them.
- Unfortunately, this approach won't work in the case of fully implicit Runge-Kutta algorithms, since these algorithms are so compact and so highly optimized that there simply is not enough freedom left in these algorithms for embedding methods to co-exist with them.

Stiffly-stable Step-size Control of Radau IIA(3) Method II

- One solution that comes to mind immediately is to run a *BDF in parallel with the IRK technique*. This solution can be implemented cheaply, because an appropriately accurate state derivative at time t_{k+1} can be obtained using the Runge-Kutta approximation, i.e., no Newton iteration is necessary.

Stiffly-stable Step-size Control of Radau IIA(3) Method II

- One solution that comes to mind immediately is to run a *BDF in parallel with the IRK technique*. This solution can be implemented cheaply, because an appropriately accurate state derivative at time t_{k+1} can be obtained using the Runge-Kutta approximation, i.e., no Newton iteration is necessary.

- For example, the *3^{rd}-order accurate Radau IIA algorithm* could be accompanied by a *3^{rd}-order accurate BDF solver* implemented as:

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

where $\mathbf{f}_{\mathbf{k+1}}$ is the function value evaluated from the model equations at time t_{k+1}:

$$\mathbf{f}_{\mathbf{k+1}} = \mathbf{f}(\mathbf{x}_{\mathbf{k+1}}^{\mathbf{IRK}}, \mathbf{u}_{\mathbf{k+1}}, t_{k+1})$$

and $\mathbf{x}_{\mathbf{k+1}}^{\mathbf{IRK}}$ is the solution found by the Radau IIA algorithm.

Stiffly-stable Step-size Control of Radau IIA(3) Method II

- One solution that comes to mind immediately is to run a *BDF in parallel with the IRK technique*. This solution can be implemented cheaply, because an appropriately accurate state derivative at time t_{k+1} can be obtained using the Runge-Kutta approximation, i.e., no Newton iteration is necessary.

- For example, the *3^{rd}-order accurate Radau IIA algorithm* could be accompanied by a *3^{rd}-order accurate BDF solver* implemented as:

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

where $\mathbf{f}_{\mathbf{k+1}}$ is the function value evaluated from the model equations at time t_{k+1}:

$$\mathbf{f}_{\mathbf{k+1}} = \mathbf{f}(\mathbf{x}_{\mathbf{k+1}}^{\mathbf{IRK}}, \mathbf{u}_{\mathbf{k+1}}, t_{k+1})$$

and $\mathbf{x}_{\mathbf{k+1}}^{\mathbf{IRK}}$ is the solution found by the Radau IIA algorithm.

- Unfortunately, such a solution inherits all the difficulties associated with step-size control in linear multi-step methods.

Stiffly-stable Step-size Control of Radau IIA(3) Method III

What other options do we have?

Stiffly-stable Step-size Control of Radau IIA(3) Method III

What other options do we have?

- Clearly, an embedding method cannot be found using only information that is being used by the Radau IIA algorithm. In each step, there are four pieces of information available: $\mathbf{x_k}$, $\mathbf{x_{k+\frac{1}{3}}}$, $\mathbf{\dot{x}_{k+\frac{1}{3}}}$, and $\mathbf{\dot{x}_{k+1}}$ to estimate $\mathbf{x_{k+1}}$.

Stiffly-stable Step-size Control of Radau IIA(3) Method III

What other options do we have?

- Clearly, an embedding method cannot be found using only information that is being used by the Radau IIA algorithm. In each step, there are four pieces of information available: $\mathbf{x_k}$, $\mathbf{x_{k+\frac{1}{3}}}$, $\mathbf{\dot{x}_{k+\frac{1}{3}}}$, and $\mathbf{\dot{x}_{k+1}}$ to estimate $\mathbf{x_{k+1}}$.

- There is only one 3^{rd}-order polynomial going through these four pieces of information, and that polynomial wouldn't even be 3^{rd}-order accurate, because the intermediate computations of the method, $\mathbf{x_{k+\frac{1}{3}}}$ and $\mathbf{\dot{x}_{k+\frac{1}{3}}}$, are themselves only 2^{nd}-order accurate.

Stiffly-stable Step-size Control of Radau IIA(3) Method III

What other options do we have?

- Clearly, an embedding method cannot be found using only information that is being used by the Radau IIA algorithm. In each step, there are four pieces of information available: $\mathbf{x_k}$, $\mathbf{x_{k+\frac{1}{3}}}$, $\mathbf{\dot{x}_{k+\frac{1}{3}}}$, and $\mathbf{\dot{x}_{k+1}}$ to estimate $\mathbf{x_{k+1}}$.
- There is only one 3^{rd}-order polynomial going through these four pieces of information, and that polynomial wouldn't even be 3^{rd}-order accurate, because the intermediate computations of the method, $\mathbf{x_{k+\frac{1}{3}}}$ and $\mathbf{\dot{x}_{k+\frac{1}{3}}}$, are themselves only 2^{nd}-order accurate.
- However, enough redundancy can be obtained to define an embedding algorithm if *information from the two last steps* is being used. In this case, the following eight pieces of information are available: $\mathbf{x_{k-1}}$, $\mathbf{x_{k-\frac{2}{3}}}$, $\mathbf{x_k}$, $\mathbf{x_{k+\frac{1}{3}}}$, $\mathbf{\dot{x}_{k-\frac{2}{3}}}$, $\mathbf{\dot{x}_k}$, $\mathbf{\dot{x}_{k+\frac{1}{3}}}$, and $\mathbf{\dot{x}_{k+1}}$.

Stiffly-stable Step-size Control of Radau IIA(3) Method IV

- It was decided to look for 4^{th}-order polynomials that go through any five of these eight pieces of information. This technique defines 56 possible embedding methods.

Stiffly-stable Step-size Control of Radau IIA(3) Method IV

- It was decided to look for 4^{th}-order polynomials that go through any five of these eight pieces of information. This technique defines 56 possible embedding methods.
- Out of these 56 methods, only six are stiffly stable.

Stiffly-stable Step-size Control of Radau IIA(3) Method IV

- It was decided to look for 4^{th}-order polynomials that go through any five of these eight pieces of information. This technique defines 56 possible embedding methods.
- Out of these 56 methods, only six are stiffly stable.
- Two of those six techniques are not A-stable. One method has a stability domain with a discontinuous derivative at the real axis, which is suspicious.

Stiffly-stable Step-size Control of Radau IIA(3) Method IV

- It was decided to look for 4^{th}-order polynomials that go through any five of these eight pieces of information. This technique defines 56 possible embedding methods.
- Out of these 56 methods, only six are stiffly stable.
- Two of those six techniques are not A-stable. One method has a stability domain with a discontinuous derivative at the real axis, which is suspicious.
- The remaining three methods are:

$$
\begin{aligned}
x^1_{k+1} &= -\frac{25}{279}\,x_{k-1} + \frac{6}{31}\,x_{k-\frac{2}{3}} + \frac{250}{279}\,x_k + \frac{25h}{31}\,\dot{x}_{k+\frac{1}{3}} + \frac{65h}{279}\,\dot{x}_{k+1} \\
x^2_{k+1} &= -\frac{1}{36}\,x_{k-1} + \frac{16}{9}\,x_k - \frac{3}{4}\,x_{k+\frac{1}{3}} + h\,\dot{x}_{k+\frac{1}{3}} + \frac{2h}{9}\,\dot{x}_{k+1} \\
x^3_{k+1} &= -\frac{2}{23}\,x_{k-\frac{2}{3}} + \frac{50}{23}\,x_k - \frac{25}{23}\,x_{k+\frac{1}{3}} + \frac{25h}{23}\,\dot{x}_{k+\frac{1}{3}} + \frac{5h}{23}\,\dot{x}_{k+1}
\end{aligned}
$$

Stiffly-stable Step-size Control of Radau IIA(3) Method V

It is possible to write these methods in the linear case as:

$$\mathbf{x}_{k+1} = \mathbf{F} \cdot \mathbf{x}_{k-1}$$

Stiffly-stable Step-size Control of Radau IIA(3) Method V

It is possible to write these methods in the linear case as:

$$\mathbf{x}_{k+1} = \mathbf{F} \cdot \mathbf{x}_{k-1}$$

The **F**-matrices of the three methods can be expanded into Taylor series around $h = 0$:

$$\mathbf{F}^1 \approx \mathbf{I}^{(n)} + 2\,\mathbf{A}h + 4\,\frac{(\mathbf{A}h)^2}{2} + \frac{2224}{279}\,\frac{(\mathbf{A}h)^3}{6} + \frac{877}{58}\,\frac{(\mathbf{A}h)^4}{24}$$

$$\mathbf{F}^2 \approx \mathbf{I}^{(n)} + 2\,\mathbf{A}h + 4\,\frac{(\mathbf{A}h)^2}{2} + \frac{73}{9}\,\frac{(\mathbf{A}h)^3}{6} + \frac{859}{54}\,\frac{(\mathbf{A}h)^4}{24}$$

$$\mathbf{F}^3 \approx \mathbf{I}^{(n)} + 2\,\mathbf{A}h + 4\,\frac{(\mathbf{A}h)^2}{2} + \frac{188}{23}\,\frac{(\mathbf{A}h)^3}{6} + \frac{374}{23}\,\frac{(\mathbf{A}h)^4}{24}$$

Stiffly-stable Step-size Control of Radau IIA(3) Method V

It is possible to write these methods in the linear case as:

$$\mathbf{x}_{k+1} = \mathbf{F} \cdot \mathbf{x}_{k-1}$$

The **F**-matrices of the three methods can be expanded into Taylor series around $h = 0$:

$$\mathbf{F}^1 \approx \mathbf{I}^{(n)} + 2\,\mathbf{A}h + 4\,\frac{(\mathbf{A}h)^2}{2} + \frac{2224}{279}\,\frac{(\mathbf{A}h)^3}{6} + \frac{877}{58}\,\frac{(\mathbf{A}h)^4}{24}$$

$$\mathbf{F}^2 \approx \mathbf{I}^{(n)} + 2\,\mathbf{A}h + 4\,\frac{(\mathbf{A}h)^2}{2} + \frac{73}{9}\,\frac{(\mathbf{A}h)^3}{6} + \frac{859}{54}\,\frac{(\mathbf{A}h)^4}{24}$$

$$\mathbf{F}^3 \approx \mathbf{I}^{(n)} + 2\,\mathbf{A}h + 4\,\frac{(\mathbf{A}h)^2}{2} + \frac{188}{23}\,\frac{(\mathbf{A}h)^3}{6} + \frac{374}{23}\,\frac{(\mathbf{A}h)^4}{24}$$

A 4^{th}-order accurate method should have the **F**-matrix:

$$\mathbf{F} \approx \mathbf{I}^{(n)} + 2\,\mathbf{A}h + 4\,\frac{(\mathbf{A}h)^2}{2} + 8\,\frac{(\mathbf{A}h)^3}{6} + 16\,\frac{(\mathbf{A}h)^4}{24}$$

Stiffly-stable Step-size Control of Radau IIA(3) Method V

It is possible to write these methods in the linear case as:

$$\mathbf{x}_{k+1} = \mathbf{F} \cdot \mathbf{x}_{k-1}$$

The **F**-matrices of the three methods can be expanded into Taylor series around $h = 0$:

$$\mathbf{F}^1 \approx \mathbf{I}^{(n)} + 2\,\mathbf{A}h + 4\,\frac{(\mathbf{A}h)^2}{2} + \frac{2224}{279}\,\frac{(\mathbf{A}h)^3}{6} + \frac{877}{58}\,\frac{(\mathbf{A}h)^4}{24}$$

$$\mathbf{F}^2 \approx \mathbf{I}^{(n)} + 2\,\mathbf{A}h + 4\,\frac{(\mathbf{A}h)^2}{2} + \frac{73}{9}\,\frac{(\mathbf{A}h)^3}{6} + \frac{859}{54}\,\frac{(\mathbf{A}h)^4}{24}$$

$$\mathbf{F}^3 \approx \mathbf{I}^{(n)} + 2\,\mathbf{A}h + 4\,\frac{(\mathbf{A}h)^2}{2} + \frac{188}{23}\,\frac{(\mathbf{A}h)^3}{6} + \frac{374}{23}\,\frac{(\mathbf{A}h)^4}{24}$$

A 4^{th}-order accurate method should have the **F**-matrix:

$$\mathbf{F} \approx \mathbf{I}^{(n)} + 2\,\mathbf{A}h + 4\,\frac{(\mathbf{A}h)^2}{2} + 8\,\frac{(\mathbf{A}h)^3}{6} + 16\,\frac{(\mathbf{A}h)^4}{24}$$

All three methods are only 2^{nd}-order accurate, because they make use of the intermediate computations that are also only 2^{nd}-order accurate.

Stiffly-stable Step-size Control of Radau IIA(3) Method VI

In order to get a 3^{rd}-order accurate embedding method, it suffices to blend any two of these three methods, e.g.:

$$\mathbf{x}_{\mathbf{k+1}}^{\mathbf{blended}} = \vartheta \cdot \mathbf{x}_{\mathbf{k+1}}^{\mathbf{1}} + (1 - \vartheta) \cdot \mathbf{x}_{\mathbf{k+1}}^{\mathbf{2}}$$

Stiffly-stable Step-size Control of Radau IIA(3) Method VI

In order to get a 3^{rd}-order accurate embedding method, it suffices to blend any two of these three methods, e.g.:

$$\mathbf{x}_{k+1}^{\text{blended}} = \vartheta \cdot \mathbf{x}_{k+1}^{1} + (1 - \vartheta) \cdot \mathbf{x}_{k+1}^{2}$$

The resulting method is:

$$\mathbf{x}_{k+1}^{\text{blended}} = -\frac{1}{13}\,\mathbf{x}_{k-1} + \frac{2}{13}\,\mathbf{x}_{k-\frac{2}{3}} + \frac{14}{13}\,\mathbf{x}_{k} - \frac{2}{13}\,\mathbf{x}_{k+\frac{1}{3}} + \frac{11h}{13}\,\dot{\mathbf{x}}_{k+\frac{1}{3}} + \frac{3h}{13}\,\dot{\mathbf{x}}_{k+1}$$

Stiffly-stable Step-size Control of Radau IIA(3) Method VI

In order to get a 3^{rd}-order accurate embedding method, it suffices to blend any two of these three methods, e.g.:

$$\mathbf{x}_{\mathbf{k+1}}^{\mathbf{blended}} = \vartheta \cdot \mathbf{x}_{\mathbf{k+1}}^{\mathbf{1}} + (1 - \vartheta) \cdot \mathbf{x}_{\mathbf{k+1}}^{\mathbf{2}}$$

The resulting method is:

$$\mathbf{x}_{\mathbf{k+1}}^{\mathbf{blended}} = -\frac{1}{13}\,\mathbf{x}_{\mathbf{k-1}} + \frac{2}{13}\,\mathbf{x}_{\mathbf{k-\frac{2}{3}}} + \frac{14}{13}\,\mathbf{x}_{\mathbf{k}} - \frac{2}{13}\,\mathbf{x}_{\mathbf{k+\frac{1}{3}}} + \frac{11h}{13}\,\dot{\mathbf{x}}_{\mathbf{k+\frac{1}{3}}} + \frac{3h}{13}\,\dot{\mathbf{x}}_{\mathbf{k+1}}$$

This method has beautifully simple rational coefficients. It is indeed 3^{rd}-order accurate:

$$\mathbf{F} \approx \mathbf{I}^{(\mathbf{n})} + 2\,\mathbf{A}h + 4\,\frac{(\mathbf{A}h)^2}{2} + 8\,\frac{(\mathbf{A}h)^3}{6} + \frac{149}{156} \cdot 16\,\frac{(\mathbf{A}h)^4}{24}$$

Stiffly-stable Step-size Control of Radau IIA(3) Method VI

In order to get a 3^{rd}-order accurate embedding method, it suffices to blend any two of these three methods, e.g.:

$$\mathbf{x}_{\mathbf{k+1}}^{\mathbf{blended}} = \vartheta \cdot \mathbf{x}_{\mathbf{k+1}}^{\mathbf{1}} + (1 - \vartheta) \cdot \mathbf{x}_{\mathbf{k+1}}^{\mathbf{2}}$$

The resulting method is:

$$\mathbf{x}_{\mathbf{k+1}}^{\mathbf{blended}} = -\frac{1}{13}\,\mathbf{x}_{\mathbf{k-1}} + \frac{2}{13}\,\mathbf{x}_{\mathbf{k-\frac{2}{3}}} + \frac{14}{13}\,\mathbf{x}_{\mathbf{k}} - \frac{2}{13}\,\mathbf{x}_{\mathbf{k+\frac{1}{3}}} + \frac{11h}{13}\,\dot{\mathbf{x}}_{\mathbf{k+\frac{1}{3}}} + \frac{3h}{13}\,\dot{\mathbf{x}}_{\mathbf{k+1}}$$

This method has beautifully simple rational coefficients. It is indeed 3^{rd}-order accurate:

$$\mathbf{F} \approx \mathbf{I}^{(\mathbf{n})} + 2\,\mathbf{A}h + 4\,\frac{(\mathbf{A}h)^2}{2} + 8\,\frac{(\mathbf{A}h)^3}{6} + \frac{149}{156} \cdot 16\,\frac{(\mathbf{A}h)^4}{24}$$

The error coefficient of the method is:

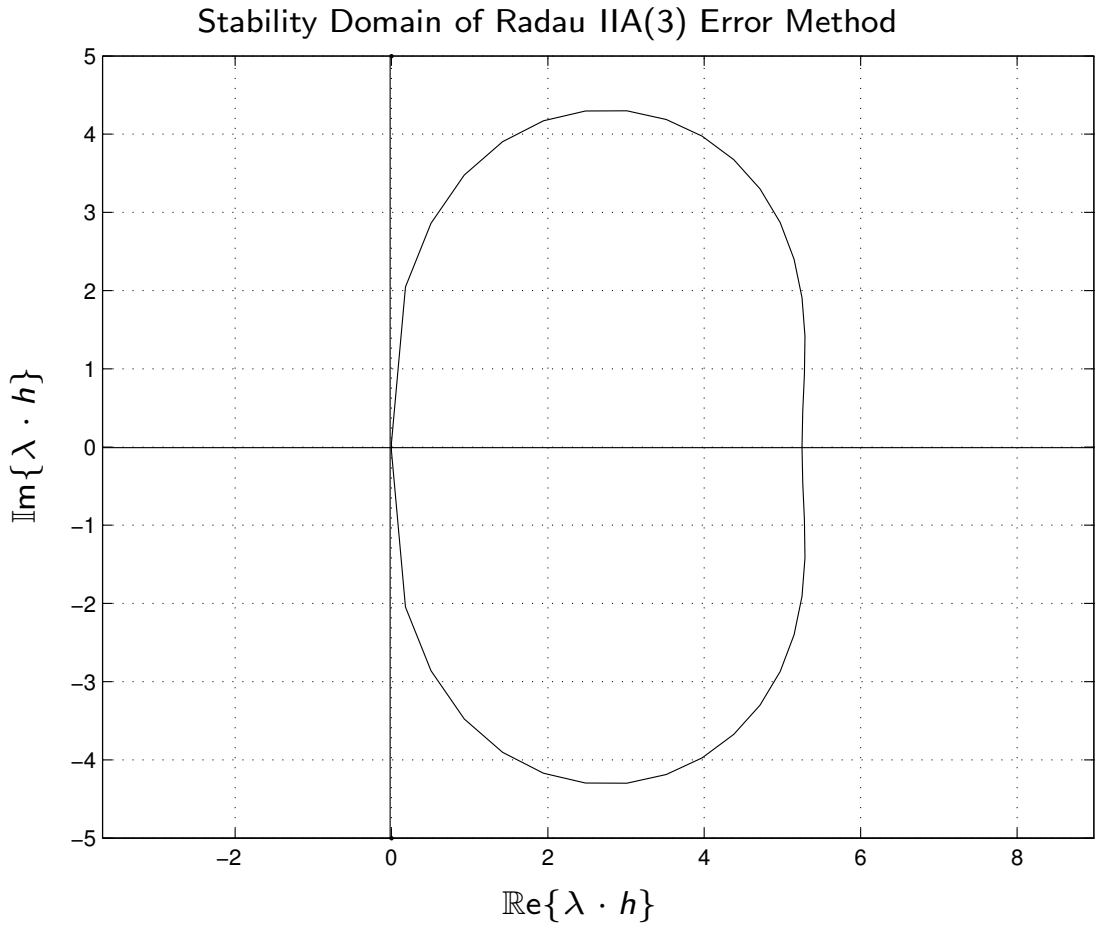
$$\varepsilon = \frac{-7}{3744}(\mathbf{A}h)^4$$

Stiffly-stable Step-size Control of Radau IIA(3) Method VII

We can plot the stability domain of the embedding method:

Stiffly-stable Step-size Control of Radau IIA(3) Method VII

We can plot the stability domain of the embedding method:

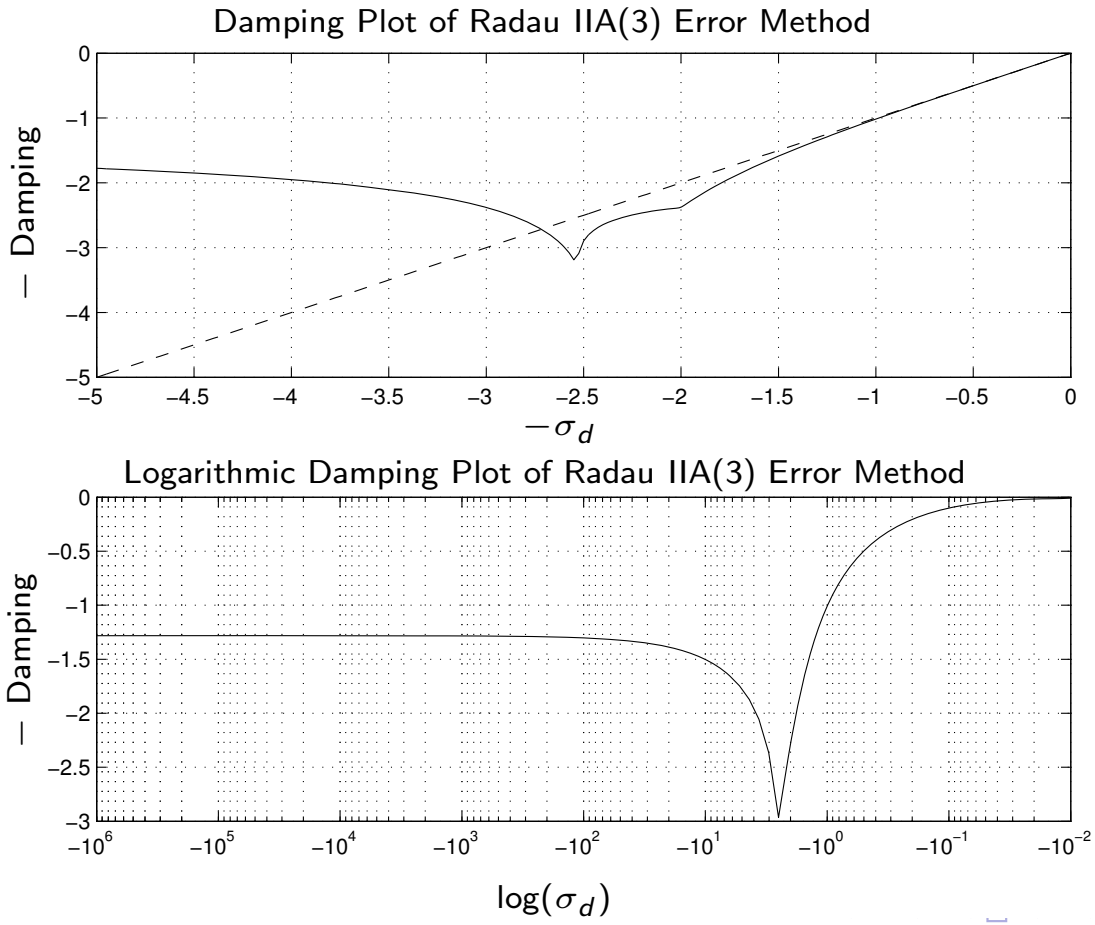


Stiffly-stable Step-size Control of Radau IIA(3) Method VIII

We can also draw the damping plot of the embedding method:

Stiffly-stable Step-size Control of Radau IIA(3) Method VIII

We can also draw the damping plot of the embedding method:



Stiffly-stable Step-size Control of Radau IIA(3) Method IX

Which method should be propagated?

Stiffly-stable Step-size Control of Radau IIA(3) Method IX

Which method should be propagated?

The error coefficient of the embedding method is considerably smaller than that of Radau IIA. Hence on a first glance, it seems reasonable to propagate the embedding technique to the next step.

Stiffly-stable Step-size Control of Radau IIA(3) Method IX

Which method should be propagated?

The error coefficient of the embedding method is considerably smaller than that of Radau IIA. Hence on a first glance, it seems reasonable to propagate the embedding technique to the next step.

Unfortunately, there are two problems with this choice:

Stiffly-stable Step-size Control of Radau IIA(3) Method IX

Which method should be propagated?

The error coefficient of the embedding method is considerably smaller than that of Radau IIA. Hence on a first glance, it seems reasonable to propagate the embedding technique to the next step.

Unfortunately, there are two problems with this choice:

1. The embedding technique was designed assuming that the Radau IIA result would be propagated. If the embedding technique is being propagated, the **F**-matrices change, and the blended method may no longer be 3^{rd}-order accurate.

Stiffly-stable Step-size Control of Radau IIA(3) Method IX

Which method should be propagated?

The error coefficient of the embedding method is considerably smaller than that of Radau IIA. Hence on a first glance, it seems reasonable to propagate the embedding technique to the next step.

Unfortunately, there are two problems with this choice:

1. The embedding technique was designed assuming that the Radau IIA result would be propagated. If the embedding technique is being propagated, the **F**-matrices change, and the blended method may no longer be 3^{rd}-order accurate.
2. Comparing the damping plots of Radau IIA(3) and the embedding method with each other, it can be seen that *the embedding method is not L-stable*, i.e., the damping does not approach infinity as the eigenvalues of the model move ever further to the left.

Stiffly-stable Step-size Control of Radau IIA(3) Method IX

Which method should be propagated?

The error coefficient of the embedding method is considerably smaller than that of Radau IIA. Hence on a first glance, it seems reasonable to propagate the embedding technique to the next step.

Unfortunately, there are two problems with this choice:

1. The embedding technique was designed assuming that the Radau IIA result would be propagated. If the embedding technique is being propagated, the **F**-matrices change, and the blended method may no longer be 3^{rd}-order accurate.
2. Comparing the damping plots of Radau IIA(3) and the embedding method with each other, it can be seen that *the embedding method is not L-stable*, i.e., the damping does not approach infinity as the eigenvalues of the model move ever further to the left.

Therefore, in spite of the smaller error coefficient, the embedding method should be used for step-size control only and not for propagation.

Stiffly-stable Step-size Control of Radau IIA(3) Method X

What is the price that we pay for extending the embedding method over two steps instead of one?

Stiffly-stable Step-size Control of Radau IIA(3) Method X

What is the price that we pay for extending the embedding method over two steps instead of one?

- Step-size control is only performed every other step, i.e., we simulate across two steps with the same step size using Radau IIA(3).

Stiffly-stable Step-size Control of Radau IIA(3) Method X

What is the price that we pay for extending the embedding method over two steps instead of one?

- Step-size control is only performed every other step, i.e., we simulate across two steps with the same step size using Radau IIA(3).
- Only after the second step has been simulated, the (blended) embedding method is computed in order to obtain an error estimate for the method.

Stiffly-stable Step-size Control of Radau IIA(3) Method X

What is the price that we pay for extending the embedding method over two steps instead of one?

- Step-size control is only performed every other step, i.e., we simulate across two steps with the same step size using Radau IIA(3).
- Only after the second step has been simulated, the (blended) embedding method is computed in order to obtain an error estimate for the method.
- As a consequence of that error estimate, we either continue as before, modify the step size for the next two steps, or reject the last two steps and repeat them with a smaller step size.

Stiffly-stable Step-size Control of Radau IIA(3) Method X

What is the price that we pay for extending the embedding method over two steps instead of one?

- Step-size control is only performed every other step, i.e., we simulate across two steps with the same step size using Radau IIA(3).
- Only after the second step has been simulated, the (blended) embedding method is computed in order to obtain an error estimate for the method.
- As a consequence of that error estimate, we either continue as before, modify the step size for the next two steps, or reject the last two steps and repeat them with a smaller step size.
- Although the embedding method extends over two steps, the overall algorithm is self-starting. It can be considered as a single-step method with double step size and two semi-steps.

Stiffly-stable Step-size Control of Radau IIA(5) Method

- **Radau IIA(5)** stores six pieces of information per step, namely $\mathbf{x_k}$, $\mathbf{x_{1_k}}$, $\mathbf{x_{2_k}}$, $\dot{\mathbf{x}}_{\mathbf{1_k}}$, $\dot{\mathbf{x}}_{\mathbf{2_k}}$, and $\dot{\mathbf{x}}_{\mathbf{k+1}}$, where $\mathbf{x_{1_k}}$ and $\mathbf{x_{2_k}}$ are the approximations of the two intermediate stages.

Stiffly-stable Step-size Control of Radau IIA(5) Method

- **Radau IIA(5)** stores six pieces of information per step, namely $\mathbf{x_k}$, $\mathbf{x_{1_k}}$, $\mathbf{x_{2_k}}$, $\dot{\mathbf{x}}_{\mathbf{1_k}}$, $\dot{\mathbf{x}}_{\mathbf{2_k}}$, and $\dot{\mathbf{x}}_{\mathbf{k+1}}$, where $\mathbf{x_{1_k}}$ and $\mathbf{x_{2_k}}$ are the approximations of the two intermediate stages.

- We decided to look for 6^{th}-order polynomials extending over two steps. Hence we need to choose seven out of 12 available support values. There are 792 methods to be evaluated.

Stiffly-stable Step-size Control of Radau IIA(5) Method

- **Radau IIA(5)** stores six pieces of information per step, namely $\mathbf{x_k}$, $\mathbf{x_{1_k}}$, $\mathbf{x_{2_k}}$, $\dot{\mathbf{x}}_{\mathbf{1_k}}$, $\dot{\mathbf{x}}_{\mathbf{2_k}}$, and $\dot{\mathbf{x}}_{\mathbf{k+1}}$, where $\mathbf{x_{1_k}}$ and $\mathbf{x_{2_k}}$ are the approximations of the two intermediate stages.

- We decided to look for 6^{th}-order polynomials extending over two steps. Hence we need to choose seven out of 12 available support values. There are 792 methods to be evaluated.

- Of those, 26 are A-stable methods. These are only 3^{rd}-order accurate, as the intermediate computations of Radau IIA(5) are only 3^{rd}-order accurate. However, three of these methods can be blended to form an alternate 5^{th}-order accurate embedding algorithm.

Stiffly-stable Step-size Control of Radau IIA(5) Method II

A very good embedding method is:

$$\mathbf{x}_{k+1}^{\text{blended}} = c_1 \cdot \mathbf{x}_{k-1} + c_2 \cdot \dot{\mathbf{x}}_{1_{k-1}} + c_3 \cdot \mathbf{x}_{2_{k-1}} + c_4 \cdot \dot{\mathbf{x}}_{2_{k-1}} + c_5 \cdot \mathbf{x}_k + c_6 \cdot \mathbf{x}_{1_k} + c_7 \cdot \dot{\mathbf{x}}_{1_k} + c_8 \cdot \mathbf{x}_{2_k} + c_9 \cdot \dot{\mathbf{x}}_{2_k} + c_{10} \cdot \dot{\mathbf{x}}_{k+1}$$

with the coefficients:

$$\begin{aligned} c_1 &= -0.00517140382204 \\ c_2 &= -0.00094714677404 \\ c_3 &= -0.04060469717694 \\ c_4 &= -0.01364429384901 \\ c_5 &= +1.41786808325433 \\ c_6 &= -0.17475783086782 \\ c_7 &= +0.48299282769491 \\ c_8 &= -0.19733415138754 \\ c_9 &= +0.55942205973218 \\ c_{10} &= +0.10695524944855 \end{aligned}$$

Stiffly-stable Step-size Control of Radau IIA(5) Method II

A very good embedding method is:

$$\mathbf{x}_{k+1}^{\text{blended}} = c_1 \cdot \mathbf{x}_{k-1} + c_2 \cdot \dot{\mathbf{x}}_{1_{k-1}} + c_3 \cdot \mathbf{x}_{2_{k-1}} + c_4 \cdot \dot{\mathbf{x}}_{2_{k-1}} + c_5 \cdot \mathbf{x}_k + c_6 \cdot \mathbf{x}_{1_k} + c_7 \cdot \dot{\mathbf{x}}_{1_k} + c_8 \cdot \mathbf{x}_{2_k} + c_9 \cdot \dot{\mathbf{x}}_{2_k} + c_{10} \cdot \dot{\mathbf{x}}_{k+1}$$

with the coefficients:

$$\begin{aligned} c_1 &= -0.00517140382204 \\ c_2 &= -0.00094714677404 \\ c_3 &= -0.04060469717694 \\ c_4 &= -0.01364429384901 \\ c_5 &= +1.41786808325433 \\ c_6 &= -0.17475783086782 \\ c_7 &= +0.48299282769491 \\ c_8 &= -0.19733415138754 \\ c_9 &= +0.55942205973218 \\ c_{10} &= +0.10695524944855 \end{aligned}$$

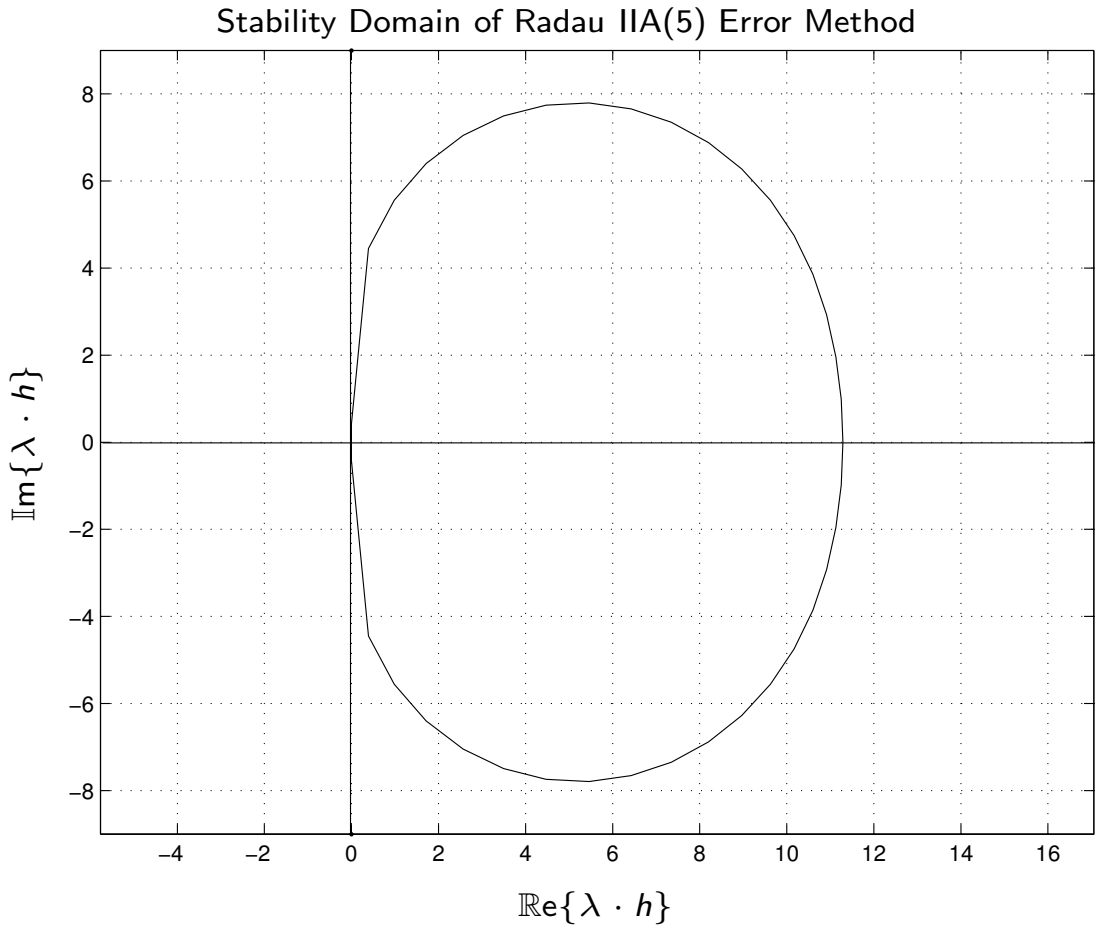
We did program the computation of the coefficients also using MATLAB's symbolic toolbox, but the resulting expressions look quite awful, thus we decided to offer the numerical versions instead.

Stiffly-stable Step-size Control of Radau IIA(5) Method III

We can plot the stability domain of the embedding method:

Stiffly-stable Step-size Control of Radau IIA(5) Method III

We can plot the stability domain of the embedding method:

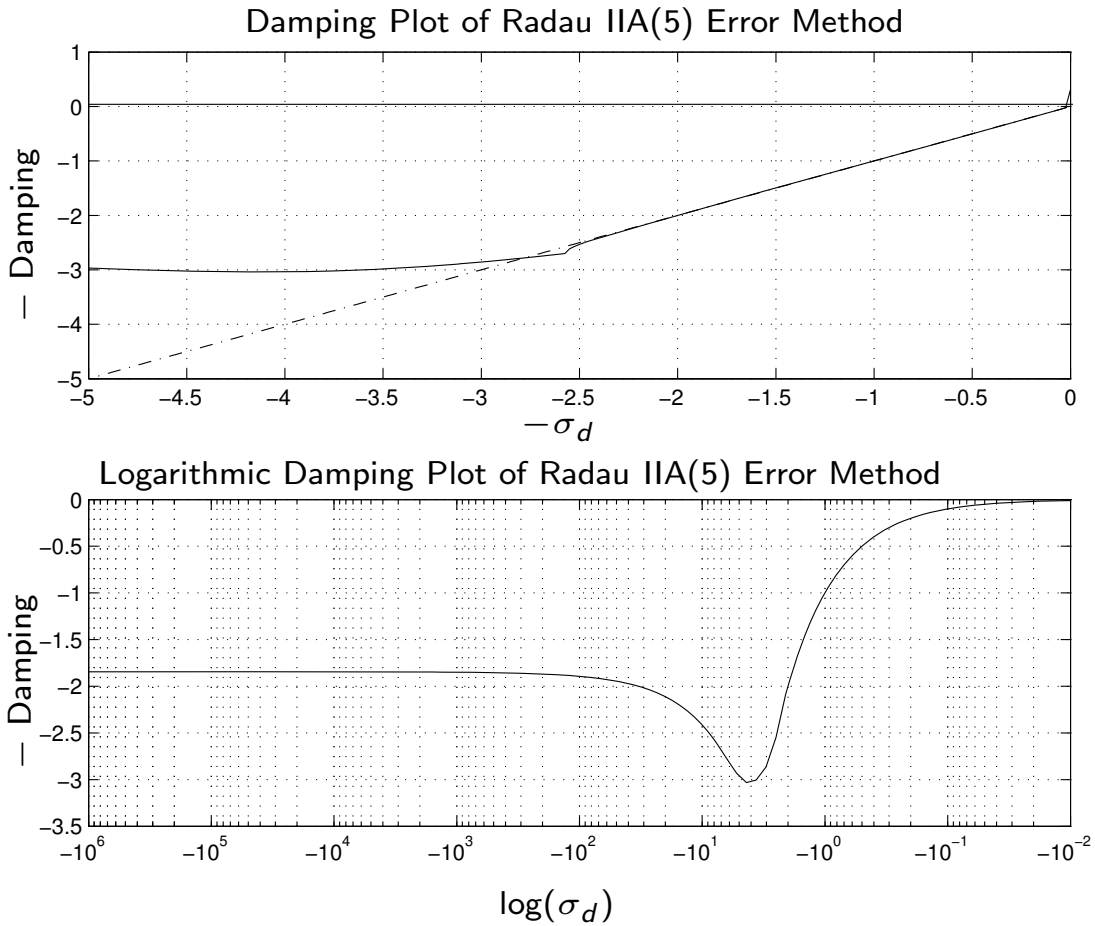


Stiffly-stable Step-size Control of Radau IIA(5) Method IV

We can also draw the damping plot of the embedding method:

Stiffly-stable Step-size Control of Radau IIA(5) Method IV

We can also draw the damping plot of the embedding method:



Stiffly-stable Step-size Control of Lobatto IIIC Method

- **Lobatto IIIC** stores six pieces of information per step, namely $\mathbf{x_k}$, $\mathbf{x_{1_k}}$, $\mathbf{x_{2_k}}$, $\dot{\mathbf{x}}_{\mathbf{1_k}}$, $\dot{\mathbf{x}}_{\mathbf{2_k}}$, and $\dot{\mathbf{x}}_{\mathbf{k+1}}$, where $\mathbf{x_{1_k}}$ and $\mathbf{x_{2_k}}$ are the approximations of the two intermediate stages.

Stiffly-stable Step-size Control of Lobatto IIIC Method

- **Lobatto IIIC** stores six pieces of information per step, namely $\mathbf{x_k}$, $\mathbf{x_{1_k}}$, $\mathbf{x_{2_k}}$, $\dot{\mathbf{x}}_{\mathbf{1_k}}$, $\dot{\mathbf{x}}_{\mathbf{2_k}}$, and $\dot{\mathbf{x}}_{\mathbf{k+1}}$, where $\mathbf{x_{1_k}}$ and $\mathbf{x_{2_k}}$ are the approximations of the two intermediate stages.
- Unfortunately, the intermediate stages are only 2^{nd}-order accurate. Hence we shall still need to blend three methods to raise the order of approximation accuracy of the embedding algorithm to four.

Stiffly-stable Step-size Control of Lobatto IIIC Method

- **Lobatto IIIC** stores six pieces of information per step, namely $\mathbf{x_k}$, $\mathbf{x_{1_k}}$, $\mathbf{x_{2_k}}$, $\dot{\mathbf{x}}_{\mathbf{1_k}}$, $\dot{\mathbf{x}}_{\mathbf{2_k}}$, and $\dot{\mathbf{x}}_{\mathbf{k+1}}$, where $\mathbf{x_{1_k}}$ and $\mathbf{x_{2_k}}$ are the approximations of the two intermediate stages.
- Unfortunately, the intermediate stages are only 2^{nd}-order accurate. Hence we shall still need to blend three methods to raise the order of approximation accuracy of the embedding algorithm to four.
- Although we are again working with 12 pieces of information across two steps, Lobatto IIIC has a peculiarity. It experiences a zero time advance between the third stage of one step and the first stage of the next. Although $\mathbf{x_k}$ and $\mathbf{x_{1_k}}$ represent the state vector at the same time instant, they are two different approximations.

Stiffly-stable Step-size Control of Lobatto IIIC Method

- **Lobatto IIIC** stores six pieces of information per step, namely $\mathbf{x_k}$, $\mathbf{x_{1_k}}$, $\mathbf{x_{2_k}}$, $\dot{\mathbf{x}}_{\mathbf{1_k}}$, $\dot{\mathbf{x}}_{\mathbf{2_k}}$, and $\dot{\mathbf{x}}_{\mathbf{k+1}}$, where $\mathbf{x_{1_k}}$ and $\mathbf{x_{2_k}}$ are the approximations of the two intermediate stages.
- Unfortunately, the intermediate stages are only 2^{nd}-order accurate. Hence we shall still need to blend three methods to raise the order of approximation accuracy of the embedding algorithm to four.
- Although we are again working with 12 pieces of information across two steps, Lobatto IIIC has a peculiarity. It experiences a zero time advance between the third stage of one step and the first stage of the next. Although $\mathbf{x_k}$ and $\mathbf{x_{1_k}}$ represent the state vector at the same time instant, they are two different approximations.
- The zero time advance reduces the flexibility in finding suitable error methods, as no individual error method can use both $\mathbf{x_k}$ and $\mathbf{x_{1_k}}$ simultaneously.

Stiffly-stable Step-size Control of Lobatto IIIC Method

- **Lobatto IIIC** stores six pieces of information per step, namely $\mathbf{x_k}$, $\mathbf{x_{1_k}}$, $\mathbf{x_{2_k}}$, $\dot{\mathbf{x}}_{\mathbf{1_k}}$, $\dot{\mathbf{x}}_{\mathbf{2_k}}$, and $\dot{\mathbf{x}}_{\mathbf{k+1}}$, where $\mathbf{x_{1_k}}$ and $\mathbf{x_{2_k}}$ are the approximations of the two intermediate stages.
- Unfortunately, the intermediate stages are only 2^{nd}-order accurate. Hence we shall still need to blend three methods to raise the order of approximation accuracy of the embedding algorithm to four.
- Although we are again working with 12 pieces of information across two steps, Lobatto IIIC has a peculiarity. It experiences a zero time advance between the third stage of one step and the first stage of the next. Although $\mathbf{x_k}$ and $\mathbf{x_{1_k}}$ represent the state vector at the same time instant, they are two different approximations.
- The zero time advance reduces the flexibility in finding suitable error methods, as no individual error method can use both $\mathbf{x_k}$ and $\mathbf{x_{1_k}}$ simultaneously.
- 16 individual error methods were found that are all A-stable. Two of them are even L-stable. Of course, none of these error methods is of higher order of approximation accuracy than two.

Stiffly-stable Step-size Control of Lobatto IIIC Method II

The best among the 4^{th}-order accurate blended methods is:

$$\begin{aligned} \mathbf{x}_{\mathbf{k+1}}^{\mathbf{blended}} = & \frac{63}{4552} \cdot \mathbf{x}_{\mathbf{1}_{k-1}} - \frac{91}{81936} \cdot \dot{\mathbf{x}}_{\mathbf{1}_{k-1}} + \frac{1381}{81936} \cdot \dot{\mathbf{x}}_{\mathbf{2}_{k-1}} + \frac{3101}{2276} \cdot \mathbf{x}_{\mathbf{k}} \\ & - \frac{393}{4552} \cdot \mathbf{x}_{\mathbf{1}_{k}} + \frac{775}{3414} \cdot \dot{\mathbf{x}}_{\mathbf{1}_{k}} - \frac{165}{569} \cdot \mathbf{x}_{\mathbf{2}_{k}} + \frac{62179}{81936} \cdot \dot{\mathbf{x}}_{\mathbf{2}_{k}} + \frac{12881}{81936} \cdot \dot{\mathbf{x}}_{\mathbf{k+1}} \end{aligned}$$

Stiffly-stable Step-size Control of Lobatto IIIC Method II

The best among the 4^{th}-order accurate blended methods is:

$$\begin{aligned} \mathbf{x}_{k+1}^{\mathbf{blended}} = & \frac{63}{4552} \cdot \mathbf{x1}_{k-1} - \frac{91}{81936} \cdot \dot{\mathbf{x}}\mathbf{1}_{k-1} + \frac{1381}{81936} \cdot \dot{\mathbf{x}}\mathbf{2}_{k-1} + \frac{3101}{2276} \cdot \mathbf{x}_k \\ & - \frac{393}{4552} \cdot \mathbf{x1}_k + \frac{775}{3414} \cdot \dot{\mathbf{x}}\mathbf{1}_k - \frac{165}{569} \cdot \mathbf{x2}_k + \frac{62179}{81936} \cdot \dot{\mathbf{x}}\mathbf{2}_k + \frac{12881}{81936} \cdot \dot{\mathbf{x}}_{k+1} \end{aligned}$$

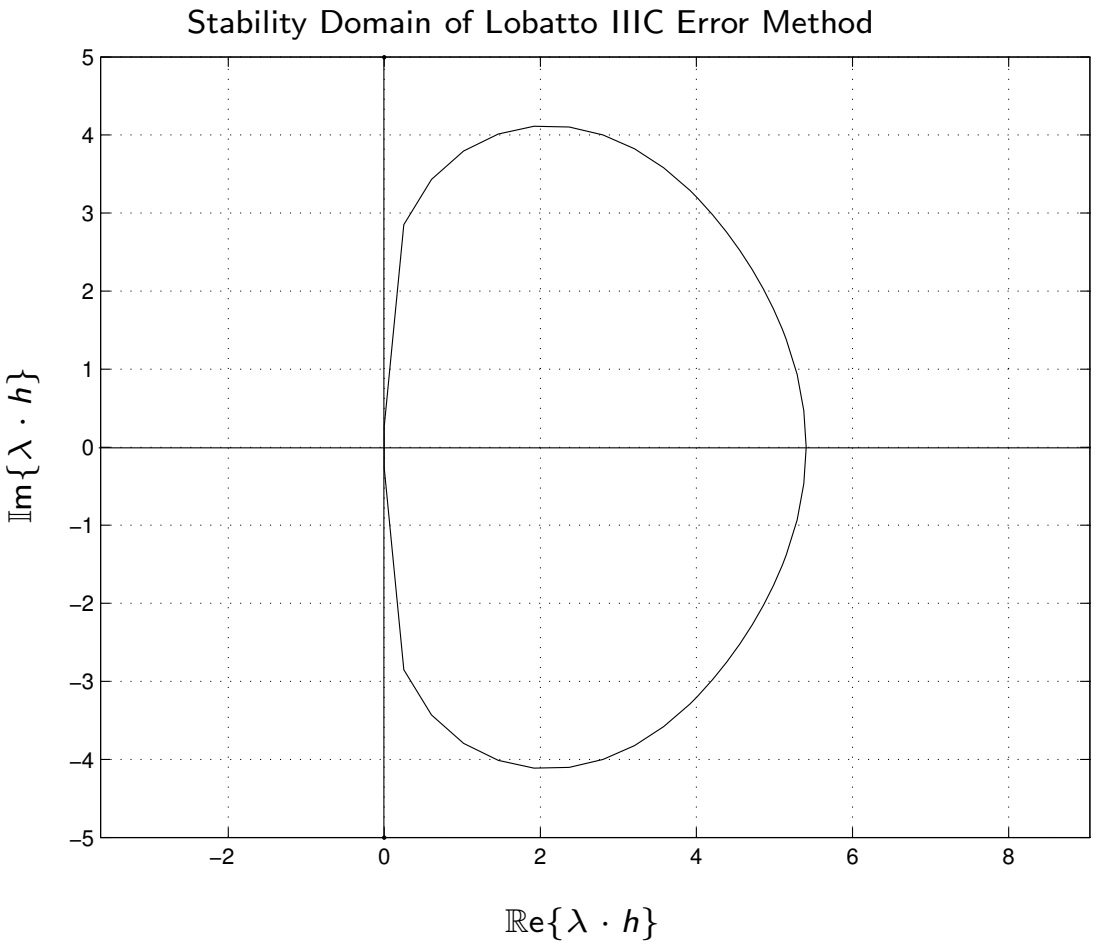
The coefficients of the embedding method were calculated using MATLAB's symbolic toolbox.

Stiffly-stable Step-size Control of Lobatto IIIC Method III

We can plot the stability domain of the embedding method:

Stiffly-stable Step-size Control of Lobatto IIIC Method III

We can plot the stability domain of the embedding method:

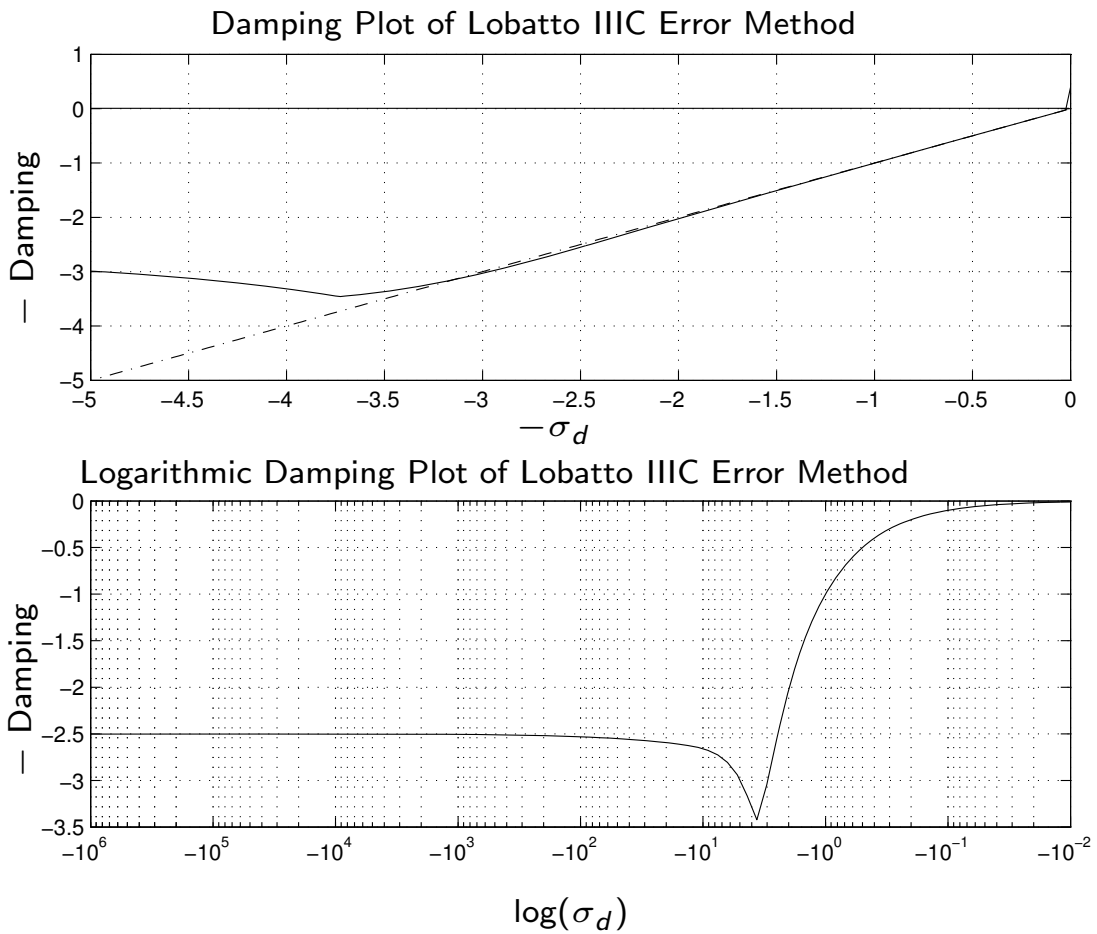


Stiffly-stable Step-size Control of Lobatto IIIC Method IV

We can also draw the damping plot of the embedding method:

Stiffly-stable Step-size Control of Lobatto IIIC Method IV

We can also draw the damping plot of the embedding method:



Inlining Partial Differential Equations

Let us return once more to the simulation of parabolic PDEs converted to sets of ODEs using the MOL approach.

Inlining Partial Differential Equations

Let us return once more to the simulation of parabolic PDEs converted to sets of ODEs using the MOL approach.

Until now, we simulated these types of problems using a stiff-system ODE solver, such as a BDF algorithm.

Inlining Partial Differential Equations

Let us return once more to the simulation of parabolic PDEs converted to sets of ODEs using the MOL approach.

Until now, we simulated these types of problems using a stiff-system ODE solver, such as a BDF algorithm.

This approach worked quite well; however, the efficiency of the simulations was less than satisfactory.

Inlining Partial Differential Equations

Let us return once more to the simulation of parabolic PDEs converted to sets of ODEs using the MOL approach.

Until now, we simulated these types of problems using a stiff-system ODE solver, such as a BDF algorithm.

This approach worked quite well; however, the efficiency of the simulations was less than satisfactory.

What killed the efficiency of our simulations were not small step sizes. What made our simulations excruciatingly slow was the computation of the inverse Hessians.

Inlining Partial Differential Equations

Let us return once more to the simulation of parabolic PDEs converted to sets of ODEs using the MOL approach.

Until now, we simulated these types of problems using a stiff-system ODE solver, such as a BDF algorithm.

This approach worked quite well; however, the efficiency of the simulations was less than satisfactory.

What killed the efficiency of our simulations were not small step sizes. What made our simulations excruciatingly slow was the computation of the inverse Hessians.

We wish to revisit the MOL simulation of the *1D heat diffusion problem* discretized using 5^{th}-order accurate central differences with 50 segments.

Inlining Partial Differential Equations

Let us return once more to the simulation of parabolic PDEs converted to sets of ODEs using the MOL approach.

Until now, we simulated these types of problems using a stiff-system ODE solver, such as a BDF algorithm.

This approach worked quite well; however, the efficiency of the simulations was less than satisfactory.

What killed the efficiency of our simulations were not small step sizes. What made our simulations excruciatingly slow was the computation of the inverse Hessians.

We wish to revisit the MOL simulation of the *1D heat diffusion problem* discretized using 5^{th}-order accurate central differences with 50 segments.

We ended up with 50 ODEs, requiring a Hessian matrix of size 50×50 to be inverted. More precisely, a linear system of 50 equations in 50 unknowns had to be solved using Gaussian elimination during every iteration step.

Inlining Partial Differential Equations II

Let us now apply *inline integration* to the problem. Let us start by inlining a variable step and variable order BDF algorithm.

Inlining Partial Differential Equations II

Let us now apply *inline integration* to the problem. Let us start by inlining a variable step and variable order BDF algorithm.

We can write the *inlined difference equation system* in matrix form:

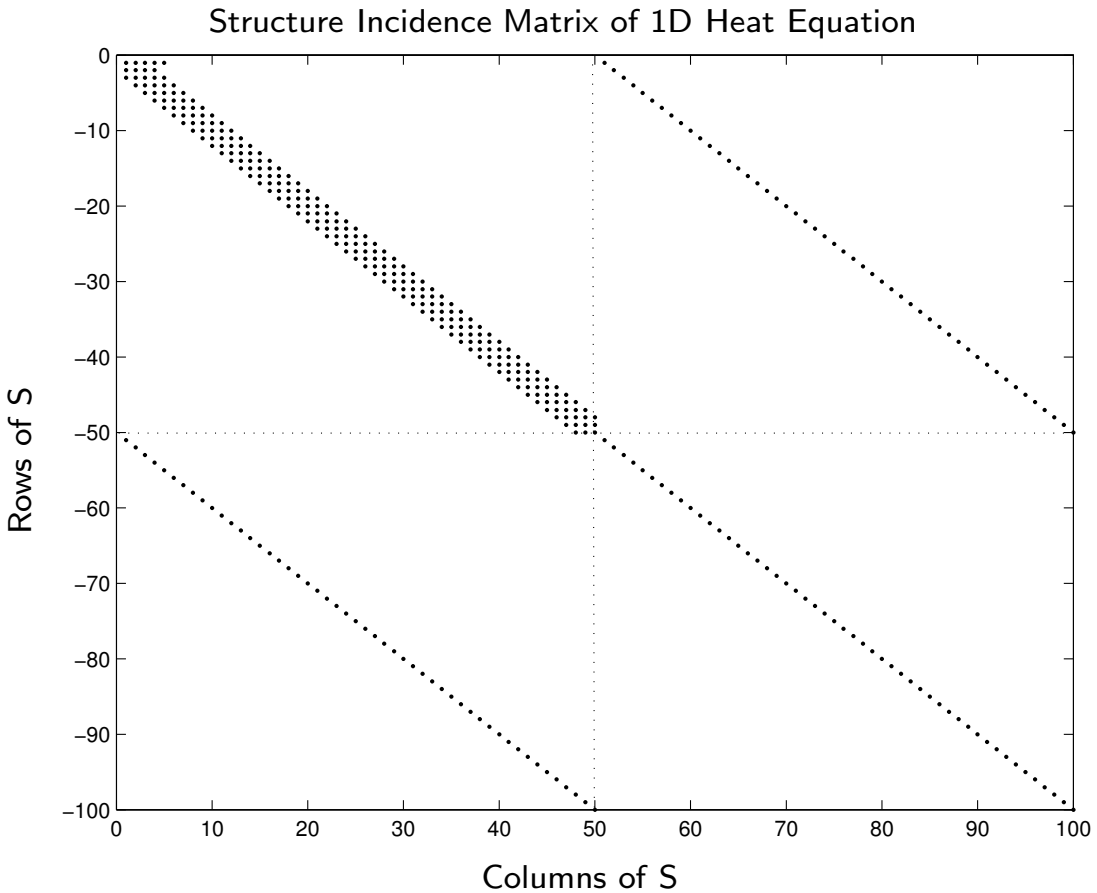
$$\dot{\mathbf{x}} = \mathbf{A} \cdot \mathbf{x} + \mathbf{b} \cdot u$$
$$\mathbf{x} = \mathrm{pre}(\mathbf{x}) + \bar{h} \cdot \dot{\mathbf{x}}$$

where $\mathbf{A}$ is the matrix:

$$\mathbf{A} = \frac{n^2}{120\pi^2} \cdot \begin{pmatrix} -15 & -4 & 14 & -6 & 1 & \dots & 0 & 0 & 0 & 0 \\ 16 & -30 & 16 & -1 & 0 & \dots & 0 & 0 & 0 & 0 \\ -1 & 16 & -30 & 16 & -1 & \dots & 0 & 0 & 0 & 0 \\ 0 & -1 & 16 & -30 & 16 & \dots & 0 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & \dots & 16 & -30 & 16 & -1 \\ 0 & 0 & 0 & 0 & 0 & \dots & -1 & 16 & -31 & 16 \\ 0 & 0 & 0 & 0 & 0 & \dots & 0 & -2 & 32 & -30 \end{pmatrix}$$

Inlining Partial Differential Equations III

The structure incidence matrix is band-structured:



Inlining Partial Differential Equations IV

What is the smallest number of tearing variables that we can get away with?

Inlining Partial Differential Equations IV

What is the smallest number of tearing variables that we can get away with?

- Two trivial tearing structures come to mind immediately. We can either plug the model equations into the solver equations, thereby eliminating the state derivatives:

$$\mathbf{x} = \mathrm{pre}(\mathbf{x}) + \bar{h} \cdot (\mathbf{A} \cdot \mathbf{x} + \mathbf{b} \cdot u)$$

  or alternatively, we can plug the solver equations into the model equations, thereby eliminating the state variables:

$$\dot{\mathbf{x}} = \mathbf{A} \cdot (\mathrm{pre}(\mathbf{x}) + \bar{h} \cdot \dot{\mathbf{x}}) + \mathbf{b} \cdot u$$

  Both solutions half the number of iteration variables to 50.

Inlining Partial Differential Equations IV

What is the smallest number of tearing variables that we can get away with?

- Two trivial tearing structures come to mind immediately. We can either plug the model equations into the solver equations, thereby eliminating the state derivatives:

$$\mathbf{x} = \mathrm{pre}(\mathbf{x}) + \bar{h} \cdot (\mathbf{A} \cdot \mathbf{x} + \mathbf{b} \cdot u)$$

  or alternatively, we can plug the solver equations into the model equations, thereby eliminating the state variables:

$$\dot{\mathbf{x}} = \mathbf{A} \cdot (\mathrm{pre}(\mathbf{x}) + \bar{h} \cdot \dot{\mathbf{x}}) + \mathbf{b} \cdot u$$

  Both solutions half the number of iteration variables to 50.

- Can we do better? Applying the heuristic procedure proposed earlier, we obtain a solution in 32 residual equations and 32 tearing variables, i.e., we are able to reduce the number of tearing variables by a factor of three.

Inlining Partial Differential Equations IV

What is the smallest number of tearing variables that we can get away with?

- Two trivial tearing structures come to mind immediately. We can either plug the model equations into the solver equations, thereby eliminating the state derivatives:

$$\mathbf{x} = \mathrm{pre}(\mathbf{x}) + \bar{h} \cdot (\mathbf{A} \cdot \mathbf{x} + \mathbf{b} \cdot u)$$

  or alternatively, we can plug the solver equations into the model equations, thereby eliminating the state variables:

$$\dot{\mathbf{x}} = \mathbf{A} \cdot (\mathrm{pre}(\mathbf{x}) + \bar{h} \cdot \dot{\mathbf{x}}) + \mathbf{b} \cdot u$$

  Both solutions half the number of iteration variables to 50.

- Can we do better? Applying the heuristic procedure proposed earlier, we obtain a solution in 32 residual equations and 32 tearing variables, i.e., we are able to reduce the number of tearing variables by a factor of three.

- Increasing the look-ahead of the heuristic a bit more, we were able to obtain a solution in 25 residual equations and 25 tearing variables, i.e., we were able to reduce the number of tearing variables by a factor of four.

Inlining Partial Differential Equations V

- We don't know whether this is the optimal solution, but it is the best solution that we were able to find.

Inlining Partial Differential Equations V

- We don't know whether this is the optimal solution, but it is the best solution that we were able to find.
- This is a big improvement. The computational effort of the Gaussian elimination algorithm grows quadratically in the size of the linear equation system. Hence by reducing the size of the Hessian from 50×50 to 25×25, we increase the simulation speed by a full factor of four.

Inlining Partial Differential Equations VI

Let us now try to inline the **Radau IIA(3)** algorithm instead.

Inlining Partial Differential Equations VI

Let us now try to inline the **Radau IIA(3)** algorithm instead.

The inlined difference equation system takes the form:

$$\dot{\mathbf{y}} = \mathbf{A} \cdot \mathbf{y} + \mathbf{b} \cdot u(t_{k+\frac{1}{3}})$$
$$\dot{\mathbf{x}} = \mathbf{A} \cdot \mathbf{x} + \mathbf{b} \cdot u(t_{k+1})$$
$$\mathbf{y} = \mathrm{pre}(\mathbf{x}) + \frac{5}{12} \cdot h \cdot \dot{\mathbf{y}} - \frac{1}{12} \cdot h \cdot \dot{\mathbf{x}}$$
$$\mathbf{x} = \mathrm{pre}(\mathbf{x}) + \frac{3}{4} \cdot h \cdot \dot{\mathbf{y}} + \frac{1}{4} \cdot h \cdot \dot{\mathbf{x}}$$

Inlining Partial Differential Equations VI

Let us now try to inline the **Radau IIA(3)** algorithm instead.

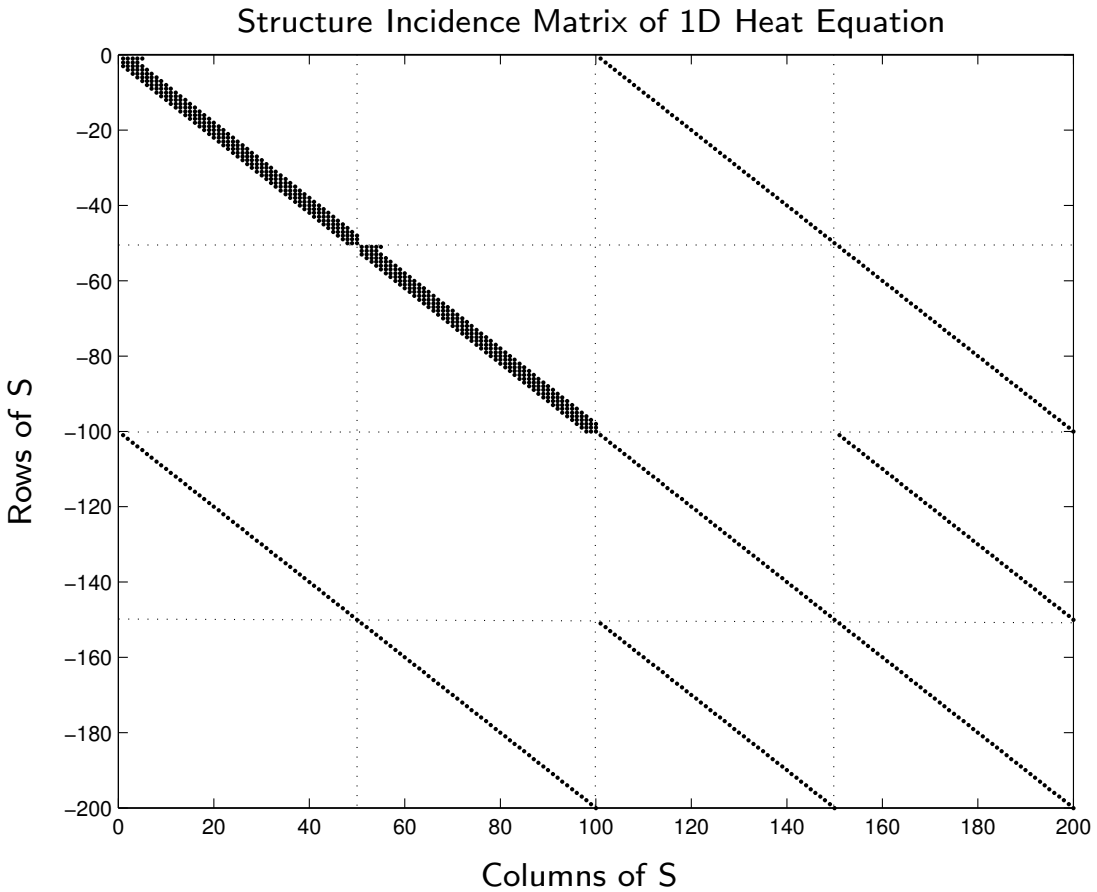
The inlined difference equation system takes the form:

$$\dot{\mathbf{y}} = \mathbf{A} \cdot \mathbf{y} + \mathbf{b} \cdot u(t_{k+\frac{1}{3}})$$
$$\dot{\mathbf{x}} = \mathbf{A} \cdot \mathbf{x} + \mathbf{b} \cdot u(t_{k+1})$$
$$\mathbf{y} = \mathrm{pre}(\mathbf{x}) + \frac{5}{12} \cdot h \cdot \dot{\mathbf{y}} - \frac{1}{12} \cdot h \cdot \dot{\mathbf{x}}$$
$$\mathbf{x} = \mathrm{pre}(\mathbf{x}) + \frac{3}{4} \cdot h \cdot \dot{\mathbf{y}} + \frac{1}{4} \cdot h \cdot \dot{\mathbf{x}}$$

- We paid a hefty price, as we are now dealing with 200 equations in 200 unknowns.

Inlining Partial Differential Equations VII

The structure incidence matrix is still band-structured:



Inlining Partial Differential Equations VIII

We can once again half the number of equations easily.

Inlining Partial Differential Equations VIII

We can once again half the number of equations easily.

We can either plug the model equations into the solver equations, thereby eliminating the state derivatives:

$$\mathbf{y} = \mathrm{pre}(\mathbf{x}) + \frac{5h}{12}\left(\mathbf{A}\cdot\mathbf{y} + \mathbf{b}\cdot u(t_{k+\frac{1}{3}})\right) - \frac{h}{12}\left(\mathbf{A}\cdot\mathbf{x} + \mathbf{b}\cdot u(t_{k+1})\right)$$

$$\mathbf{x} = \mathrm{pre}(\mathbf{x}) + \frac{3h}{4}\left(\mathbf{A}\cdot\mathbf{y} + \mathbf{b}\cdot u(t_{k+\frac{1}{3}})\right) + \frac{h}{4}\left(\mathbf{A}\cdot\mathbf{x} + \mathbf{b}\cdot u(t_{k+1})\right)$$

Inlining Partial Differential Equations VIII

We can once again half the number of equations easily.

We can either plug the model equations into the solver equations, thereby eliminating the state derivatives:

$$\mathbf{y} = \mathrm{pre}(\mathbf{x}) + \frac{5h}{12}\left(\mathbf{A}\cdot\mathbf{y} + \mathbf{b}\cdot u(t_{k+\frac{1}{3}})\right) - \frac{h}{12}\left(\mathbf{A}\cdot\mathbf{x} + \mathbf{b}\cdot u(t_{k+1})\right)$$
$$\mathbf{x} = \mathrm{pre}(\mathbf{x}) + \frac{3h}{4}\left(\mathbf{A}\cdot\mathbf{y} + \mathbf{b}\cdot u(t_{k+\frac{1}{3}})\right) + \frac{h}{4}\left(\mathbf{A}\cdot\mathbf{x} + \mathbf{b}\cdot u(t_{k+1})\right)$$

or alternatively, we can plug the solver equations into the model equations, thereby eliminating the state variables:

$$\dot{\mathbf{y}} = \mathbf{A}\cdot\left(\mathrm{pre}(\mathbf{x}) + \frac{5h}{12}\,\dot{\mathbf{y}} - \frac{h}{12}\,\dot{\mathbf{x}}\right) + \mathbf{b}\cdot u(t_{k+\frac{1}{3}})$$
$$\dot{\mathbf{x}} = \mathbf{A}\cdot\left(\mathrm{pre}(\mathbf{x}) + \frac{3h}{4}\,\dot{\mathbf{y}} + \frac{h}{4}\,\dot{\mathbf{x}}\right) + \mathbf{b}\cdot u(t_{k+1})$$

Inlining Partial Differential Equations VIII

We can once again half the number of equations easily.

We can either plug the model equations into the solver equations, thereby eliminating the state derivatives:

$$\mathbf{y} = \mathrm{pre}(\mathbf{x}) + \frac{5h}{12}\left(\mathbf{A}\cdot\mathbf{y} + \mathbf{b}\cdot u(t_{k+\frac{1}{3}})\right) - \frac{h}{12}\left(\mathbf{A}\cdot\mathbf{x} + \mathbf{b}\cdot u(t_{k+1})\right)$$
$$\mathbf{x} = \mathrm{pre}(\mathbf{x}) + \frac{3h}{4}\left(\mathbf{A}\cdot\mathbf{y} + \mathbf{b}\cdot u(t_{k+\frac{1}{3}})\right) + \frac{h}{4}\left(\mathbf{A}\cdot\mathbf{x} + \mathbf{b}\cdot u(t_{k+1})\right)$$

or alternatively, we can plug the solver equations into the model equations, thereby eliminating the state variables:

$$\dot{\mathbf{y}} = \mathbf{A}\cdot\left(\mathrm{pre}(\mathbf{x}) + \frac{5h}{12}\,\dot{\mathbf{y}} - \frac{h}{12}\,\dot{\mathbf{x}}\right) + \mathbf{b}\cdot u(t_{k+\frac{1}{3}})$$
$$\dot{\mathbf{x}} = \mathbf{A}\cdot\left(\mathrm{pre}(\mathbf{x}) + \frac{3h}{4}\,\dot{\mathbf{y}} + \frac{h}{4}\,\dot{\mathbf{x}}\right) + \mathbf{b}\cdot u(t_{k+1})$$

In either case, we end up with 100 iteration variables.

Inlining Partial Differential Equations IX

Let us see whether our heuristic procedure can do better.

Inlining Partial Differential Equations IX

Let us see whether our heuristic procedure can do better.

- Unfortunately, the algorithm breaks down after having chosen about 60 tearing variables, and after having causalized about 120 equations. The heuristic procedure has maneuvered itself into a corner. Every further selection of a combination of residual equation and tearing variable leads to a structural singularity.

Inlining Partial Differential Equations IX

Let us see whether our heuristic procedure can do better.

- Unfortunately, the algorithm breaks down after having chosen about 60 tearing variables, and after having causalized about 120 equations. The heuristic procedure has maneuvered itself into a corner. Every further selection of a combination of residual equation and tearing variable leads to a structural singularity.
- Although the algorithm had been programmed to ignore selections that would lead to a structural singularity at once, it hadn't been programmed to backtrack beyond the last selection, i.e., throw earlier residual equations and tearing variables away to avoid future mishap.

Inlining Partial Differential Equations IX

Let us see whether our heuristic procedure can do better.

- Unfortunately, the algorithm breaks down after having chosen about 60 tearing variables, and after having causalized about 120 equations. The heuristic procedure has maneuvered itself into a corner. Every further selection of a combination of residual equation and tearing variable leads to a structural singularity.
- Although the algorithm had been programmed to ignore selections that would lead to a structural singularity at once, it hadn't been programmed to backtrack beyond the last selection, i.e., throw earlier residual equations and tearing variables away to avoid future mishap.
- This is why we remarked earlier that the computational complexity of the heuristic procedure grows quadratically with the size of the equation system *for most applications*. It does so, if no backtracking is required.

Inlining Partial Differential Equations X

Can Dymola solve the problem?

Inlining Partial Differential Equations X

Can Dymola solve the problem?

- I quickly programmed the equation system into **Dymola Version 4.1d**. Whereas Dymola usually tears equation systems with tens of thousands of equations within a few seconds, it chewed on this problem for about three hours.

Inlining Partial Differential Equations X

Can Dymola solve the problem?

- I quickly programmed the equation system into **Dymola Version 4.1d**. Whereas Dymola usually tears equation systems with tens of thousands of equations within a few seconds, it chewed on this problem for about three hours.
- It turned out that the heuristic algorithm built into Version 4.1d of Dymola did not break down. Evidently, it had been programmed to backtrack sufficiently to get itself out of the corner. Unfortunately, Dymola came up with one of the two trivial tearing structures.

Inlining Partial Differential Equations X

Can Dymola solve the problem?

- I quickly programmed the equation system into **Dymola Version 4.1d**. Whereas Dymola usually tears equation systems with tens of thousands of equations within a few seconds, it chewed on this problem for about three hours.
- It turned out that the heuristic algorithm built into Version 4.1d of Dymola did not break down. Evidently, it had been programmed to backtrack sufficiently to get itself out of the corner. Unfortunately, Dymola came up with one of the two trivial tearing structures.
- I tried later with **Dymola Version 5.3d**. This time around, Dymola tore this system after only six seconds of compilation time. The answer, however, was still the same. Dymola chose one of the two trivial tearing structures.

Inlining Partial Differential Equations X

Can Dymola solve the problem?

- I quickly programmed the equation system into **Dymola Version 4.1d**. Whereas Dymola usually tears equation systems with tens of thousands of equations within a few seconds, it chewed on this problem for about three hours.
- It turned out that the heuristic algorithm built into Version 4.1d of Dymola did not break down. Evidently, it had been programmed to backtrack sufficiently to get itself out of the corner. Unfortunately, Dymola came up with one of the two trivial tearing structures.
- I tried later with **Dymola Version 5.3d**. This time around, Dymola tore this system after only six seconds of compilation time. The answer, however, was still the same. Dymola chose one of the two trivial tearing structures.
- I had sent this example to Dynasim, when I discovered that tearing took so long. Whenever someone stumbles upon an example that the tearing algorithm does not handle well, the good folks up at Dynasim go into overdrive to improve their tearing heuristics.

Inlining Partial Differential Equations XI

- Unfortunately, these are bad news. If indeed we pay for using Radau IIA instead of BDF3 with increasing the size of the Hessian by a factor of four, Radau IIA would have to be able to use step sizes that are on average at least 16 times larger than those used by BDF for the same accuracy. Otherwise, Radau IIA is not competitive for dealing with this problem. We doubt very much that Radau IIA will be able to do so.

Inlining Partial Differential Equations XI

- Unfortunately, these are bad news. If indeed we pay for using Radau IIA instead of BDF3 with increasing the size of the Hessian by a factor of four, Radau IIA would have to be able to use step sizes that are on average at least 16 times larger than those used by BDF for the same accuracy. Otherwise, Radau IIA is not competitive for dealing with this problem. We doubt very much that Radau IIA will be able to do so.

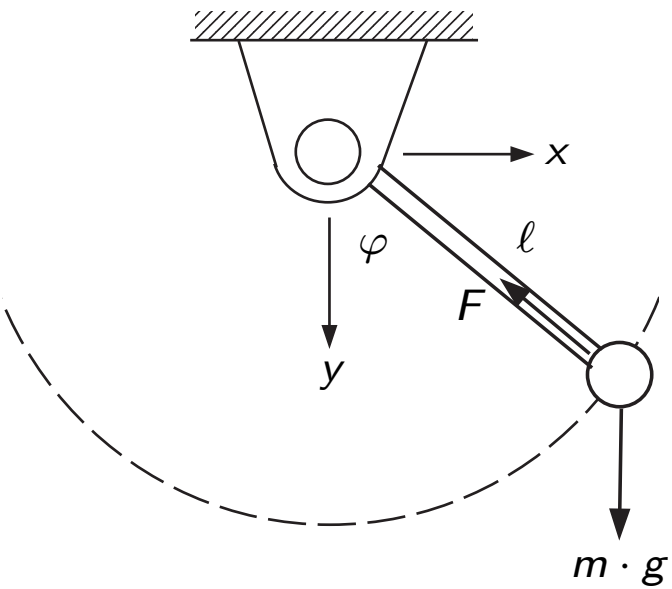
- PDE problems are notoriously difficult simulation problems. Although tearing is a very powerful symbolic sparse matrix technique, it cannot make an intrinsically difficult problem easy to solve.

Over-determined DAEs

Let us return to the pendulum analyzed earlier:

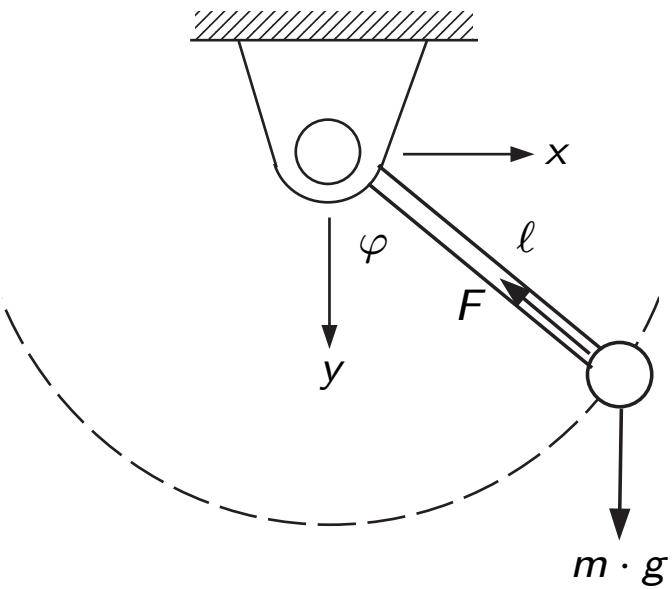
Over-determined DAEs

Let us return to the pendulum analyzed earlier:



Over-determined DAEs

Let us return to the pendulum analyzed earlier:



describable by the following set of *explicit ODEs* (after index reduction):

$$x = \ell \cdot \sin(\varphi)$$
$$v_x = \ell \cdot \cos(\varphi) \cdot \dot{\varphi}$$
$$y = \ell \cdot \cos(\varphi)$$
$$v_y = -\ell \cdot \sin(\varphi) \cdot \dot{\varphi}$$
$$dv_x = -\frac{x \cdot \ell \cdot \dot{\varphi}^2 + x \cdot \cos(\varphi) \cdot g}{x \cdot \sin(\varphi) + y \cdot \cos(\varphi)}$$
$$\ddot{\varphi} = \frac{dv_x}{\ell \cdot \cos(\varphi)} + \frac{\sin(\varphi)}{\cos(\varphi)} \cdot \dot{\varphi}^2$$
$$dv_y = -\ell \cdot \sin(\varphi) \cdot \ddot{\varphi} - \ell \cdot \cos(\varphi) \cdot \dot{\varphi}^2$$
$$F = \frac{m \cdot g \cdot \ell}{y} - \frac{m \cdot \ell \cdot dv_y}{y}$$

Over-determined DAEs II

- We know that the pendulum, as described, is a *conservative (Hamiltonian) system*, since no friction was assumed anywhere. Hence the pendulum, once disturbed, should swing forever with the same frequency and amplitude.

Over-determined DAEs II

- We know that the pendulum, as described, is a *conservative (Hamiltonian) system*, since no friction was assumed anywhere. Hence the pendulum, once disturbed, should swing forever with the same frequency and amplitude.

- The *total free energy*, E_f:

$$E_f = E_p + E_k$$

  which is the sum of the potential energy, E_p, and the kinetic energy, E_k, should be constant.

Over-determined DAEs II

- We know that the pendulum, as described, is a *conservative (Hamiltonian) system*, since no friction was assumed anywhere. Hence the pendulum, once disturbed, should swing forever with the same frequency and amplitude.

- The *total free energy*, E_f:

$$E_f = E_p + E_k$$

  which is the sum of the potential energy, E_p, and the kinetic energy, E_k, should be constant.

- The *potential energy* can be modeled as:

$$E_p = m \cdot g \cdot (y_0 - y)$$

  and the *kinetic energy* can be expressed using the formula:

$$E_k = \frac{1}{2} \cdot m \cdot v_x^2 + \frac{1}{2} \cdot m \cdot v_y^2$$

Over-determined DAEs II

- We know that the pendulum, as described, is a *conservative (Hamiltonian) system*, since no friction was assumed anywhere. Hence the pendulum, once disturbed, should swing forever with the same frequency and amplitude.

- The *total free energy*, E_f:

$$E_f = E_p + E_k$$

  which is the sum of the potential energy, E_p, and the kinetic energy, E_k, should be constant.

- The *potential energy* can be modeled as:

$$E_p = m \cdot g \cdot (y_0 - y)$$

  and the *kinetic energy* can be expressed using the formula:

$$E_k = \frac{1}{2} \cdot m \cdot v_x^2 + \frac{1}{2} \cdot m \cdot v_y^2$$

- Let us add these three equations to the model.

Over-determined DAEs III

We shall *inline the FE algorithm*:

$$\dot{\varphi}_{k+1} = \dot{\varphi}_k + h \cdot \ddot{\varphi}_k$$
$$\varphi_{k+1} = \varphi_k + h \cdot \dot{\varphi}_k$$

and simulate the problem across $10\ sec$ of simulated time with a fixed step size of $h = 0.01\ sec$ by simply iterating over the 13 equations in 13 unknowns.

We use: $g = 9.81\ m/(sec^2)$, $m = 10\ kg$, $\ell = 1\ m$, $\varphi_0 = +45^{\circ} = \pi/4\ rad$, and $\dot{\varphi}_0 = 0\ rad/sec$. Thus, $y_0 = \sqrt{2}/2\ m$.

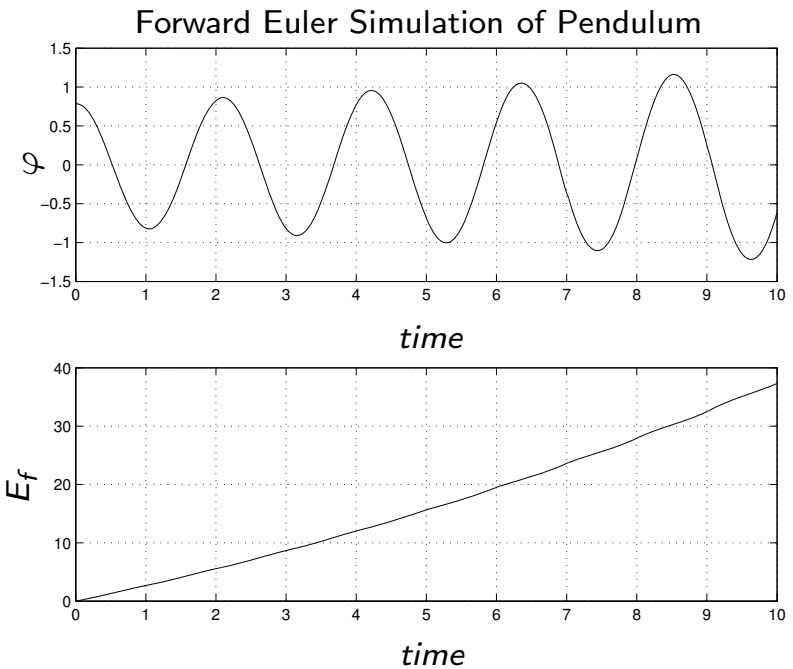
Over-determined DAEs III

We shall *inline the FE algorithm*:

$$\dot{\varphi}_{k+1} = \dot{\varphi}_k + h \cdot \ddot{\varphi}_k$$
$$\varphi_{k+1} = \varphi_k + h \cdot \dot{\varphi}_k$$

and simulate the problem across 10 *sec* of simulated time with a fixed step size of $h = 0.01$ *sec* by simply iterating over the 13 equations in 13 unknowns.

We use: $g = 9.81\ m/(sec^2)$, $m = 10\ kg$, $\ell = 1\ m$, $\varphi_0 = +45^o = \pi/4\ rad$, and $\dot{\varphi}_0 = 0\ rad/sec$. Thus, $y_0 = \sqrt{2}/2\ m$.



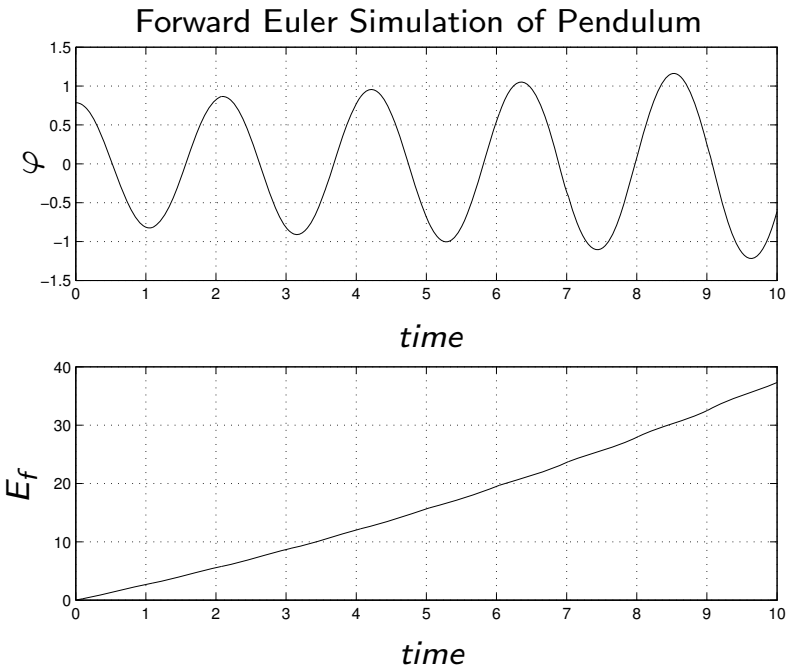
Over-determined DAEs III

We shall *inline the FE algorithm*:

$$\dot{\varphi}_{k+1} = \dot{\varphi}_k + h \cdot \ddot{\varphi}_k$$
$$\varphi_{k+1} = \varphi_k + h \cdot \dot{\varphi}_k$$

and simulate the problem across 10 *sec* of simulated time with a fixed step size of $h = 0.01$ *sec* by simply iterating over the 13 equations in 13 unknowns.

We use: $g = 9.81\ m/(sec^2)$, $m = 10\ kg$, $\ell = 1\ m$, $\varphi_0 = +45^o = \pi/4\ rad$, and $\dot{\varphi}_0 = 0\ rad/sec$. Thus, $y_0 = \sqrt{2}/2\ m$.



- **We just solved the world's energy depletion problem once and for all as we seem to be able to generate free energy out of thin air.**

Over-determined DAEs IV

Let us *inline the BE algorithm* instead:

$$\dot{\varphi} = \mathrm{pre}(\dot{\varphi}) + h \cdot \ddot{\varphi}$$
$$\varphi = \mathrm{pre}(\varphi) + h \cdot \dot{\varphi}$$

Over-determined DAEs IV

Let us *inline the BE algorithm* instead:

$$\dot{\varphi} = \mathrm{pre}(\dot{\varphi}) + h \cdot \ddot{\varphi}$$
$$\varphi = \mathrm{pre}(\varphi) + h \cdot \dot{\varphi}$$

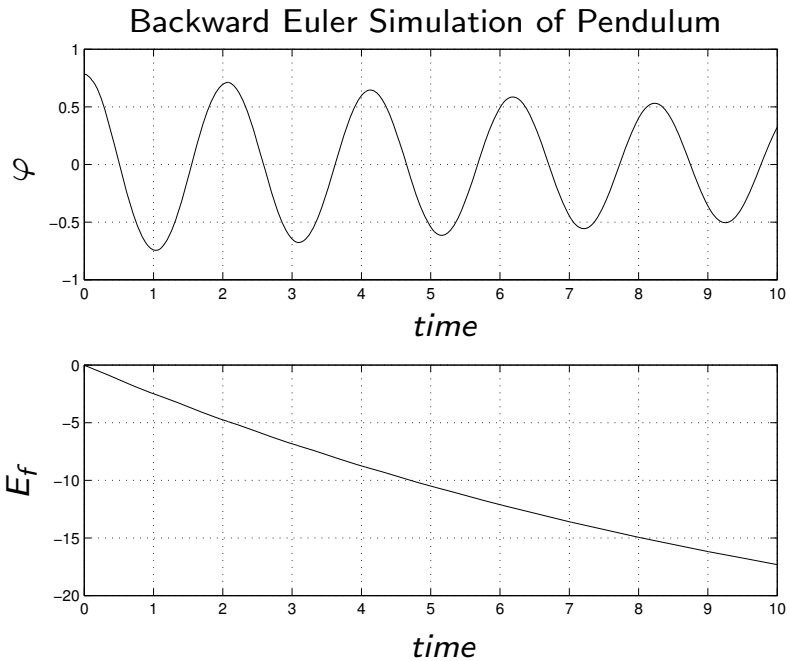
Since BE is an implicit algorithm, we encounter another algebraic loop in six equations and one tearing variable, $\ddot{\varphi}$. We solve that algebraic equation by Newton iteration and simulate across 10 *sec* of simulated time with a fixed step size of $h = 0.01$ *sec*.

Over-determined DAEs IV

Let us *inline the BE algorithm* instead:

$$\dot{\varphi} = \mathrm{pre}(\dot{\varphi}) + h \cdot \ddot{\varphi}$$
$$\varphi = \mathrm{pre}(\varphi) + h \cdot \dot{\varphi}$$

Since BE is an implicit algorithm, we encounter another algebraic loop in six equations and one tearing variable, $\ddot{\varphi}$. We solve that algebraic equation by Newton iteration and simulate across 10 *sec* of simulated time with a fixed step size of $h = 0.01$ *sec*.

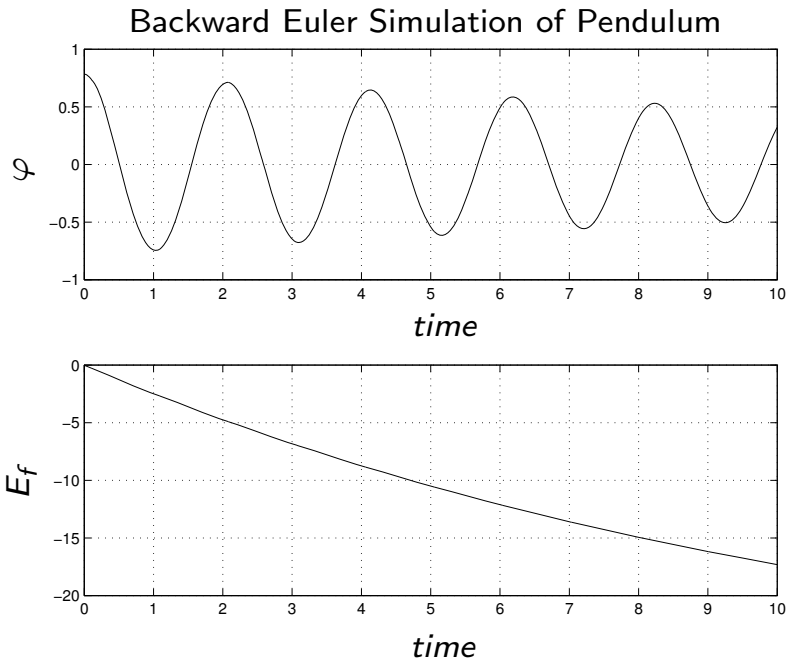


Over-determined DAEs IV

Let us *inline the BE algorithm* instead:

$$\dot{\varphi} = \mathrm{pre}(\dot{\varphi}) + h \cdot \ddot{\varphi}$$
$$\varphi = \mathrm{pre}(\varphi) + h \cdot \dot{\varphi}$$

Since BE is an implicit algorithm, we encounter another algebraic loop in six equations and one tearing variable, $\ddot{\varphi}$. We solve that algebraic equation by Newton iteration and simulate across 10 *sec* of simulated time with a fixed step size of $h = 0.01$ *sec*.



- **This time around, the numerical simulation dissipates energy.**

Over-determined DAEs V

I tried once more with *inlining the BI2 algorithm*.

Over-determined DAEs V

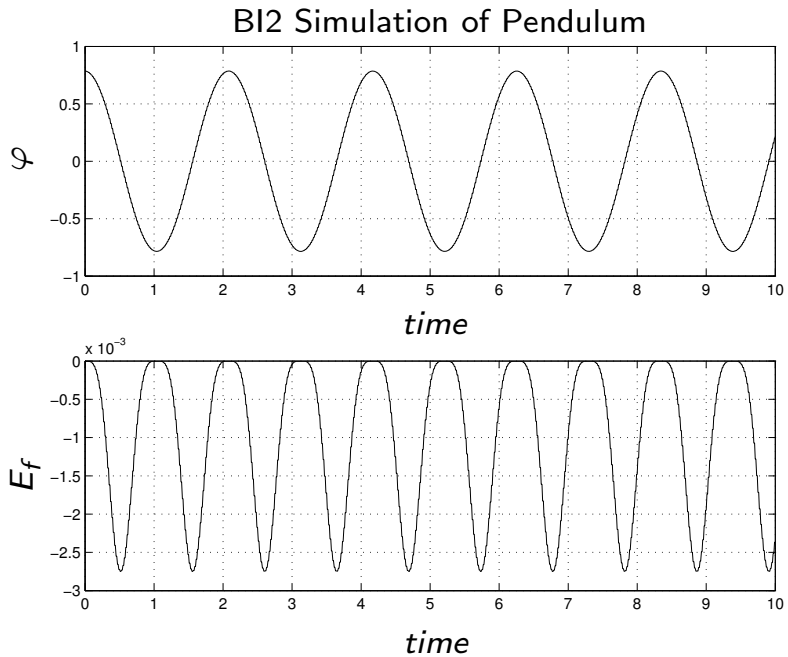
I tried once more with *inlining the BI2 algorithm*.

For simplicity, I implemented the trapezoidal rule as a cyclic method, toggling between one step of FE and one step of BE.

Over-determined DAEs V

I tried once more with *inlining the BI2 algorithm*.

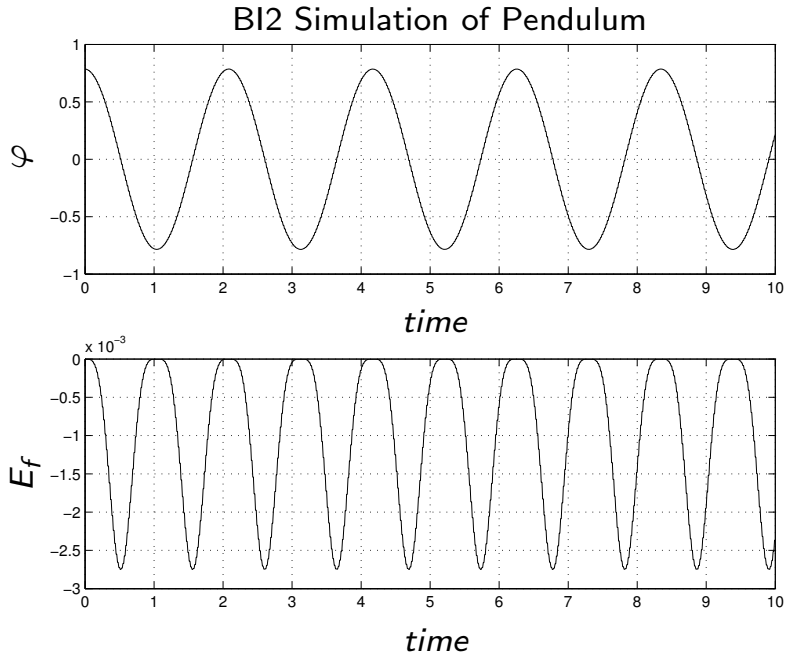
For simplicity, I implemented the trapezoidal rule as a cyclic method, toggling between one step of FE and one step of BE.



Over-determined DAEs V

I tried once more with *inlining the BI2 algorithm*.

For simplicity, I implemented the trapezoidal rule as a cyclic method, toggling between one step of FE and one step of BE.



- **This time, it finally worked! The free energy doesn't stay exactly constant, but it varies in the *mW* range only (for numerical reasons), and it doesn't either systematically increase or decrease over time.**

Over-determined DAEs VI

- The results are not exactly surprising. This is a *conservative system*. If it were linear, it would have its eigenvalues on the imaginary axis. Since it is non-linear, the eigenvalues of its Jacobian may wobble back and force between the left-half and the right-half plane, but will stay in the vicinity of the imaginary axis at all times.

Over-determined DAEs VI

- The results are not exactly surprising. This is a *conservative system*. If it were linear, it would have its eigenvalues on the imaginary axis. Since it is non-linear, the eigenvalues of its Jacobian may wobble back and force between the left-half and the right-half plane, but will stay in the vicinity of the imaginary axis at all times.

- Since **FE** has its stability domain loop in the left-half plane, it sees the imaginary axis as unstable, and consequently, the oscillation will grow.

Over-determined DAEs VI

- The results are not exactly surprising. This is a *conservative system*. If it were linear, it would have its eigenvalues on the imaginary axis. Since it is non-linear, the eigenvalues of its Jacobian may wobble back and force between the left-half and the right-half plane, but will stay in the vicinity of the imaginary axis at all times.
- Since **FE** has its stability domain loop in the left-half plane, it sees the imaginary axis as unstable, and consequently, the oscillation will grow.
- Since **BE** has its stability domain loop in the right-half plane, it sees the imaginary axis as stable, and consequently, the oscillation will decay.

Over-determined DAEs VI

- The results are not exactly surprising. This is a *conservative system*. If it were linear, it would have its eigenvalues on the imaginary axis. Since it is non-linear, the eigenvalues of its Jacobian may wobble back and force between the left-half and the right-half plane, but will stay in the vicinity of the imaginary axis at all times.
- Since **FE** has its stability domain loop in the left-half plane, it sees the imaginary axis as unstable, and consequently, the oscillation will grow.
- Since **BE** has its stability domain loop in the right-half plane, it sees the imaginary axis as stable, and consequently, the oscillation will decay.
- Since **BI2** is an F-stable algorithm, it sees the imaginary axis as borderline stable, and consequently, the oscillation will neither grow nor decay.

Over-determined DAEs VI

- The results are not exactly surprising. This is a *conservative system*. If it were linear, it would have its eigenvalues on the imaginary axis. Since it is non-linear, the eigenvalues of its Jacobian may wobble back and force between the left-half and the right-half plane, but will stay in the vicinity of the imaginary axis at all times.
- Since **FE** has its stability domain loop in the left-half plane, it sees the imaginary axis as unstable, and consequently, the oscillation will grow.
- Since **BE** has its stability domain loop in the right-half plane, it sees the imaginary axis as stable, and consequently, the oscillation will decay.
- Since **BI2** is an F-stable algorithm, it sees the imaginary axis as borderline stable, and consequently, the oscillation will neither grow nor decay.

We were finally able to get a "stable" oscillation, but only, because we analyzed the problem and came up with a suitable solution. The BI2 code itself still has no inkling that it is supposed to conserve free energy. It does so by accident rather than by design.

Over-determined DAEs VII

Let us try to change that. We shall force the BE algorithm to *preserve the free energy*.

Over-determined DAEs VII

Let us try to change that. We shall force the BE algorithm to *preserve the free energy*.

To this end, we simply add the equation:

$$E_f = 0$$

to the set of equations.

Over-determined DAEs VII

Let us try to change that. We shall force the BE algorithm to *preserve the free energy*.

To this end, we simply add the equation:

$$E_f = 0$$

to the set of equations.

This is a completely new situation. We haven't added any new variables to the set of equations. We only added another equation. Thus, we now have 14 equations in 13 unknowns. Clearly, this *problem is constrained*.

Over-determined DAEs VII

Let us try to change that. We shall force the BE algorithm to *preserve the free energy*.

To this end, we simply add the equation:

$$E_f = 0$$

to the set of equations.

This is a completely new situation. We haven't added any new variables to the set of equations. We only added another equation. Thus, we now have 14 equations in 13 unknowns. Clearly, this *problem is constrained*.

If we present this problem to the *Pantelides algorithm*, it will differentiate itself to death, or rather, until the compiler runs out of virtual memory. The Pantelides algorithm always adds exactly as many equations as variables, thus after each application of the algorithm, the number of equations is still one larger than the number of variables.

Over-determined DAEs VIII

Inlining again saves our neck.

Over-determined DAEs VIII

Inlining again saves our neck.

We simply add the constraint equation to the iteration equations of the Newton iteration. Thus, the set of zero functions can now be written as:

$$\mathcal{F} = \begin{pmatrix} \ddot{\varphi}_{new} - \ddot{\varphi} \\ E_f \end{pmatrix}$$

and therefore:

$$\mathcal{H} = \frac{\partial \mathcal{F}}{\partial \ddot{\varphi}} = \begin{pmatrix} \partial \ddot{\varphi}_{new} / \partial \ddot{\varphi} - 1 \\ \partial E_f / \partial \ddot{\varphi} \end{pmatrix}$$

Over-determined DAEs VIII

Inlining again saves our neck.

We simply add the constraint equation to the iteration equations of the Newton iteration. Thus, the set of zero functions can now be written as:

$$\mathcal{F} = \begin{pmatrix} \ddot{\varphi}_{new} - \ddot{\varphi} \\ E_f \end{pmatrix}$$

and therefore:

$$\mathcal{H} = \frac{\partial \mathcal{F}}{\partial \ddot{\varphi}} = \begin{pmatrix} \partial \ddot{\varphi}_{new} / \partial \ddot{\varphi} - 1 \\ \partial E_f / \partial \ddot{\varphi} \end{pmatrix}$$

The Newton iteration can be written as:

$$\mathcal{H}^{\ell} \cdot \mathbf{dx}^{\ell} = \mathcal{F}^{\ell}$$
$$\mathbf{x}^{\ell+1} = \mathbf{x}^{\ell} - \mathbf{dx}^{\ell}$$

Over-determined DAEs IX

However, $\mathcal{H}$ is no longer a square matrix. It is now a rectangular matrix with 2 rows and 1 column.

Over-determined DAEs IX

However, $\mathcal{H}$ is no longer a square matrix. It is now a rectangular matrix with 2 rows and 1 column.

In general with n tearing variables and p additional constraints, the Hessian turns out to be a rectangular matrix with $n + p$ rows and n columns. Thus, the Newton iteration is *over-determined*. It cannot be satisfied exactly. The **dx**-vector can only be *determined in a least square sense*.

Over-determined DAEs IX

However, $\mathcal{H}$ is no longer a square matrix. It is now a rectangular matrix with 2 rows and 1 column.

In general with n tearing variables and p additional constraints, the Hessian turns out to be a rectangular matrix with $n + p$ rows and n columns. Thus, the Newton iteration is *over-determined*. It cannot be satisfied exactly. The **dx**-vector can only be *determined in a least square sense*.

This can be accomplished by multiplying the linear equation system from the left with $\mathcal{H}^*$, i.e., with the Hermitian transpose of $\mathcal{H}$. If the rank of $\mathcal{H}$ is n, then $\mathcal{H}^* \cdot \mathcal{H}$ is a Hermitian matrix of full rank.

Over-determined DAEs IX

However, $\mathcal{H}$ is no longer a square matrix. It is now a rectangular matrix with 2 rows and 1 column.

In general with n tearing variables and p additional constraints, the Hessian turns out to be a rectangular matrix with $n + p$ rows and n columns. Thus, the Newton iteration is *over-determined*. It cannot be satisfied exactly. The **dx**-vector can only be *determined in a least square sense*.

This can be accomplished by multiplying the linear equation system from the left with $\mathcal{H}^*$, i.e., with the Hermitian transpose of $\mathcal{H}$. If the rank of $\mathcal{H}$ is n, then $\mathcal{H}^* \cdot \mathcal{H}$ is a Hermitian matrix of full rank.

Thus, we can compute **dx** as:

$$\mathbf{dx} = (\mathcal{H}^* \cdot \mathcal{H})^{-1} \cdot \mathcal{H}^* \cdot \mathcal{F}$$

where $(\mathcal{H}^* \cdot \mathcal{H})^{-1} \cdot \mathcal{H}^*$ is the *Penrose-Moore pseudo-inverse* of $\mathcal{H}$.

Over-determined DAEs IX

However, $\mathcal{H}$ is no longer a square matrix. It is now a rectangular matrix with 2 rows and 1 column.

In general with n tearing variables and p additional constraints, the Hessian turns out to be a rectangular matrix with $n + p$ rows and n columns. Thus, the Newton iteration is *over-determined*. It cannot be satisfied exactly. The **dx**-vector can only be *determined in a least square sense*.

This can be accomplished by multiplying the linear equation system from the left with $\mathcal{H}^*$, i.e., with the Hermitian transpose of $\mathcal{H}$. If the rank of $\mathcal{H}$ is n, then $\mathcal{H}^* \cdot \mathcal{H}$ is a Hermitian matrix of full rank.

Thus, we can compute **dx** as:

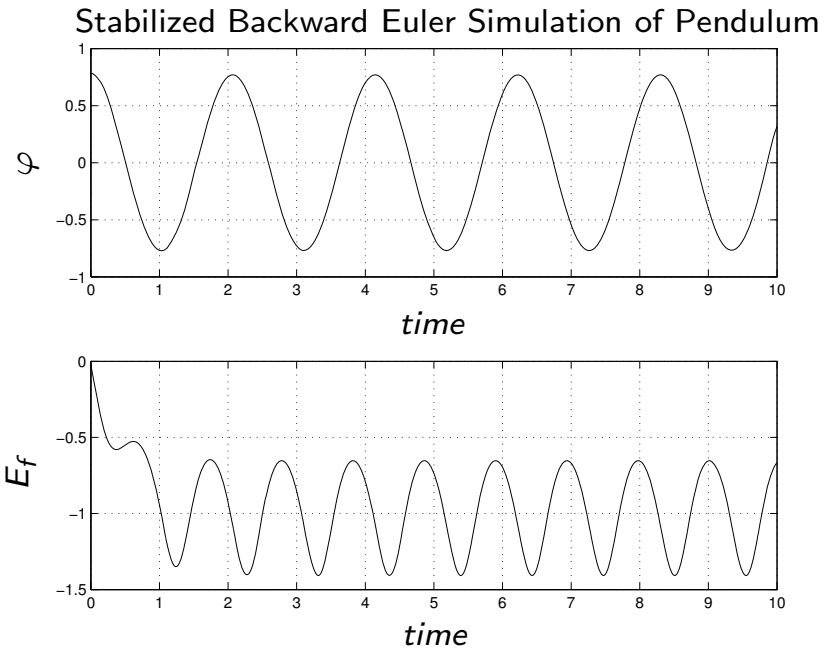
$$\mathbf{dx} = (\mathcal{H}^* \cdot \mathcal{H})^{-1} \cdot \mathcal{H}^* \cdot \mathcal{F}$$

where $(\mathcal{H}^* \cdot \mathcal{H})^{-1} \cdot \mathcal{H}^*$ is the *Penrose-Moore pseudo-inverse* of $\mathcal{H}$.

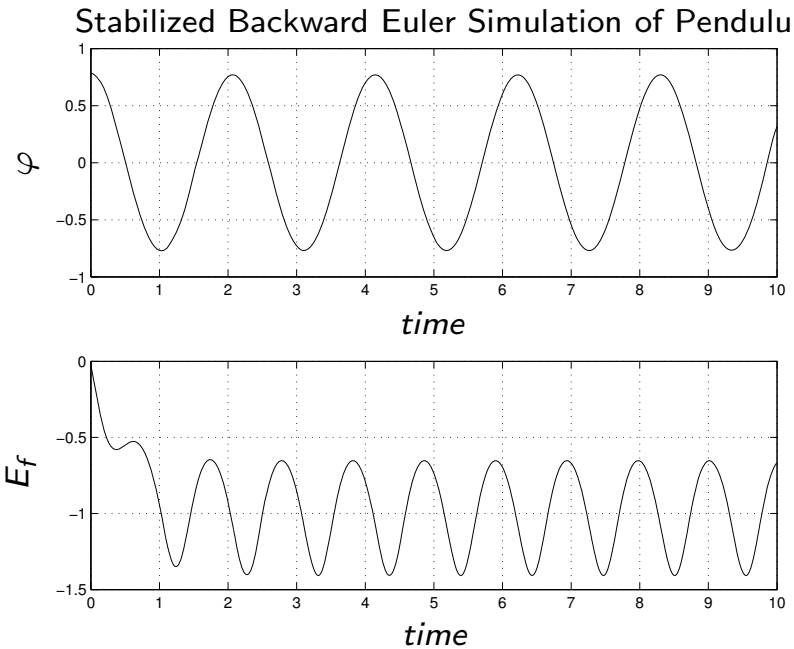
In **MATLAB**, this can be abbreviated as:

$$\mathbf{dx} = \mathcal{H} \backslash \mathcal{F}$$

Over-determined DAEs X

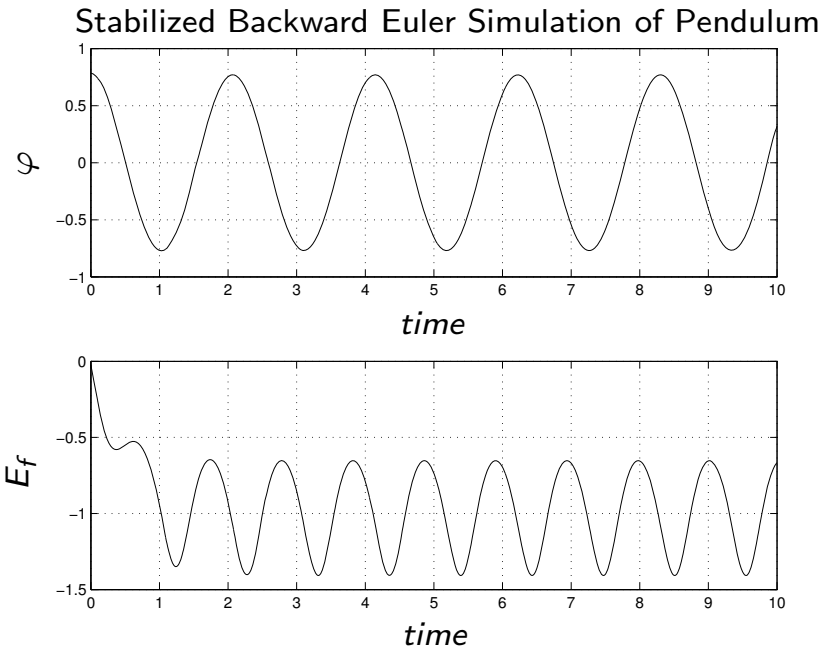


Over-determined DAEs X



- The oscillation has indeed been stabilized.

Over-determined DAEs X

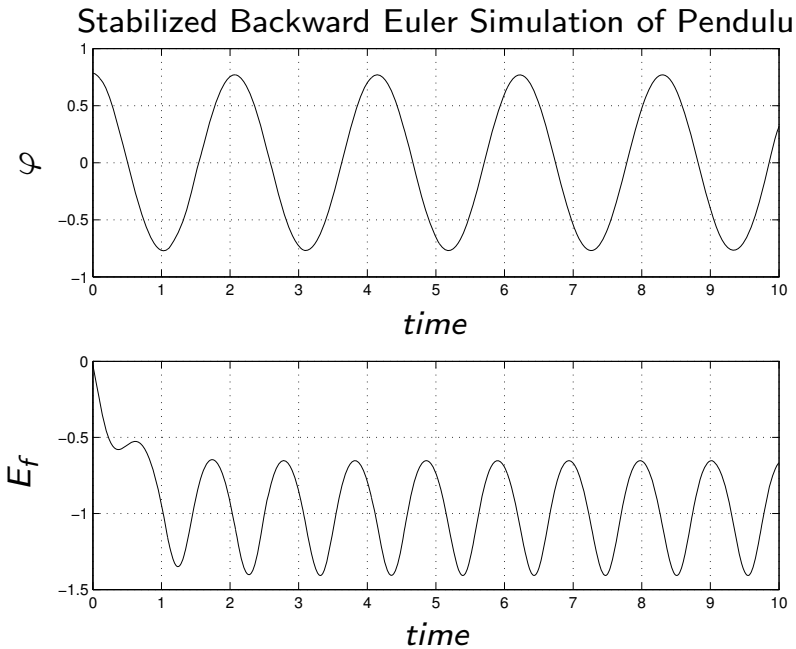


- The oscillation has indeed been stabilized.
- Of course, the equation:

$$\mathcal{F} = 0$$

can no longer be solved precisely. The equation system does not contain enough freedom to do so.

Over-determined DAEs X



- The oscillation has indeed been stabilized.
- Of course, the equation:

$$\mathcal{F} = 0$$

can no longer be solved precisely. The equation system does not contain enough freedom to do so.

- Yet, the error is minimized in a least square sense, and both the oscillation and the free energy are now stable by design.

Over-determined DAEs XI

- Initially, the approach still loses a bit of free energy, but the loss stops after the solution is stabilized.

Over-determined DAEs XI

- Initially, the approach still loses a bit of free energy, but the loss stops after the solution is stabilized.
- The solution using back-interpolation turns out to be better, but the solution using an over-determined equation set is more robust.

Over-determined DAEs XI

- Initially, the approach still loses a bit of free energy, but the loss stops after the solution is stabilized.
- The solution using back-interpolation turns out to be better, but the solution using an over-determined equation set is more robust.
- There are DAE solvers on the market that can handle over-determined DAEs, such as **ODASSLRT** (a "dialect" of DASSL) and **MEXX** (a code based on Richardson extrapolation).

Over-determined DAEs XI

- Initially, the approach still loses a bit of free energy, but the loss stops after the solution is stabilized.
- The solution using back-interpolation turns out to be better, but the solution using an over-determined equation set is more robust.
- There are DAE solvers on the market that can handle over-determined DAEs, such as **ODASSLRT** (a "dialect" of DASSL) and **MEXX** (a code based on Richardson extrapolation).
- Over-determined DAE solvers have become popular primarily among specialists of *multi-body dynamics*, and the early codes tackling this problem indeed evolved in the engineering community. Most of these early codes were quite specialized.

Over-determined DAEs XII

- More recently, the problem was discovered by mainstream applied mathematicians.

Over-determined DAEs XII

- More recently, the problem was discovered by mainstream applied mathematicians.
- Ernst Hairer, rather than constraining the problem, generalized on the BI2 solution presented earlier.

Over-determined DAEs XII

- More recently, the problem was discovered by mainstream applied mathematicians.
- Ernst Hairer, rather than constraining the problem, generalized on the BI2 solution presented earlier.
- He discovered that, in order for a DAE solver to tackle such a problem successfully, *the solver must be symmetric.*

Over-determined DAEs XII

- More recently, the problem was discovered by mainstream applied mathematicians.
- Ernst Hairer, rather than constraining the problem, generalized on the BI2 solution presented earlier.
- He discovered that, in order for a DAE solver to tackle such a problem successfully, *the solver must be symmetric.*
- Some algorithms do not change, when h is replaced by $-h$. For example, the trapezoidal rule:

$$\mathbf{x_{k+1}} = \mathbf{x_k} + \frac{h}{2} \cdot (\dot{\mathbf{x}}_\mathbf{k} + \dot{\mathbf{x}}_{\mathbf{k+1}})$$

turns into:

$$\mathbf{x_k} = \mathbf{x_{k+1}} - \frac{h}{2} \cdot (\dot{\mathbf{x}}_{\mathbf{k+1}} + \dot{\mathbf{x}}_\mathbf{k})$$

i.e., the formula doesn't change.

Over-determined DAEs XII

- More recently, the problem was discovered by mainstream applied mathematicians.
- Ernst Hairer, rather than constraining the problem, generalized on the BI2 solution presented earlier.
- He discovered that, in order for a DAE solver to tackle such a problem successfully, *the solver must be symmetric.*
- Some algorithms do not change, when h is replaced by $-h$. For example, the trapezoidal rule:

$$\mathbf{x_{k+1}} = \mathbf{x_k} + \frac{h}{2} \cdot (\dot{\mathbf{x}}_\mathbf{k} + \dot{\mathbf{x}}_{\mathbf{k+1}})$$

  turns into:

$$\mathbf{x_k} = \mathbf{x_{k+1}} - \frac{h}{2} \cdot (\dot{\mathbf{x}}_{\mathbf{k+1}} + \dot{\mathbf{x}}_\mathbf{k})$$

  i.e., the formula doesn't change.
- Such an ODE solver is called a *symmetric integration algorithm*.

Over-determined DAEs XIII

- The stability domains of symmetric ODE solvers are evidently symmetric to the imaginary axis.

Over-determined DAEs XIII

- The stability domains of symmetric ODE solvers are evidently symmetric to the imaginary axis.
- All of the F-stable algorithms introduced earlier are symmetric.

Over-determined DAEs XIII

- The stability domains of symmetric ODE solvers are evidently symmetric to the imaginary axis.
- All of the F-stable algorithms introduced earlier are symmetric.
- Symmetric integration algorithms are not only symmetric to the imaginary axis w.r.t. their stability properties, but also w.r.t. their damping properties. Thus, symmetric integration algorithms are accompanied by symmetric order stars as well.

Over-determined DAEs XIII

- The stability domains of symmetric ODE solvers are evidently symmetric to the imaginary axis.
- All of the F-stable algorithms introduced earlier are symmetric.
- Symmetric integration algorithms are not only symmetric to the imaginary axis w.r.t. their stability properties, but also w.r.t. their damping properties. Thus, symmetric integration algorithms are accompanied by symmetric order stars as well.
- This symmetry can be exploited in the simulation of Hamiltonian systems. At least, if we carefully choose our step size to be in sync with the eigenfrequency of oscillation of the system, we can ensure that the damping errors committed during the integration over a full period cancel out, such that the solution at the end of one cycle coincides with that at the beginning of the cycle.

Over-determined DAEs XIII

- The stability domains of symmetric ODE solvers are evidently symmetric to the imaginary axis.
- All of the F-stable algorithms introduced earlier are symmetric.
- Symmetric integration algorithms are not only symmetric to the imaginary axis w.r.t. their stability properties, but also w.r.t. their damping properties. Thus, symmetric integration algorithms are accompanied by symmetric order stars as well.
- This symmetry can be exploited in the simulation of Hamiltonian systems. At least, if we carefully choose our step size to be in sync with the eigenfrequency of oscillation of the system, we can ensure that the damping errors committed during the integration over a full period cancel out, such that the solution at the end of one cycle coincides with that at the beginning of the cycle.
- Yet, we still prefer the constrained solution proposed in this section, as it is considerably more robust. It works with any numerical integration scheme and enforces the physical constraint directly rather than indirectly.

Conclusions

- In this third and last presentation on DAE solvers, we looked at a variety of issues not previously discussed.

Conclusions

- In this third and last presentation on DAE solvers, we looked at a variety of issues not previously discussed.
- We started out with discussing how the inlining concept can be applied to implicit Runge-Kutta (IRK) algorithms.

Conclusions

- In this third and last presentation on DAE solvers, we looked at a variety of issues not previously discussed.
- We started out with discussing how the inlining concept can be applied to implicit Runge-Kutta (IRK) algorithms.
- We then discussed stiffly-stable step-size control algorithms for IRK methods. To this end, we found stiffly-stable embedding algorithms spanning over two steps of the original IRK algorithm.

Conclusions

- In this third and last presentation on DAE solvers, we looked at a variety of issues not previously discussed.
- We started out with discussing how the inlining concept can be applied to implicit Runge-Kutta (IRK) algorithms.
- We then discussed stiffly-stable step-size control algorithms for IRK methods. To this end, we found stiffly-stable embedding algorithms spanning over two steps of the original IRK algorithm.
- We then returned to the issue of simulating parabolic PDEs and discussed, how inlining can help us with this endeavor.

Conclusions

- In this third and last presentation on DAE solvers, we looked at a variety of issues not previously discussed.
- We started out with discussing how the inlining concept can be applied to implicit Runge-Kutta (IRK) algorithms.
- We then discussed stiffly-stable step-size control algorithms for IRK methods. To this end, we found stiffly-stable embedding algorithms spanning over two steps of the original IRK algorithm.
- We then returned to the issue of simulating parabolic PDEs and discussed, how inlining can help us with this endeavor.
- The presentation ended with the discussion of over-determined DAEs and how they can be simulated using a constrained inlining approach.

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. Cellier, F.E. (2000), "Inlining Step-size Controlled Fully Implicit Runge-Kutta Algorithms for the Semi-analytical and Semi-numerical Solution of Stiff ODEs and DAEs," *Proc. V^{th} Conference on Computer Simulation*, Mexico City, Mexico, pp.259-262.

3. 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.