Numerical Simulation of Dynamic Systems XXIV

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

May 14, 2013

Introduction

A number of important simulation applications must run under real-time constraints.

Introduction

A number of important simulation applications must run under real-time constraints.

- A *training simulator* (flight simulator, driving simulator, plant operation simulator) must run in real time to offer meaningful training to the user.

Introduction

A number of important simulation applications must run under real-time constraints.

- A *training simulator* (flight simulator, driving simulator, plant operation simulator) must run in real time to offer meaningful training to the user.

- A *model reference adaptive controller (MRAC)* makes use of a plant model for its control decisions. Evidently, the plant model must then be simulated under real-time constraints so that the control action can be taken in a timely manner.

Introduction

A number of important simulation applications must run under real-time constraints.

- A *training simulator* (flight simulator, driving simulator, plant operation simulator) must run in real time to offer meaningful training to the user.
- A *model reference adaptive controller (MRAC)* makes use of a plant model for its control decisions. Evidently, the plant model must then be simulated under real-time constraints so that the control action can be taken in a timely manner.
- A *watchdog monitor* makes use of a plant model, simulated under real-time constraints, to compare the simulation trajectories with real measurements to identify anomalous behavior of the real plant when it occurs, i.e., it is being used for *fault detection*, *fault isolation*, and *fault identification*.

Introduction II

What is special about running a simulation under real-time constraints?

Introduction II

What is special about running a simulation under real-time constraints?

Conceptually, the implementation of real-time simulation software is straightforward. It contains only four new components:

Introduction II

What is special about running a simulation under real-time constraints?

Conceptually, the implementation of real-time simulation software is straightforward. It contains only four new components:

- The *real-time clock* is responsible for the synchronization of real time and simulated time. The real-time clock is programmed to send a trigger impulse once every h time units of real time, where h is the current step size of the integration algorithm, and the simulation program is equipped with a *busy waiting* mechanism that is launched as soon as all computations associated with the current step have been completed, and that checks for arrival of the next trigger signal. The new step will not begin until the trigger signal has been received.

Introduction II

What is special about running a simulation under real-time constraints?

Conceptually, the implementation of real-time simulation software is straightforward. It contains only four new components:

- The *real-time clock* is responsible for the synchronization of real time and simulated time. The real-time clock is programmed to send a trigger impulse once every h time units of real time, where h is the current step size of the integration algorithm, and the simulation program is equipped with a *busy waiting* mechanism that is launched as soon as all computations associated with the current step have been completed, and that checks for arrival of the next trigger signal. The new step will not begin until the trigger signal has been received.

- The *analog to digital (A/D) converters* are read at the beginning of each integration step to update the values of all external driving functions. This corresponds effectively to a *sample and hold (S/H)* mechanism. The inputs are updated once at the beginning of every integration step and are then kept constant during the entire step.

Introduction III

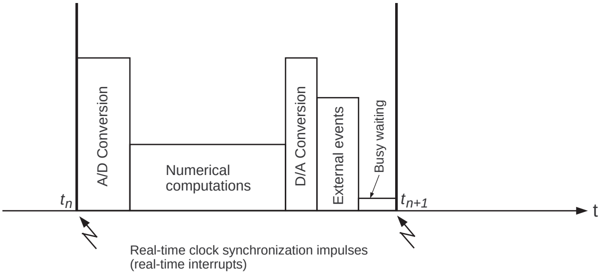
- The *digital to analog (D/A) converters* are set at the end of each integration step, i.e., the newest output information is put out through the D/A converters for inspection by the user, or for driving real hardware (for so-called *hardware-in-the-loop* simulations).

Introduction III

- The *digital to analog (D/A) converters* are set at the end of each integration step, i.e., the newest output information is put out through the D/A converters for inspection by the user, or for driving real hardware (for so-called *hardware-in-the-loop* simulations).

- *External events* are time events that are generated outside the simulation. External events are used for asynchronous communication with the simulation program, e.g. for the modification of parameter values, or for handling asynchronous readout requests, or for communication between several asynchronously running computer programs either on the same or different computers. External events are usually postponed to the end of the current step and replace a portion of the busy waiting period.

Introduction IV



Introduction V

The *interprocessor and intertask communication* mechanisms can be implemented in many different ways.

Introduction V

The *interprocessor and intertask communication* mechanisms can be implemented in many different ways.

- In some cases, it may be desirable to use the waiting time of the processor for background tasks, rather than waste it in a busy waiting loop. In that case, it is not sufficient for the real-time clock to *send a message* to the simulation program. Instead, it must use the *interrupt mechanism of the processor* on which the simulation is running to interrupt whatever other task the processor is currently working on.

Introduction V

The *interprocessor and intertask communication* mechanisms can be implemented in many different ways.

- In some cases, it may be desirable to use the waiting time of the processor for background tasks, rather than waste it in a busy waiting loop. In that case, it is not sufficient for the real-time clock to *send a message* to the simulation program. Instead, it must use the *interrupt mechanism of the processor* on which the simulation is running to interrupt whatever other task the processor is currently working on.

- In other cases, the same processor may be used for multiple real-time tasks using a *time-multiplexing* scheme. In that situation, the real-time clocks of the different tasks need to be synchronized with each other.

Introduction V

The *interprocessor and intertask communication* mechanisms can be implemented in many different ways.

- In some cases, it may be desirable to use the waiting time of the processor for background tasks, rather than waste it in a busy waiting loop. In that case, it is not sufficient for the real-time clock to *send a message* to the simulation program. Instead, it must use the *interrupt mechanism of the processor* on which the simulation is running to interrupt whatever other task the processor is currently working on.

- In other cases, the same processor may be used for multiple real-time tasks using a *time-multiplexing* scheme. In that situation, the real-time clocks of the different tasks need to be synchronized with each other.

- When the *interrupt mechanism of the real-time operating system* is being used, we also need to make sure that tasks that are related to the real-time operation cannot be interrupted by other tasks.

Introduction VI

The difficulties of real-time simulation are not of a conceptual nature. They have to do with keeping track of real time.

Introduction VI

The difficulties of real-time simulation are not of a conceptual nature. They have to do with keeping track of real time.

- How can we guarantee that all that needs to be accomplished during the integration step can be completed prior to the arrival of the next trigger impulse?

Introduction VI

The difficulties of real-time simulation are not of a conceptual nature. They have to do with keeping track of real time.

- How can we guarantee that all that needs to be accomplished during the integration step can be completed prior to the arrival of the next trigger impulse?
- How do we control the *computational load* of an integration algorithm during the execution of an integration step?

Introduction VI

The difficulties of real-time simulation are not of a conceptual nature. They have to do with keeping track of real time.

- How can we guarantee that all that needs to be accomplished during the integration step can be completed prior to the arrival of the next trigger impulse?
- How do we control the *computational load* of an integration algorithm during the execution of an integration step?
- What happens when we fail in this endeavor and cause an *over-run*?

Introduction VII

Previously, we introduced more and more bells and whistles that would help us in being able to *maximize user convenience*, to ensure the *robustness* of the modeling and simulation environment, and to guarantee the *correctness* of the simulation results obtained, but all these additional features were accompanied by a substantial amount of run-time overhead, and in many cases, the amount of time needed to bring these algorithms to completion was not fixed.

Introduction VII

Previously, we introduced more and more bells and whistles that would help us in being able to *maximize user convenience*, to ensure the *robustness* of the modeling and simulation environment, and to guarantee the *correctness* of the simulation results obtained, but all these additional features were accompanied by a substantial amount of run-time overhead, and in many cases, the amount of time needed to bring these algorithms to completion was not fixed.

- If we use an *implicit integration algorithm*, how can we know beforehand how many iterations will be needed to guarantee a prescribed tolerance of the results?

Introduction VII

Previously, we introduced more and more bells and whistles that would help us in being able to *maximize user convenience*, to ensure the *robustness* of the modeling and simulation environment, and to guarantee the *correctness* of the simulation results obtained, but all these additional features were accompanied by a substantial amount of run-time overhead, and in many cases, the amount of time needed to bring these algorithms to completion was not fixed.

- If we use an *implicit integration algorithm*, how can we know beforehand how many iterations will be needed to guarantee a prescribed tolerance of the results?
- If we do not limit the number of iterations available to the algorithm, how can we possibly know for sure that the step will be completed before the arrival of the next trigger impulse from the real-time clock?

Introduction VII

Previously, we introduced more and more bells and whistles that would help us in being able to *maximize user convenience*, to ensure the *robustness* of the modeling and simulation environment, and to guarantee the *correctness* of the simulation results obtained, but all these additional features were accompanied by a substantial amount of run-time overhead, and in many cases, the amount of time needed to bring these algorithms to completion was not fixed.

- If we use an *implicit integration algorithm*, how can we know beforehand how many iterations will be needed to guarantee a prescribed tolerance of the results?
- If we do not limit the number of iterations available to the algorithm, how can we possibly know for sure that the step will be completed before the arrival of the next trigger impulse from the real-time clock?
- *Iteration on state events* is a great thing. Yet, can we afford it under real-time constraints?

Introduction VII

Previously, we introduced more and more bells and whistles that would help us in being able to *maximize user convenience*, to ensure the *robustness* of the modeling and simulation environment, and to guarantee the *correctness* of the simulation results obtained, but all these additional features were accompanied by a substantial amount of run-time overhead, and in many cases, the amount of time needed to bring these algorithms to completion was not fixed.

- If we use an *implicit integration algorithm*, how can we know beforehand how many iterations will be needed to guarantee a prescribed tolerance of the results?
- If we do not limit the number of iterations available to the algorithm, how can we possibly know for sure that the step will be completed before the arrival of the next trigger impulse from the real-time clock?
- *Iteration on state events* is a great thing. Yet, can we afford it under real-time constraints?
- What happens if we do not iterate on the event? Can we still know something about the accuracy of the results obtained?

Introduction VIII

Real-time simulations are usually run on *dedicated processors* that often are bought just for the task at hand and have to be *as cheap as possible* in order to make the final product *competitive on the market*.

Introduction VIII

Real-time simulations are usually run on *dedicated processors* that often are bought just for the task at hand and have to be *as cheap as possible* in order to make the final product *competitive on the market*.

Consequently, *simulation speed* is of utmost importance, whereas *user convenience* is fairly irrelevant.

Introduction VIII

Real-time simulations are usually run on *dedicated processors* that often are bought just for the task at hand and have to be *as cheap as possible* in order to make the final product *competitive on the market*.

Consequently, *simulation speed* is of utmost importance, whereas *user convenience* is fairly irrelevant.

Consequently, whereas the average user of **Dymola** can simulate many quite involved and complex models without knowing anything about the numerics of the underlying simulation engine, an engineer designing a *real-time simulation program* cannot be protected from such knowledge.

Introduction VIII

Real-time simulations are usually run on *dedicated processors* that often are bought just for the task at hand and have to be *as cheap as possible* in order to make the final product *competitive on the market*.

Consequently, *simulation speed* is of utmost importance, whereas *user convenience* is fairly irrelevant.

Consequently, whereas the average user of **Dymola** can simulate many quite involved and complex models without knowing anything about the numerics of the underlying simulation engine, an engineer designing a *real-time simulation program* cannot be protected from such knowledge.

Real-time simulation designers must know their simulation algorithms intimately, or else, they will surely fail in their endeavors.

The Race Against Time

There are four things that we can do if we don't meet the schedule. We can:

The Race Against Time

There are four things that we can do if we don't meet the schedule. We can:

1. increase the step size, h, in order to make more time for the tasks that need to be completed,

The Race Against Time

There are four things that we can do if we don't meet the schedule. We can:

1. increase the step size, h, in order to make more time for the tasks that need to be completed,
2. make the function evaluation more efficient, i.e., optimize the program that represents our state-space model,

The Race Against Time

There are four things that we can do if we don't meet the schedule. We can:

1. increase the step size, h, in order to make more time for the tasks that need to be completed,
2. make the function evaluation more efficient, i.e., optimize the program that represents our state-space model,
3. improve the speed of the integration algorithm, e.g. by reducing the number of function evaluations necessary during one step, and finally

The Race Against Time

There are four things that we can do if we don't meet the schedule. We can:

1. increase the step size, h, in order to make more time for the tasks that need to be completed,
2. make the function evaluation more efficient, i.e., optimize the program that represents our state-space model,
3. improve the speed of the integration algorithm, e.g. by reducing the number of function evaluations necessary during one step, and finally
4. buy ourselves a faster computer.

The Race Against Time

There are four things that we can do if we don't meet the schedule. We can:

1. increase the step size, h, in order to make more time for the tasks that need to be completed,
2. make the function evaluation more efficient, i.e., optimize the program that represents our state-space model,
3. improve the speed of the integration algorithm, e.g. by reducing the number of function evaluations necessary during one step, and finally
4. buy ourselves a faster computer.

The fourth approach may sound like a measure of last resort, but in these times of cheap hardware and expensive software and manpower, it may often be the wisest thing to do.

The Race Against Time II

The first approach is interesting. Until now, the step size was always bounded from the top due to accuracy and stability requirements. Now suddenly, *the step size is also bounded from the bottom*.

The Race Against Time II

The first approach is interesting. Until now, the step size was always bounded from the top due to accuracy and stability requirements. Now suddenly, *the step size is also bounded from the bottom*.

- We cannot reduce the step size to a value smaller than the total real time needed to perform all the computations associated with simulating the model across one step plus the real time needed for dealing with the administration of the simulation during that step.

The Race Against Time II

The first approach is interesting. Until now, the step size was always bounded from the top due to accuracy and stability requirements. Now suddenly, *the step size is also bounded from the bottom*.

- We cannot reduce the step size to a value smaller than the total real time needed to perform all the computations associated with simulating the model across one step plus the real time needed for dealing with the administration of the simulation during that step.
- Especially, when we are using an explicit integration algorithm, as we most likely will in order to avoid the need for iteration, the two bounds are interrelated. By increasing the step size to avoid over-runs, we are weakening the numerical stability of the algorithm.

The Race Against Time II

The first approach is interesting. Until now, the step size was always bounded from the top due to accuracy and stability requirements. Now suddenly, *the step size is also bounded from the bottom*.

- We cannot reduce the step size to a value smaller than the total real time needed to perform all the computations associated with simulating the model across one step plus the real time needed for dealing with the administration of the simulation during that step.
- Especially, when we are using an explicit integration algorithm, as we most likely will in order to avoid the need for iteration, the two bounds are interrelated. By increasing the step size to avoid over-runs, we are weakening the numerical stability of the algorithm.
- If it happens that the lower bound is larger than the upper, then we are in real trouble.

The Race Against Time III

The second approach is interesting also.

The Race Against Time III

The second approach is interesting also.

- In order to reduce the computational load associated with function evaluations, we may consider *simplifying the model*.

The Race Against Time III

The second approach is interesting also.

- In order to reduce the computational load associated with function evaluations, we may consider *simplifying the model*.

- In particular, we may choose to throw out the fastest modes (eigenvalues) from the model. This will also help with the numerical stability of the scheme, thereby allowing us to increase the step size some more.

The Race Against Time III

The second approach is interesting also.

- In order to reduce the computational load associated with function evaluations, we may consider *simplifying the model*.
- In particular, we may choose to throw out the fastest modes (eigenvalues) from the model. This will also help with the numerical stability of the scheme, thereby allowing us to increase the step size some more.
- Often the problems with spending too much time in function evaluations is related to needs for *interpolation in data tables*.

The Race Against Time III

The second approach is interesting also.

- In order to reduce the computational load associated with function evaluations, we may consider *simplifying the model*.
- In particular, we may choose to throw out the fastest modes (eigenvalues) from the model. This will also help with the numerical stability of the scheme, thereby allowing us to increase the step size some more.
- Often the problems with spending too much time in function evaluations is related to needs for *interpolation in data tables*.
- An attractive answer to this problem may be the *parallelization of the function evaluation*, e.g. by *implementing the model as a neural network*.

The Race Against Time IV

The third approach is directly related to the integration algorithms themselves, and as these are the central focus point of this class, it is the third solution that we shall be pursuing in more detail in this presentation.

Suitable Numerical Integration Methods

Single-step vs. multi-step algorithms

Suitable Numerical Integration Methods

Single-step vs. multi-step algorithms

- In general-purpose simulations, we may prefer single-step algorithms, because they offer cheaper step-size control and allow us to use larger steps.

Suitable Numerical Integration Methods

Single-step vs. multi-step algorithms

- In general-purpose simulations, we may prefer single-step algorithms, because they offer cheaper step-size control and allow us to use larger steps.
- In real-time simulation, step-size control is not an option, since the steps need to be synchronized with the real-time clock. Thus, one of the major advantages of single-step algorithms is gone.

Suitable Numerical Integration Methods

Single-step vs. multi-step algorithms

- In general-purpose simulations, we may prefer single-step algorithms, because they offer cheaper step-size control and allow us to use larger steps.
- In real-time simulation, step-size control is not an option, since the steps need to be synchronized with the real-time clock. Thus, one of the major advantages of single-step algorithms is gone.
- The larger step sizes may not help us either, because we need to sample external input signals and incorporate them in the model. The larger the step size, the less frequently we are able to update the external input signals.

Suitable Numerical Integration Methods

Single-step vs. multi-step algorithms

- In general-purpose simulations, we may prefer single-step algorithms, because they offer cheaper step-size control and allow us to use larger steps.
- In real-time simulation, step-size control is not an option, since the steps need to be synchronized with the real-time clock. Thus, one of the major advantages of single-step algorithms is gone.
- The larger step sizes may not help us either, because we need to sample external input signals and incorporate them in the model. The larger the step size, the less frequently we are able to update the external input signals.
- For this reason, we may prefer smaller and cheaper steps over larger and more expensive ones even if the overall computational efficiency of using larger steps is better.

Suitable Numerical Integration Methods

Single-step vs. multi-step algorithms

- In general-purpose simulations, we may prefer single-step algorithms, because they offer cheaper step-size control and allow us to use larger steps.
- In real-time simulation, step-size control is not an option, since the steps need to be synchronized with the real-time clock. Thus, one of the major advantages of single-step algorithms is gone.
- The larger step sizes may not help us either, because we need to sample external input signals and incorporate them in the model. The larger the step size, the less frequently we are able to update the external input signals.
- For this reason, we may prefer smaller and cheaper steps over larger and more expensive ones even if the overall computational efficiency of using larger steps is better.

Explicit linear multi-step methods are well suited for real-time simulation, as long as the model to be simulated is neither stiff nor discontinuous.

Suitable Numerical Integration Methods II

Explicit vs. implicit algorithms

Suitable Numerical Integration Methods II

Explicit vs. implicit algorithms

- In general-purpose simulation environments, we may prefer (implicit) stiff-system solvers, as they increase the robustness of our simulations, thereby protecting the users from having to understand much of the intricacies of the simulation engine they are using.

Suitable Numerical Integration Methods II

Explicit vs. implicit algorithms

- In general-purpose simulation environments, we may prefer (implicit) stiff-system solvers, as they increase the robustness of our simulations, thereby protecting the users from having to understand much of the intricacies of the simulation engine they are using.

- In real-time simulation, run-time efficiency is much more important than convenience of use. Also, implicit solvers require a Newton iteration, which makes it impossible to know beforehand how long (how many iterations) it will take to complete one step of the simulation.

Suitable Numerical Integration Methods II

Explicit vs. implicit algorithms

- In general-purpose simulation environments, we may prefer (implicit) stiff-system solvers, as they increase the robustness of our simulations, thereby protecting the users from having to understand much of the intricacies of the simulation engine they are using.
- In real-time simulation, run-time efficiency is much more important than convenience of use. Also, implicit solvers require a Newton iteration, which makes it impossible to know beforehand how long (how many iterations) it will take to complete one step of the simulation.
- For this reason, we may prefer explicit over implicit solvers in real-time simulation.

Suitable Numerical Integration Methods II

Explicit vs. implicit algorithms

- In general-purpose simulation environments, we may prefer (implicit) stiff-system solvers, as they increase the robustness of our simulations, thereby protecting the users from having to understand much of the intricacies of the simulation engine they are using.
- In real-time simulation, run-time efficiency is much more important than convenience of use. Also, implicit solvers require a Newton iteration, which makes it impossible to know beforehand how long (how many iterations) it will take to complete one step of the simulation.
- For this reason, we may prefer explicit over implicit solvers in real-time simulation.

Explicit ODE solvers are preferred for real-time simulation, as long as the model to be simulated is non-stiff.

Suitable Numerical Integration Methods III

High-order vs. Low-order Algorithms

Suitable Numerical Integration Methods III

High-order vs. Low-order Algorithms

- In general-purpose simulation environments, we usually prefer higher-order over lower-order methods. The reason is that we normally set the error tolerance to 10^{-4} (default value in **Dymola**), and at that accuracy requirement, a 4^{th}- or 5^{th}-order method is usually most economical. In some cases, such as the simulation of *chemical reaction dynamics*, we may tighten our accuracy requirements even more to e.g. 10^{-10}, as otherwise the Newton iterations may not converge.

Suitable Numerical Integration Methods III

High-order vs. Low-order Algorithms

- In general-purpose simulation environments, we usually prefer higher-order over lower-order methods. The reason is that we normally set the error tolerance to 10^{-4} (default value in **Dymola**), and at that accuracy requirement, a 4^{th}- or 5^{th}-order method is usually most economical. In some cases, such as the simulation of *chemical reaction dynamics*, we may tighten our accuracy requirements even more to e.g. 10^{-10}, as otherwise the Newton iterations may not converge.

- In real-time simulation, speed is of the essence, and we may simply not be able to afford such high accuracy. For this reason, we hardly ever use an algorithm of higher than 3^{rd}-order in a real-time simulation.

Suitable Numerical Integration Methods III

High-order vs. Low-order Algorithms

- In general-purpose simulation environments, we usually prefer higher-order over lower-order methods. The reason is that we normally set the error tolerance to 10^{-4} (default value in **Dymola**), and at that accuracy requirement, a 4^{th}- or 5^{th}-order method is usually most economical. In some cases, such as the simulation of *chemical reaction dynamics*, we may tighten our accuracy requirements even more to e.g. 10^{-10}, as otherwise the Newton iterations may not converge.

- In real-time simulation, speed is of the essence, and we may simply not be able to afford such high accuracy. For this reason, we hardly ever use an algorithm of higher than 3^{rd}-order in a real-time simulation.

- **AB3** may be a good choice for a real-time simulation of a non-stiff model, but many designers of real-time simulation software actually use **FE** . . . because that is an algorithm they fully understand and know how to implement.

Suitable Numerical Integration Methods III

High-order vs. Low-order Algorithms

- In general-purpose simulation environments, we usually prefer higher-order over lower-order methods. The reason is that we normally set the error tolerance to 10^{-4} (default value in **Dymola**), and at that accuracy requirement, a 4^{th}- or 5^{th}-order method is usually most economical. In some cases, such as the simulation of *chemical reaction dynamics*, we may tighten our accuracy requirements even more to e.g. 10^{-10}, as otherwise the Newton iterations may not converge.
- In real-time simulation, speed is of the essence, and we may simply not be able to afford such high accuracy. For this reason, we hardly ever use an algorithm of higher than 3^{rd}-order in a real-time simulation.
- **AB3** may be a good choice for a real-time simulation of a non-stiff model, but many designers of real-time simulation software actually use **FE** . . . because that is an algorithm they fully understand and know how to implement.

Explicit low-order ODE solvers are preferred for real-time simulation, as long as the model to be simulated is non-stiff.

Suitable Numerical Integration Methods IV

It is kind of sad . . .

Suitable Numerical Integration Methods IV

It is kind of sad . . .

Here we went and developed more and more sophisticated numerical ODE and DAE solvers that could do ever more good things for us, and now, faced with real-time constraints, we simply resign.

Suitable Numerical Integration Methods IV

It is kind of sad . . .

Here we went and developed more and more sophisticated numerical ODE and DAE solvers that could do ever more good things for us, and now, faced with real-time constraints, we simply resign.

Most real-time simulations make use of very crude (garbage!) ODE solvers

Suitable Numerical Integration Methods IV

It is kind of sad . . .

Here we went and developed more and more sophisticated numerical ODE and DAE solvers that could do ever more good things for us, and now, faced with real-time constraints, we simply resign.

Most real-time simulations make use of very crude (garbage!) ODE solvers to simulate heavily simplified (garbage!) models,

Suitable Numerical Integration Methods IV

It is kind of sad . . .

Here we went and developed more and more sophisticated numerical ODE and DAE solvers that could do ever more good things for us, and now, faced with real-time constraints, we simply resign.

Most real-time simulations make use of very crude (garbage!) ODE solvers to simulate heavily simplified (garbage!) models, thereby generating highly inaccurate (garbage!) simulation trajectories . . .

Suitable Numerical Integration Methods IV

It is kind of sad . . .

Here we went and developed more and more sophisticated numerical ODE and DAE solvers that could do ever more good things for us, and now, faced with real-time constraints, we simply resign.

Most real-time simulations make use of very crude (garbage!) ODE solvers to simulate heavily simplified (garbage!) models, thereby generating highly inaccurate (garbage!) simulation trajectories . . .

. . . but at least, they do it very fast.

Linearly Implicit Methods

What do we do if the model to be simulated in real time is stiff?

Linearly Implicit Methods

What do we do if the model to be simulated in real time is stiff?

- We already know that we cannot afford a Newton iteration, because we cannot guarantee that it will converge in time before the next real-time synchronization impulse arrives.

Linearly Implicit Methods

What do we do if the model to be simulated in real time is stiff?

- We already know that we cannot afford a Newton iteration, because we cannot guarantee that it will converge in time before the next real-time synchronization impulse arrives.

- Yet, we also know that we need a stiff system solver.

Linearly Implicit Methods

What do we do if the model to be simulated in real time is stiff?

- We already know that we cannot afford a Newton iteration, because we cannot guarantee that it will converge in time before the next real-time synchronization impulse arrives.
- Yet, we also know that we need a stiff system solver.
- Hence we are in a real quandary.

Linearly Implicit Methods

What do we do if the model to be simulated in real time is stiff?

- We already know that we cannot afford a Newton iteration, because we cannot guarantee that it will converge in time before the next real-time synchronization impulse arrives.
- Yet, we also know that we need a stiff system solver.
- Hence we are in a real quandary.
- Or are we not?

Linearly Implicit Methods II

Given the explicit state-space model:

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

Linearly Implicit Methods II

Given the explicit state-space model:

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

We start with **BE**:

$$\mathbf{x_{k+1}} = \mathbf{x_k} + h \cdot \mathbf{f}(\mathbf{x_{k+1}}, t_{k+1})$$

Linearly Implicit Methods II

Given the explicit state-space model:

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

We start with **BE**:

$$\mathbf{x_{k+1}} = \mathbf{x_k} + h \cdot \mathbf{f}(\mathbf{x_{k+1}}, t_{k+1})$$

We develop $\mathbf{f}(\mathbf{x_{k+1}}, t_{k+1})$ into a Taylor series around $\mathbf{f}(\mathbf{x_k}, t_k)$:

$$\mathbf{f}(\mathbf{x_{k+1}}, t_{k+1}) = \mathbf{f}(\mathbf{x_k}, t_k) + \mathcal{J}_{\mathbf{x_k}, t_k} \cdot (\mathbf{x_{k+1}} - \mathbf{x_k}) + \ldots$$

Linearly Implicit Methods II

Given the explicit state-space model:

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

We start with **BE**:

$$\mathbf{x_{k+1}} = \mathbf{x_k} + h \cdot \mathbf{f}(\mathbf{x_{k+1}}, t_{k+1})$$

We develop $\mathbf{f}(\mathbf{x_{k+1}}, t_{k+1})$ into a Taylor series around $\mathbf{f}(\mathbf{x_k}, t_k)$:

$$\mathbf{f}(\mathbf{x_{k+1}}, t_{k+1}) = \mathbf{f}(\mathbf{x_k}, t_k) + \mathcal{J}_{\mathbf{x_k}, t_k} \cdot (\mathbf{x_{k+1}} - \mathbf{x_k}) + \ldots$$

where:

$$\mathcal{J}_{\mathbf{x_k}, t_k} = \left.\frac{\partial \mathbf{f}}{\partial \mathbf{x}}\right|_{\mathbf{x_k}, t_k}$$

Linearly Implicit Methods III

We can truncate the Taylor series after the linear term, and plug the expression into the solver equation:

$$\mathbf{x_{k+1}} = \mathbf{x_k} + h \cdot [\mathbf{f}(\mathbf{x_k}, t_k) + \mathcal{J}_{\mathbf{x_k}, t_k} \cdot (\mathbf{x_{k+1}} - \mathbf{x_k})]$$

Linearly Implicit Methods III

We can truncate the Taylor series after the linear term, and plug the expression into the solver equation:

$$\mathbf{x_{k+1}} = \mathbf{x_k} + h \cdot [\mathbf{f}(\mathbf{x_k}, t_k) + \mathcal{J}_{\mathbf{x_k},t_k} \cdot (\mathbf{x_{k+1}} - \mathbf{x_k})]$$

Solving for $\mathbf{x_{k+1}}$:

$$(I - h \cdot \mathcal{J}_{\mathbf{x_k},t_k}) \cdot \mathbf{x_{k+1}} = (I - h \cdot \mathcal{J}_{\mathbf{x_k},t_k}) \cdot \mathbf{x_k} + h \cdot \mathbf{f}(\mathbf{x_k}, t_k)$$

Linearly Implicit Methods III

We can truncate the Taylor series after the linear term, and plug the expression into the solver equation:

$$\mathbf{x_{k+1}} = \mathbf{x_k} + h \cdot [\mathbf{f}(\mathbf{x_k}, t_k) + \mathcal{J}_{\mathbf{x_k}, t_k} \cdot (\mathbf{x_{k+1}} - \mathbf{x_k})]$$

Solving for $\mathbf{x_{k+1}}$:

$$(I - h \cdot \mathcal{J}_{\mathbf{x_k}, t_k}) \cdot \mathbf{x_{k+1}} = (I - h \cdot \mathcal{J}_{\mathbf{x_k}, t_k}) \cdot \mathbf{x_k} + h \cdot \mathbf{f}(\mathbf{x_k}, t_k)$$

or:

$$\mathbf{x_{k+1}} = \mathbf{x_k} + h \cdot (I - h \cdot \mathcal{J}_{\mathbf{x_k}, t_k})^{-1} \cdot \mathbf{f}(\mathbf{x_k}, t_k)$$

Linearly Implicit Methods III

We can truncate the Taylor series after the linear term, and plug the expression into the solver equation:

$$\mathbf{x_{k+1}} = \mathbf{x_k} + h \cdot [\mathbf{f}(\mathbf{x_k}, t_k) + \mathcal{J}_{\mathbf{x_k},t_k} \cdot (\mathbf{x_{k+1}} - \mathbf{x_k})]$$

Solving for $\mathbf{x_{k+1}}$:

$$(I - h \cdot \mathcal{J}_{\mathbf{x_k},t_k}) \cdot \mathbf{x_{k+1}} = (I - h \cdot \mathcal{J}_{\mathbf{x_k},t_k}) \cdot \mathbf{x_k} + h \cdot \mathbf{f}(\mathbf{x_k}, t_k)$$

or:

$$\mathbf{x_{k+1}} = \mathbf{x_k} + h \cdot (I - h \cdot \mathcal{J}_{\mathbf{x_k},t_k})^{-1} \cdot \mathbf{f}(\mathbf{x_k}, t_k)$$

The new method *approximates backward Euler*, since the Taylor series was truncated after the linear term. As the Taylor series gets multiplied by h, the approximation is second-order accurate, and since the entire method is only first-order accurate, the approximation is acceptable.

Linearly Implicit Methods IV

What does the stability domain of this new method look like?

Linearly Implicit Methods IV

What does the stability domain of this new method look like?

To answer this question, we plug the linear system:

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

into the solver.

Linearly Implicit Methods IV

What does the stability domain of this new method look like?

To answer this question, we plug the linear system:

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

into the solver.

With $\mathbf{f}(\mathbf{x_k}, t_k) = \mathbf{A} \cdot \mathbf{x_k}$ and $\mathcal{J}_{\mathbf{x_k}, t_k} = \mathbf{A}$:

$$\mathbf{x_{k+1}} = \mathbf{x_k} + h \cdot (I - \mathbf{A}h)^{-1} \cdot \mathbf{A} \cdot \mathbf{x_k}$$

Linearly Implicit Methods IV

What does the stability domain of this new method look like?

To answer this question, we plug the linear system:

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

into the solver.

With $\mathbf{f}(\mathbf{x_k}, t_k) = \mathbf{A} \cdot \mathbf{x_k}$ and $\mathcal{J}_{\mathbf{x_k}, t_k} = \mathbf{A}$:

$$\mathbf{x_{k+1}} = \mathbf{x_k} + h \cdot (I - \mathbf{A}h)^{-1} \cdot \mathbf{A} \cdot \mathbf{x_k}$$

Thus:

$$\begin{aligned}
\mathbf{x_{k+1}} &= (I - \mathbf{A}h)^{-1} \cdot (I - \mathbf{A}h) \cdot \mathbf{x_k} + h \cdot (I - \mathbf{A}h)^{-1} \cdot \mathbf{A} \cdot \mathbf{x_k} \\
&= (I - \mathbf{A}h)^{-1} \cdot [(I - \mathbf{A}h) \cdot \mathbf{x_k} + h \cdot \mathbf{A} \cdot \mathbf{x_k}] \\
&= (I - \mathbf{A}h)^{-1} \cdot \mathbf{x_k}
\end{aligned}$$

Linearly Implicit Methods IV

What does the stability domain of this new method look like?

To answer this question, we plug the linear system:

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

into the solver.

With $\mathbf{f}(\mathbf{x_k}, t_k) = \mathbf{A} \cdot \mathbf{x_k}$ and $\mathcal{J}_{\mathbf{x_k}, t_k} = \mathbf{A}$:

$$\mathbf{x_{k+1}} = \mathbf{x_k} + h \cdot (I - \mathbf{A}h)^{-1} \cdot \mathbf{A} \cdot \mathbf{x_k}$$

Thus:

$$\begin{aligned}
\mathbf{x_{k+1}} &= (I - \mathbf{A}h)^{-1} \cdot (I - \mathbf{A}h) \cdot \mathbf{x_k} + h \cdot (I - \mathbf{A}h)^{-1} \cdot \mathbf{A} \cdot \mathbf{x_k} \\
&= (I - \mathbf{A}h)^{-1} \cdot [(I - \mathbf{A}h) \cdot \mathbf{x_k} + h \cdot \mathbf{A} \cdot \mathbf{x_k}] \\
&= (I - \mathbf{A}h)^{-1} \cdot \mathbf{x_k}
\end{aligned}$$

The numerical stability domain of the new method is the same as that of BE.

Linearly Implicit Methods V

- The *linearly implicit backward Euler (LIBE)* method is a *stiff-system solver*.

Linearly Implicit Methods V

- The *linearly implicit backward Euler (LIBE)* method is a *stiff-system solver*.
- The method is similar to Forward Euler, but differs by the presence of the term $(I - h \cdot \mathcal{J}_{\mathbf{x_k},t_k})^{-1}$.

Linearly Implicit Methods V

- The *linearly implicit backward Euler (LIBE)* method is a *stiff-system solver*.
- The method is similar to Forward Euler, but differs by the presence of the term $(I - h \cdot \mathcal{J}_{\mathbf{x_k},t_k})^{-1}$.
- From a computational point of view, that term implies that the algorithm has to calculate the Jacobian at each step and then either solve a linear equation system or invert a matrix.

Linearly Implicit Methods V

- The *linearly implicit backward Euler (LIBE)* method is a *stiff-system solver*.
- The method is similar to Forward Euler, but differs by the presence of the term $(I - h \cdot \mathcal{J}_{\mathbf{x_k},t_k})^{-1}$.
- From a computational point of view, that term implies that the algorithm has to calculate the Jacobian at each step and then either solve a linear equation system or invert a matrix.
- Despite the fact that those calculations may turn out to be quite expensive, the *computational effort is predictable*, which makes the method suited for real-time simulation.

Linearly Implicit Methods V

- The *linearly implicit backward Euler (LIBE)* method is a *stiff-system solver*.
- The method is similar to Forward Euler, but differs by the presence of the term $(I - h \cdot \mathcal{J}_{\mathbf{x_k},t_k})^{-1}$.
- From a computational point of view, that term implies that the algorithm has to calculate the Jacobian at each step and then either solve a linear equation system or invert a matrix.
- Despite the fact that those calculations may turn out to be quite expensive, the *computational effort is predictable*, which makes the method suited for real-time simulation.
- Low-order linearly implicit methods may indeed often be the best choice for real-time simulation of stiff systems.

Linearly Implicit Methods V

- The *linearly implicit backward Euler (LIBE)* method is a *stiff-system solver*.
- The method is similar to Forward Euler, but differs by the presence of the term $(I - h \cdot \mathcal{J}_{\mathbf{x_k},t_k})^{-1}$.
- From a computational point of view, that term implies that the algorithm has to calculate the Jacobian at each step and then either solve a linear equation system or invert a matrix.
- Despite the fact that those calculations may turn out to be quite expensive, the *computational effort is predictable*, which makes the method suited for real-time simulation.
- Low-order linearly implicit methods may indeed often be the best choice for real-time simulation of stiff systems.
- However, they share one drawback with implicit methods: if the size of the problem is large, then the solution of the resulting linear equation system is computationally expensive.

Linearly Implicit Methods VI

- It is important to recognize that the numerical stability domain of the **LIBE** algorithm is only the same as that of **BE**, as long as the *Jacobian is computed accurately*.

Linearly Implicit Methods VI

- It is important to recognize that the numerical stability domain of the **LIBE** algorithm is only the same as that of **BE**, as long as the *Jacobian is computed accurately*.

- If the Jacobian is only approximated, the stability domain changes, and it takes little approximation, until it flips over to the left-half complex plane, i.e., until stiff stability is getting lost.

Linearly Implicit Methods VI

- It is important to recognize that the numerical stability domain of the **LIBE** algorithm is only the same as that of **BE**, as long as the *Jacobian is computed accurately*.
- If the Jacobian is only approximated, the stability domain changes, and it takes little approximation, until it flips over to the left-half complex plane, i.e., until stiff stability is getting lost.
- If the Jacobian is "approximated" by the zero matrix, **LIBE** turns into **FE**.

Linearly Implicit Methods VI

- It is important to recognize that the numerical stability domain of the **LIBE** algorithm is only the same as that of **BE**, as long as the *Jacobian is computed accurately*.
- If the Jacobian is only approximated, the stability domain changes, and it takes little approximation, until it flips over to the left-half complex plane, i.e., until stiff stability is getting lost.
- If the Jacobian is "approximated" by the zero matrix, **LIBE** turns into **FE**.
- A good approach may be to compute the non-zero elements of the Jacobian, which is usually quite sparse, analytically, i.e., by *symbolic (algebraic) differentiation*.

Linearly Implicit Methods VI

- It is important to recognize that the numerical stability domain of the **LIBE** algorithm is only the same as that of **BE**, as long as the *Jacobian is computed accurately*.
- If the Jacobian is only approximated, the stability domain changes, and it takes little approximation, until it flips over to the left-half complex plane, i.e., until stiff stability is getting lost.
- If the Jacobian is “approximated” by the zero matrix, **LIBE** turns into **FE**.
- A good approach may be to compute the non-zero elements of the Jacobian, which is usually quite sparse, analytically, i.e., by *symbolic (algebraic) differentiation*.
- The underlying linear system can then be solved by *tearing* to minimize the number of iteration variables.

Linearly Implicit Methods VI

- It is important to recognize that the numerical stability domain of the **LIBE** algorithm is only the same as that of **BE**, as long as the *Jacobian is computed accurately*.
- If the Jacobian is only approximated, the stability domain changes, and it takes little approximation, until it flips over to the left-half complex plane, i.e., until stiff stability is getting lost.
- If the Jacobian is "approximated" by the zero matrix, **LIBE** turns into **FE**.
- A good approach may be to compute the non-zero elements of the Jacobian, which is usually quite sparse, analytically, i.e., by *symbolic (algebraic) differentiation*.
- The underlying linear system can then be solved by *tearing* to minimize the number of iteration variables.
- Since the equation system to be solved is linear, the iteration will converge within a single iteration step. Thus, the computational load can be anticipated accurately.

Multi-rate Integration

There are many applications, in which the stiffness is due to the presence of a sub-system with very fast dynamics compared to the rest of the system.

Multi-rate Integration

There are many applications, in which the stiffness is due to the presence of a sub-system with very fast dynamics compared to the rest of the system.

Typical examples of this can be found in *multi-domain physical systems*, since the components of different physical domains usually involve distinct time constants.

Multi-rate Integration

There are many applications, in which the stiffness is due to the presence of a sub-system with very fast dynamics compared to the rest of the system.

Typical examples of this can be found in *multi-domain physical systems*, since the components of different physical domains usually involve distinct time constants.

- In a *mechatronics system*, the electrical time constants are usually somewhere between 1 *kHz* and 10 *kHz*, whereas the mechanical time constants are in the order of 10 *Hz* to 100 *Hz*.

Multi-rate Integration

There are many applications, in which the stiffness is due to the presence of a sub-system with very fast dynamics compared to the rest of the system.

Typical examples of this can be found in *multi-domain physical systems*, since the components of different physical domains usually involve distinct time constants.

- In a *mechatronics system*, the electrical time constants are usually somewhere between 1 *kHz* and 10 *kHz*, whereas the mechanical time constants are in the order of 10 *Hz* to 100 *Hz*.
- In the *thermal analysis of an integrated electronic circuit*, we need to simulate simultaneously the electrical and thermal properties of the system. Yet, the thermal time constants are usually below 1 *Hz*.

Multi-rate Integration

There are many applications, in which the stiffness is due to the presence of a sub-system with very fast dynamics compared to the rest of the system.

Typical examples of this can be found in *multi-domain physical systems*, since the components of different physical domains usually involve distinct time constants.

- In a *mechatronics system*, the electrical time constants are usually somewhere between 1 *kHz* and 10 *kHz*, whereas the mechanical time constants are in the order of 10 *Hz* to 100 *Hz*.

- In the *thermal analysis of an integrated electronic circuit*, we need to simulate simultaneously the electrical and thermal properties of the system. Yet, the thermal time constants are usually below 1 *Hz*.

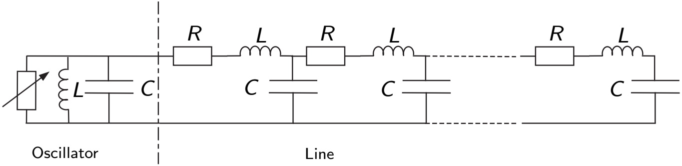
In those cases, it may make sense to split the simulation into a fast and a slow part and integrate these parts using different step sizes.

Multi-rate Integration II

Let us introduce the idea with an example:

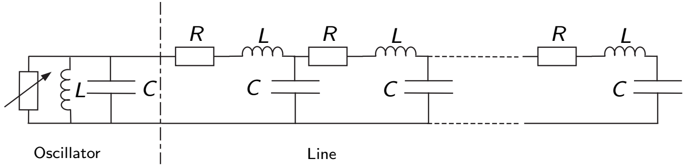
Multi-rate Integration II

Let us introduce the idea with an example:



Multi-rate Integration II

Let us introduce the idea with an example:

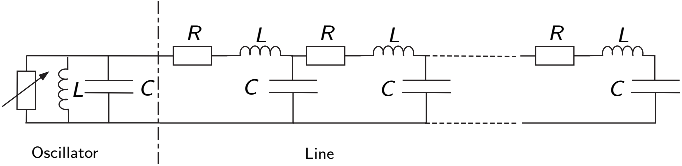


We shall assume that the non-linear resistor of the oscillator circuit satisfies the law:

$$i_R = k \cdot u_R^3 - u_R$$

Multi-rate Integration II

Let us introduce the idea with an example:



We shall assume that the non-linear resistor of the oscillator circuit satisfies the law:

$$i_R = k \cdot u_R^3 - u_R$$

The example is clearly non-physical, as the "resistor" doesn't operate strictly in the first and third quadrant of the plane spanned by u_R and i_R. Consequently, there are times when the resistor converts thermal energy to electrical energy, something no respectable resistor will ever do.

Multi-rate Integration III

The system can be described by the following set of state equations:

$$
\begin{aligned}
\frac{di_L}{dt} &= \frac{1}{L} u_C \\
\frac{du_C}{dt} &= \frac{1}{C}(u_C - k \cdot u_C^3 - i_L - i_1) \\
\frac{di_1}{dt} &= \frac{1}{L} u_C - \frac{R}{L} i_1 - \frac{1}{L} u_1 \\
\frac{du_1}{dt} &= \frac{1}{C} i_1 - \frac{1}{C} i_2 \\
\frac{di_2}{dt} &= \frac{1}{L} u_1 - \frac{R}{L} i_2 - \frac{1}{L} u_2 \\
\frac{du_2}{dt} &= \frac{1}{C} i_2 - \frac{1}{C} i_3 \\
&\vdots \\
\frac{di_n}{dt} &= \frac{1}{L} u_{n-1} - \frac{R}{L} i_n - \frac{1}{L} u_n \\
\frac{du_n}{dt} &= \frac{1}{C} i_n
\end{aligned}
$$

Multi-rate Integration IV

- Let us assume that the transmission line has 5 stages (i.e., $n = 5$), and the parameters are $L = 10\ \mathrm{mH}$, $C = 1\ \mathrm{mF}$, $R = 10\Omega$, and $k = 0.04$.

Multi-rate Integration IV

- Let us assume that the transmission line has 5 stages (i.e., $n = 5$), and the parameters are $L = 10\ \mathrm{mH}$, $C = 1\ \mathrm{mF}$, $R = 10\Omega$, and $k = 0.04$.

- If we simulate the system using **FE**, we need to use a step size no greater than $h = 10^{-4}\ \mathrm{seconds}$. Otherwise, the oscillator output (u_C) is computed with an error that is unacceptably large.

Multi-rate Integration IV

- Let us assume that the transmission line has 5 stages (i.e., $n = 5$), and the parameters are $L = 10 \text{ mH}$, $C = 1 \text{ mF}$, $R = 10\Omega$, and $k = 0.04$.
- If we simulate the system using **FE**, we need to use a step size no greater than $h = 10^{-4} \text{ seconds}$. Otherwise, the oscillator output (u_C) is computed with an error that is unacceptably large.
- On the other hand, if we eliminate the oscillator from the circuit and replace it by a voltage source, the transmission line alone can be simulated with a step size that is 10 times bigger.

Multi-rate Integration IV

- Let us assume that the transmission line has 5 stages (i.e., $n = 5$), and the parameters are $L = 10\ \mathrm{mH}$, $C = 1\ \mathrm{mF}$, $R = 10\Omega$, and $k = 0.04$.

- If we simulate the system using **FE**, we need to use a step size no greater than $h = 10^{-4}\ \mathrm{seconds}$. Otherwise, the oscillator output (u_C) is computed with an error that is unacceptably large.

- On the other hand, if we eliminate the oscillator from the circuit and replace it by a voltage source, the transmission line alone can be simulated with a step size that is 10 times bigger.

- Thus, we decided to split the system into two subsystems, the oscillator circuit and the transmission line, using two different step sizes: $10^{-4}\ \mathrm{seconds}$ for the former, and $10^{-3}\ \mathrm{seconds}$ for the latter.

Multi-rate Integration IV

- Let us assume that the transmission line has 5 stages (i.e., $n = 5$), and the parameters are $L = 10\ \mathrm{mH}$, $C = 1\ \mathrm{mF}$, $R = 10\Omega$, and $k = 0.04$.
- If we simulate the system using **FE**, we need to use a step size no greater than $h = 10^{-4}\ \mathrm{seconds}$. Otherwise, the oscillator output (u_C) is computed with an error that is unacceptably large.
- On the other hand, if we eliminate the oscillator from the circuit and replace it by a voltage source, the transmission line alone can be simulated with a step size that is 10 times bigger.
- Thus, we decided to split the system into two subsystems, the oscillator circuit and the transmission line, using two different step sizes: $10^{-4}\ \mathrm{seconds}$ for the former, and $10^{-3}\ \mathrm{seconds}$ for the latter.
- In this way, we integrate the fast but small (2^{nd}-order) sub-system using a small step size, whereas we integrate the slow and large (10^{th}-order) sub-system using a ten times larger step size.

Multi-rate Integration V

As a consequence, during each millisecond of real time, the computer has to evaluate ten times the two scalar functions corresponding to the two first state equations, whereas it only needs to evaluate once the remaining ten functions. Thus, the number of floating-point operations is reduced by about a factor of four compared with a regular simulation using a single step size throughout.

Multi-rate Integration V

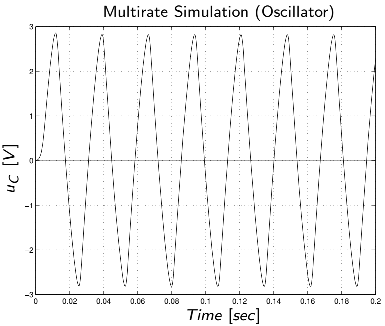
As a consequence, during each millisecond of real time, the computer has to evaluate ten times the two scalar functions corresponding to the two first state equations, whereas it only needs to evaluate once the remaining ten functions. Thus, the number of floating-point operations is reduced by about a factor of four compared with a regular simulation using a single step size throughout.

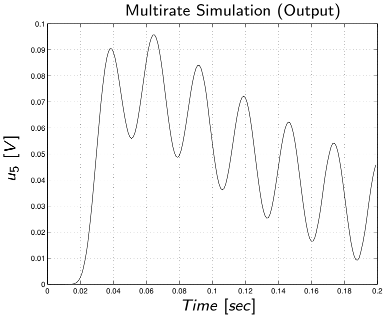
The simulation results are shown below:

Multi-rate Integration V

As a consequence, during each millisecond of real time, the computer has to evaluate ten times the two scalar functions corresponding to the two first state equations, whereas it only needs to evaluate once the remaining ten functions. Thus, the number of floating-point operations is reduced by about a factor of four compared with a regular simulation using a single step size throughout.

The simulation results are shown below:





Multi-rate Integration VI

We can generalize this procedure to systems of the form:

$$\begin{aligned} \dot{\mathbf{x}}_\mathbf{f}(t) &= \mathbf{f}_\mathbf{f}(\mathbf{x}_\mathbf{f}, \mathbf{x}_\mathbf{s}, t) \\ \dot{\mathbf{x}}_\mathbf{s}(t) &= \mathbf{f}_\mathbf{s}(\mathbf{x}_\mathbf{f}, \mathbf{x}_\mathbf{s}, t) \end{aligned}$$

where the sub-indices, f and s, stand for "fast" and "slow," respectively.

Multi-rate Integration VI

We can generalize this procedure to systems of the form:

$$\begin{aligned}\dot{\mathbf{x}}_\mathbf{f}(t) &= \mathbf{f}_\mathbf{f}(\mathbf{x}_\mathbf{f}, \mathbf{x}_\mathbf{s}, t) \\ \dot{\mathbf{x}}_\mathbf{s}(t) &= \mathbf{f}_\mathbf{s}(\mathbf{x}_\mathbf{f}, \mathbf{x}_\mathbf{s}, t)\end{aligned}$$

where the sub-indices, f and s, stand for "fast" and "slow," respectively.

Then, the use of the *multi-rate version of Forward Euler with inlining* results in a set of difference equations of the form:

$$\begin{aligned}\mathbf{x}_\mathbf{f}(t_i + (j+1)\cdot h) &= \mathbf{x}_\mathbf{f}(t_i + j\cdot h) + h\cdot \mathbf{f}_\mathbf{f}(\mathbf{x}_\mathbf{f}(t_i + j\cdot h), \\ &\quad \mathbf{x}_\mathbf{s}(t_i + j\cdot h), t_i + j\cdot h) \\ \mathbf{x}_\mathbf{s}(t_i + k\cdot h) &= \mathbf{x}_\mathbf{s}(t_i) + k\cdot h\cdot \mathbf{f}_\mathbf{s}(\mathbf{x}_\mathbf{f}(t_i), \mathbf{x}_\mathbf{s}(t_i), t_i)\end{aligned}$$

where k is the (integer) ratio of the two step sizes, $j = 0 \ldots k-1$, and h is the step-size of the fast sub-system.

Multi-rate Integration VII

- The above equations do not specify, how $\mathbf{x_s}(t_i + j \cdot h)$ is being calculated, since the variables of the slow sub-system are not evaluated at the intermediate time instants.

Multi-rate Integration VII

- The above equations do not specify, how $\mathbf{x_s}(t_i + j \cdot h)$ is being calculated, since the variables of the slow sub-system are not evaluated at the intermediate time instants.
- We chose $\mathbf{x_s}(t_i + j \cdot h) = \mathbf{x_s}(t_i)$, i.e., we used the last calculated value.

Multi-rate Integration VII

- The above equations do not specify, how $\mathbf{x_s}(t_i + j \cdot h)$ is being calculated, since the variables of the slow sub-system are not evaluated at the intermediate time instants.
- We chose $\mathbf{x_s}(t_i + j \cdot h) = \mathbf{x_s}(t_i)$, i.e., we used the last calculated value.
- A more accurate solution would involve using some form of extrapolation technique.

Multi-rate Integration VII

- The above equations do not specify, how $\mathbf{x_s}(t_i + j \cdot h)$ is being calculated, since the variables of the slow sub-system are not evaluated at the intermediate time instants.
- We chose $\mathbf{x_s}(t_i + j \cdot h) = \mathbf{x_s}(t_i)$, i.e., we used the last calculated value.
- A more accurate solution would involve using some form of extrapolation technique.
- The problem is known as the *interfacing problem*. It is related to the way, in which the fast and slow sub-systems are interconnected with each other.

Multi-rate Integration VII

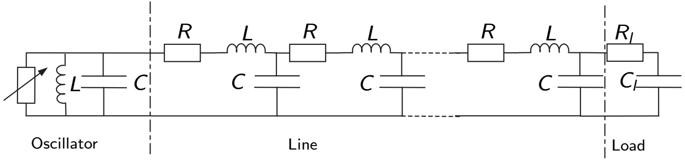
- The above equations do not specify, how $\mathbf{x_s}(t_i + j \cdot h)$ is being calculated, since the variables of the slow sub-system are not evaluated at the intermediate time instants.
- We chose $\mathbf{x_s}(t_i + j \cdot h) = \mathbf{x_s}(t_i)$, i.e., we used the last calculated value.
- A more accurate solution would involve using some form of extrapolation technique.
- The problem is known as the *interfacing problem*. It is related to the way, in which the fast and slow sub-systems are interconnected with each other.
- In our example, we used **FE**. Similar approaches have been reported in the literature based on the **AB2**, including some *improvements for parallel implementation*.

Inline Integration

Let us now try to simulate a slightly modified circuit:

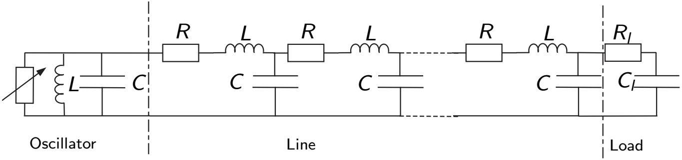
Inline Integration

Let us now try to simulate a slightly modified circuit:



Inline Integration

Let us now try to simulate a slightly modified circuit:



Let us assume that the load parameters are $R_l = 1\ \mathrm{k\Omega}$ and $C_l = 1\ \mathrm{nF}$.

Inline Integration II

- Since the load resistor is much bigger than the line resistors, the newly introduced term in the state-space model won't influence the dynamics of the transmission line significantly, and we can expect the load not to influence the behavior of the oscillator and the transmission line significantly.

Inline Integration II

- Since the load resistor is much bigger than the line resistors, the newly introduced term in the state-space model won't influence the dynamics of the transmission line significantly, and we can expect the load not to influence the behavior of the oscillator and the transmission line significantly.

- However, the added state equation introduces a fast pole. The position of this pole is approximately located at:

$$\lambda_l \approx -\frac{1}{R_l \cdot C_l} = -10^6 \text{ sec}^{-1}$$

Inline Integration II

- Since the load resistor is much bigger than the line resistors, the newly introduced term in the state-space model won't influence the dynamics of the transmission line significantly, and we can expect the load not to influence the behavior of the oscillator and the transmission line significantly.

- However, the added state equation introduces a fast pole. The position of this pole is approximately located at:

$$\lambda_l \approx -\frac{1}{R_l \cdot C_l} = -10^6 \ \mathrm{sec}^{-1}$$

- This means that we would have to reduce the step size by about a factor of 1000 with respect to the previous example, in order to obtain a numerically stable result.

Inline Integration II

- Since the load resistor is much bigger than the line resistors, the newly introduced term in the state-space model won't influence the dynamics of the transmission line significantly, and we can expect the load not to influence the behavior of the oscillator and the transmission line significantly.

- However, the added state equation introduces a fast pole. The position of this pole is approximately located at:

$$\lambda_l \approx -\frac{1}{R_l \cdot C_l} = -10^6 \text{ sec}^{-1}$$

- This means that we would have to reduce the step size by about a factor of 1000 with respect to the previous example, in order to obtain a numerically stable result.

- Unfortunately, such a solution is totally unacceptable in the context of a real-time simulation.

Inline Integration III

A first alternative might be to replace **FE** by **LIBE**. However, this is a system of order 13, and we may not have the luxury of inverting a 13×13 matrix at each step.

Inline Integration III

A first alternative might be to replace **FE** by **LIBE**. However, this is a system of order 13, and we may not have the luxury of inverting a 13×13 matrix at each step.

A second alternative might be to inline **BE** and apply the tearing method to the resulting set of difference equations.

Inline Integration III

A first alternative might be to replace **FE** by **LIBE**. However, this is a system of order 13, and we may not have the luxury of inverting a 13×13 matrix at each step.

A second alternative might be to inline **BE** and apply the tearing method to the resulting set of difference equations.

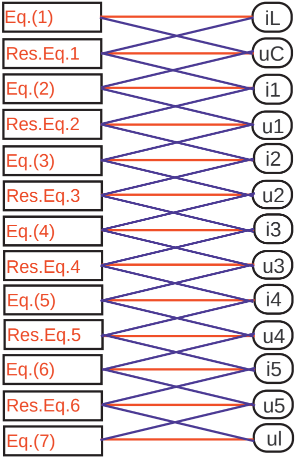
Let us rewrite the model using the inling approach.

Inline Integration IV

$$
\begin{aligned}
i_L &= \text{pre}(i_L) + \frac{h}{L} u_C \\
u_C &= \text{pre}(u_C) + \frac{h}{C}(u_C - k \cdot u_C^3 - i_L - i_1) \\
i_1 &= \text{pre}(i_1) + \frac{h}{L} u_C - \frac{Rh}{L} i_1 - \frac{h}{L} u_1 \\
u_1 &= \text{pre}(u_1) + \frac{h}{C} i_1 - \frac{h}{C} i_2 \\
i_2 &= \text{pre}(i_2) + \frac{h}{L} u_1 - \frac{Rh}{L} i_2 - \frac{h}{L} u_2 \\
u_2 &= \text{pre}(u_2) + \frac{h}{C} i_2 - \frac{h}{C} i_3 \\
&\vdots \\
i_n &= \text{pre}(i_n) + \frac{h}{L} u_{n-1} - \frac{Rh}{L} i_n - \frac{h}{L} u_n \\
u_n &= \text{pre}(u_n) + \frac{h}{C} i_n - \frac{h}{R_l \cdot C}(u_n - u_l) \\
u_l &= \text{pre}(u_l) + \frac{h}{R_l \cdot C_l}(u_n - u_l)
\end{aligned}
$$

Inline Integration IV

$$
\begin{aligned}
i_L &= \text{pre}(i_L) + \frac{h}{L} u_C \\
u_C &= \text{pre}(u_C) + \frac{h}{C}(u_C - k \cdot u_C^3 - i_L - i_1) \\
i_1 &= \text{pre}(i_1) + \frac{h}{L} u_C - \frac{Rh}{L} i_1 - \frac{h}{L} u_1 \\
u_1 &= \text{pre}(u_1) + \frac{h}{C} i_1 - \frac{h}{C} i_2 \\
i_2 &= \text{pre}(i_2) + \frac{h}{L} u_1 - \frac{Rh}{L} i_2 - \frac{h}{L} u_2 \\
u_2 &= \text{pre}(u_2) + \frac{h}{C} i_2 - \frac{h}{C} i_3 \\
&\vdots \\
i_n &= \text{pre}(i_n) + \frac{h}{L} u_{n-1} - \frac{Rh}{L} i_n - \frac{h}{L} u_n \\
u_n &= \text{pre}(u_n) + \frac{h}{C} i_n - \frac{h}{R_l \cdot C}(u_n - u_l) \\
u_l &= \text{pre}(u_l) + \frac{h}{R_l \cdot C_l}(u_n - u_l)
\end{aligned}
$$



Inline Integration V

Inlining did indeed help. We got away with six tearing variables.

Inline Integration V

Inlining did indeed help. We got away with six tearing variables.

- Instead of having to invert a 13×13 matrix in every step, we now must invert a 6×6 matrix.

Inline Integration V

Inlining did indeed help. We got away with six tearing variables.

- Instead of having to invert a 13×13 matrix in every step, we now must invert a 6×6 matrix.
- Since even the best linear sparse matrix solver grows at least quadratically with the size of the system in terms of its computational complexity, the savings were quite dramatic. The computations just got faster by about a factor of four.

Inline Integration V

Inlining did indeed help. We got away with six tearing variables.

- Instead of having to invert a 13×13 matrix in every step, we now must invert a 6×6 matrix.
- Since even the best linear sparse matrix solver grows at least quadratically with the size of the system in terms of its computational complexity, the savings were quite dramatic. The computations just got faster by about a factor of four.
- Although inline integration had been developed for general simulation problems, it turns out that this method has become a quite powerful ally in dealing with real-time simulation as well.

Inline Integration V

Inlining did indeed help. We got away with six tearing variables.

- Instead of having to invert a 13×13 matrix in every step, we now must invert a 6×6 matrix.
- Since even the best linear sparse matrix solver grows at least quadratically with the size of the system in terms of its computational complexity, the savings were quite dramatic. The computations just got faster by about a factor of four.
- Although inline integration had been developed for general simulation problems, it turns out that this method has become a quite powerful ally in dealing with real-time simulation as well.
- Unfortunately, we are now once again using an implicit algorithm, iterating on non-linear equations. This means that we have no guarantee that the iteration on the tearing variables will converge in time.

Inline Integration V

Inlining did indeed help. We got away with six tearing variables.

- Instead of having to invert a 13×13 matrix in every step, we now must invert a 6×6 matrix.
- Since even the best linear sparse matrix solver grows at least quadratically with the size of the system in terms of its computational complexity, the savings were quite dramatic. The computations just got faster by about a factor of four.
- Although inline integration had been developed for general simulation problems, it turns out that this method has become a quite powerful ally in dealing with real-time simulation as well.
- Unfortunately, we are now once again using an implicit algorithm, iterating on non-linear equations. This means that we have no guarantee that the iteration on the tearing variables will converge in time.
- If the Newton iteration converges after three steps, we may still be ahead of the game, but implicit algorithms are problematic for use in real-time simulation.

Mixed-mode Integration

Let us check whether we can further increase the simulation speed.

Mixed-mode Integration

Let us check whether we can further increase the simulation speed.

We notice that the fast and the slow sub-systems are *weakly coupled*.

Mixed-mode Integration

Let us check whether we can further increase the simulation speed.

We notice that the fast and the slow sub-systems are *weakly coupled*.

Hence it may be reasonable to inline the slow sub-system using **FE**, while inlining the fast sub-system using either **BE** or **LIBE**.

Mixed-mode Integration

Let us check whether we can further increase the simulation speed.

We notice that the fast and the slow sub-systems are *weakly coupled*.

Hence it may be reasonable to inline the slow sub-system using **FE**, while inlining the fast sub-system using either **BE** or **LIBE**.

In our example circuit, we inlined the equations once more, this time using the *explicit Forward Euler algorithm* everywhere except for the last equation, where we still used the *implicit Backward Euler method*.

Mixed-mode Integration II

$$
\begin{aligned}
i_L &= \mathrm{pre}(i_L) + \frac{h}{L}\mathrm{pre}(u_C) \\
u_C &= \mathrm{pre}(u_C) + \frac{h}{C} \cdot [\mathrm{pre}(u_C) - k \cdot \mathrm{pre}(u_C)^3 - \mathrm{pre}(i_L) - \mathrm{pre}(i_1)] \\
i_1 &= \mathrm{pre}(i_1) + \frac{h}{L}\mathrm{pre}(u_C) - \frac{Rh}{L}\mathrm{pre}(i_1) - \frac{h}{L}\mathrm{pre}(u_1) \\
u_1 &= \mathrm{pre}(u_1) + \frac{h}{C}\mathrm{pre}(i_1) - \frac{h}{C}\mathrm{pre}(i_2) \\
i_2 &= \mathrm{pre}(i_2) + \frac{h}{L}\mathrm{pre}(u_1) - \frac{Rh}{L}\mathrm{pre}(i_2) - \frac{h}{L}\mathrm{pre}(u_2) \\
u_2 &= \mathrm{pre}(u_2) + \frac{h}{C}\mathrm{pre}(i_2) - \frac{h}{C}\mathrm{pre}(i_3) \\
&\vdots \\
i_n &= \mathrm{pre}(i_n) + \frac{h}{L}\mathrm{pre}(u_{n-1}) - \frac{Rh}{L}\mathrm{pre}(i_n) - \frac{h}{L}\mathrm{pre}(u_n) \\
u_n &= \mathrm{pre}(u_n) + \frac{h}{C}\mathrm{pre}(i_n) - \frac{h}{R_l \cdot C}[\mathrm{pre}(u_n) - \mathrm{pre}(u_l)] \\
u_l &= \mathrm{pre}(u_l) + \frac{h}{R_l \cdot C_l}(u_n - u_l)
\end{aligned}
$$

Mixed-mode Integration II

$$
\begin{aligned}
i_L &= \mathrm{pre}(i_L) + \frac{h}{L}\mathrm{pre}(u_C) \\
u_C &= \mathrm{pre}(u_C) + \frac{h}{C} \cdot [\mathrm{pre}(u_C) - k \cdot \mathrm{pre}(u_C)^3 - \mathrm{pre}(i_L) - \mathrm{pre}(i_1)] \\
i_1 &= \mathrm{pre}(i_1) + \frac{h}{L}\mathrm{pre}(u_C) - \frac{Rh}{L}\mathrm{pre}(i_1) - \frac{h}{L}\mathrm{pre}(u_1) \\
u_1 &= \mathrm{pre}(u_1) + \frac{h}{C}\mathrm{pre}(i_1) - \frac{h}{C}\mathrm{pre}(i_2) \\
i_2 &= \mathrm{pre}(i_2) + \frac{h}{L}\mathrm{pre}(u_1) - \frac{Rh}{L}\mathrm{pre}(i_2) - \frac{h}{L}\mathrm{pre}(u_2) \\
u_2 &= \mathrm{pre}(u_2) + \frac{h}{C}\mathrm{pre}(i_2) - \frac{h}{C}\mathrm{pre}(i_3) \\
&\vdots \\
i_n &= \mathrm{pre}(i_n) + \frac{h}{L}\mathrm{pre}(u_{n-1}) - \frac{Rh}{L}\mathrm{pre}(i_n) - \frac{h}{L}\mathrm{pre}(u_n) \\
u_n &= \mathrm{pre}(u_n) + \frac{h}{C}\mathrm{pre}(i_n) - \frac{h}{R_l \cdot C}[\mathrm{pre}(u_n) - \mathrm{pre}(u_l)] \\
u_l &= \mathrm{pre}(u_l) + \frac{h}{R_l \cdot C_l}(u_n - u_l)
\end{aligned}
$$

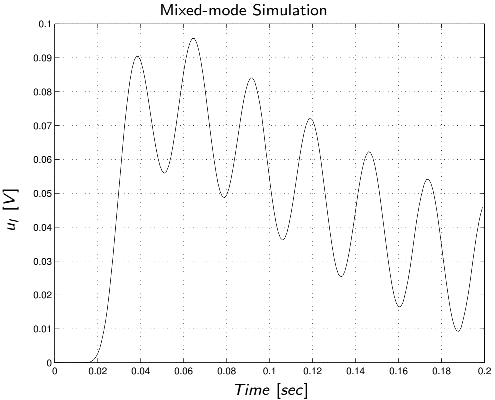
- All equations are now explicit, except for the very last equation, which is implicit in the variable u_l, but can be solved symbolically for u_l.

Mixed-mode Integration II

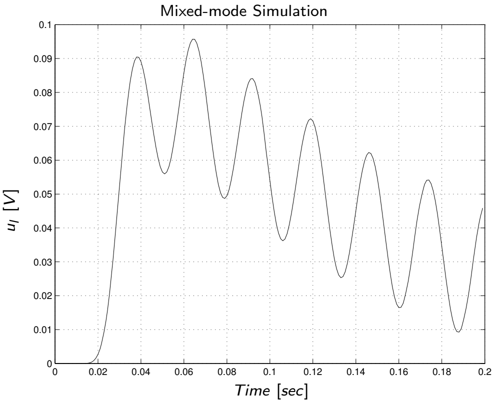
$$
\begin{aligned}
i_L &= \text{pre}(i_L) + \frac{h}{L}\text{pre}(u_C) \\
u_C &= \text{pre}(u_C) + \frac{h}{C}\cdot[\text{pre}(u_C) - k\cdot\text{pre}(u_C)^3 - \text{pre}(i_L) - \text{pre}(i_1)] \\
i_1 &= \text{pre}(i_1) + \frac{h}{L}\text{pre}(u_C) - \frac{Rh}{L}\text{pre}(i_1) - \frac{h}{L}\text{pre}(u_1) \\
u_1 &= \text{pre}(u_1) + \frac{h}{C}\text{pre}(i_1) - \frac{h}{C}\text{pre}(i_2) \\
i_2 &= \text{pre}(i_2) + \frac{h}{L}\text{pre}(u_1) - \frac{Rh}{L}\text{pre}(i_2) - \frac{h}{L}\text{pre}(u_2) \\
u_2 &= \text{pre}(u_2) + \frac{h}{C}\text{pre}(i_2) - \frac{h}{C}\text{pre}(i_3) \\
&\vdots \\
i_n &= \text{pre}(i_n) + \frac{h}{L}\text{pre}(u_{n-1}) - \frac{Rh}{L}\text{pre}(i_n) - \frac{h}{L}\text{pre}(u_n) \\
u_n &= \text{pre}(u_n) + \frac{h}{C}\text{pre}(i_n) - \frac{h}{R_l\cdot C}[\text{pre}(u_n) - \text{pre}(u_l)] \\
u_l &= \text{pre}(u_l) + \frac{h}{R_l\cdot C_l}(u_n - u_l)
\end{aligned}
$$

- All equations are now explicit, except for the very last equation, which is implicit in the variable u_l, but can be solved symbolically for u_l.
- We simulated the system using the same approach as before, i.e., we applied a step size of 10^{-4} seconds to the two oscillator equation, whereas we used a step size of 10^{-3} seconds to all other equations, including the implicit load equation.

Mixed-mode Integration III



Mixed-mode Integration III



The simulation executed now very fast, while the simulation results are adequately accurate.

Mixed-mode Integration IV

In more general terms, the *Backward-Forward Euler Mixed-Mode integration scheme* can be written as:

$$\begin{aligned}
\mathbf{x_s}(t_{k+1}) &= \mathbf{x_s}(t_k) + h \cdot \mathbf{f_s}(\mathbf{x_f}(t_k), \mathbf{x_s}(t_k), t_k) \\
\mathbf{x_f}(t_{k+1}) &= \mathbf{x_f}(t_k) + h \cdot \mathbf{f_f}(\mathbf{x_f}(t_{k+1}), \mathbf{x_s}(t_{k+1}), t_{k+1})
\end{aligned}$$

Mixed-mode Integration IV

In more general terms, the *Backward-Forward Euler Mixed-Mode integration scheme* can be written as:

$$\begin{aligned}\mathbf{x_s}(t_{k+1}) &= \mathbf{x_s}(t_k) + h \cdot \mathbf{f_s}(\mathbf{x_f}(t_k), \mathbf{x_s}(t_k), t_k)\\ \mathbf{x_f}(t_{k+1}) &= \mathbf{x_f}(t_k) + h \cdot \mathbf{f_f}(\mathbf{x_f}(t_{k+1}), \mathbf{x_s}(t_{k+1}), t_{k+1})\end{aligned}$$

- The algorithm starts by computing explicitly the value of $\mathbf{x_s}(t_{k+1})$.

Mixed-mode Integration IV

In more general terms, the *Backward-Forward Euler Mixed-Mode integration scheme* can be written as:

$$\begin{aligned}\mathbf{x_s}(t_{k+1}) &= \mathbf{x_s}(t_k) + h \cdot \mathbf{f_s}(\mathbf{x_f}(t_k), \mathbf{x_s}(t_k), t_k) \\ \mathbf{x_f}(t_{k+1}) &= \mathbf{x_f}(t_k) + h \cdot \mathbf{f_f}(\mathbf{x_f}(t_{k+1}), \mathbf{x_s}(t_{k+1}), t_{k+1})\end{aligned}$$

- The algorithm starts by computing explicitly the value of $\mathbf{x_s}(t_{k+1})$.
- It then uses this value to evaluate $\mathbf{x_f}(t_{k+1})$ either implicitly or in a semi-implicit fashion.

Discontinuous Systems

We still need to discuss, how we can perform real-time simulations of discontinuous systems.

Discontinuous Systems

We still need to discuss, how we can perform real-time simulations of discontinuous systems.

We need to distinguish between external and internal events.

Discontinuous Systems

We still need to discuss, how we can perform real-time simulations of discontinuous systems.

We need to distinguish between external and internal events.

- *External events* are time events that are received from outside the simulation. They are used by either humans or control agents to interfere with the real-time simulation, e.g. for changing a parameter value.

Discontinuous Systems

We still need to discuss, how we can perform real-time simulations of discontinuous systems.

We need to distinguish between external and internal events.

- *External events* are time events that are received from outside the simulation. They are used by either humans or control agents to interfere with the real-time simulation, e.g. for changing a parameter value.
- External events are not time-critical, as they are always *asynchronous* to the simulation. Consequently, a small delay is acceptable, and therefore, external events are always delayed until the end of the current integration step and are handled during the waiting period, i.e., before the next synchronization impulse arrives from the real-time clock.

Discontinuous Systems II

- *Internal events* are events caused by the real-time simulation itself. They are thus *synchronous* to the simulation, and consequently, can be time-critical. They should be handled at the correct time.

Discontinuous Systems II

- *Internal events* are events caused by the real-time simulation itself. They are thus *synchronous* to the simulation, and consequently, can be time-critical. They should be handled at the correct time.

- In real-time simulation, there is not much difference between an *internal time event* and an *internal state event*, as we shall see.

Discontinuous Systems II

- *Internal events* are events caused by the real-time simulation itself. They are thus *synchronous* to the simulation, and consequently, can be time-critical. They should be handled at the correct time.
- In real-time simulation, there is not much difference between an *internal time event* and an *internal state event*, as we shall see.

State-event handling in real-time simulation is simplified, when comparing it to the techniques introduced earlier, by two factors:

Discontinuous Systems II

- *Internal events* are events caused by the real-time simulation itself. They are thus *synchronous* to the simulation, and consequently, can be time-critical. They should be handled at the correct time.

- In real-time simulation, there is not much difference between an *internal time event* and an *internal state event*, as we shall see.

State-event handling in real-time simulation is simplified, when comparing it to the techniques introduced earlier, by two factors:

1. As we are using low-order integration techniques, we can also use low-order event localization algorithms.

Discontinuous Systems II

- *Internal events* are events caused by the real-time simulation itself. They are thus *synchronous* to the simulation, and consequently, can be time-critical. They should be handled at the correct time.
- In real-time simulation, there is not much difference between an *internal time event* and an *internal state event*, as we shall see.

State-event handling in real-time simulation is simplified, when comparing it to the techniques introduced earlier, by two factors:

1. As we are using low-order integration techniques, we can also use low-order event localization algorithms.
2. Since we use much smaller step sizes, the precise localization of state events becomes less critical, and there shouldn't occur as often multiple state events within a single integration step.

Discontinuous Systems III

Since we must control the total amount of computations performed within an integration step, *iterative techniques* for localizing state events are out. We must rely on *interpolation* alone.

Discontinuous Systems III

Since we must control the total amount of computations performed within an integration step, *iterative techniques* for localizing state events are out. We must rely on *interpolation* alone.

Yet, as we are using low-order integration techniques, the former iterative algorithms can now be employed as interpolators.

Discontinuous Systems III

Since we must control the total amount of computations performed within an integration step, *iterative techniques* for localizing state events are out. We must rely on *interpolation* alone.

Yet, as we are using low-order integration techniques, the former iterative algorithms can now be employed as interpolators.

- If we integrate by inlining a first-order accurate algorithm, i.e., either **FE**, **BE**, or **LIBE**, we can use a single step of *Regula Falsi* to locate the event as accurately as we can hope to accomplish with such a crude integration algorithm.

Discontinuous Systems III

Since we must control the total amount of computations performed within an integration step, *iterative techniques* for localizing state events are out. We must rely on *interpolation* alone.

Yet, as we are using low-order integration techniques, the former iterative algorithms can now be employed as interpolators.

- If we integrate by inlining a first-order accurate algorithm, i.e., either **FE**, **BE**, or **LIBE**, we can use a single step of *Regula Falsi* to locate the event as accurately as we can hope to accomplish with such a crude integration algorithm.
- If we decide to inline the *Radau-IIA(3) algorithm*, a single step of *cubic interpolation* will localize the discontinuity as accurately as can be done using such an integration method.

Discontinuous Systems III

Since we must control the total amount of computations performed within an integration step, *iterative techniques* for localizing state events are out. We must rely on *interpolation* alone.

Yet, as we are using low-order integration techniques, the former iterative algorithms can now be employed as interpolators.

- If we integrate by inlining a first-order accurate algorithm, i.e., either **FE**, **BE**, or **LIBE**, we can use a single step of *Regula Falsi* to locate the event as accurately as we can hope to accomplish with such a crude integration algorithm.
- If we decide to inline the *Radau-IIA(3) algorithm*, a single step of *cubic interpolation* will localize the discontinuity as accurately as can be done using such an integration method.

Of course, it may be possible to reduce the residual on the zero-crossing function further by iteration, but this does not necessarily imply that we would thereby locate the event more accurately, as already the previous integration steps are contaminated by numerical errors.

Discontinuous Systems IV

Let us discuss, how event handling may proceed under real-time constraints.

Discontinuous Systems IV

Let us discuss, how event handling may proceed under real-time constraints.

- We perform a regular integration step, advancing the simulation from time t_n to time t_{n+1}.

Discontinuous Systems IV

Let us discuss, how event handling may proceed under real-time constraints.

- We perform a regular integration step, advancing the simulation from time t_n to time t_{n+1}.
- At the end of the step, we discover that a zero crossing has taken place.

Discontinuous Systems IV

Let us discuss, how event handling may proceed under real-time constraints.

- We perform a regular integration step, advancing the simulation from time t_n to time t_{n+1}.
- At the end of the step, we discover that a zero crossing has taken place.
- We interpolate to the next event time, $t_{\text{next}} \in [t_n, t_{n+1}]$, using a formula of the same order of accuracy as that of the integration method in use.

Discontinuous Systems IV

Let us discuss, how event handling may proceed under real-time constraints.

- We perform a regular integration step, advancing the simulation from time t_n to time t_{n+1}.
- At the end of the step, we discover that a zero crossing has taken place.
- We interpolate to the next event time, $t_{\text{next}} \in [t_n, t_{n+1}]$, using a formula of the same order of accuracy as that of the integration method in use.
- We repeat the last integration step to advance the entire state vector from time t_n to time t_{next}.

Discontinuous Systems IV

Let us discuss, how event handling may proceed under real-time constraints.

- We perform a regular integration step, advancing the simulation from time t_n to time t_{n+1}.
- At the end of the step, we discover that a zero crossing has taken place.
- We interpolate to the next event time, $t_{\mathrm{next}} \in [t_n, t_{n+1}]$, using a formula of the same order of accuracy as that of the integration method in use.
- We repeat the last integration step to advance the entire state vector from time t_n to time t_{next}.
- We then perform the actions associated with the event, and compute a new consistent initial state.

Discontinuous Systems IV

Let us discuss, how event handling may proceed under real-time constraints.

- We perform a regular integration step, advancing the simulation from time t_n to time t_{n+1}.
- At the end of the step, we discover that a zero crossing has taken place.
- We interpolate to the next event time, $t_{\text{next}} \in [t_n, t_{n+1}]$, using a formula of the same order of accuracy as that of the integration method in use.
- We repeat the last integration step to advance the entire state vector from time t_n to time t_{next}.
- We then perform the actions associated with the event, and compute a new consistent initial state.
- Starting from the new initial state, we perform a partial step advancing the state vector from time t_{next} to time t_{n+1}.

Discontinuous Systems V

- As no iteration takes place, the amount of work, i.e., the total number of floating-point operations needed, can be estimated accurately.

Discontinuous Systems V

- As no iteration takes place, the amount of work, i.e., the total number of floating-point operations needed, can be estimated accurately.

- Assuming that only *one state event is allowed to occur within a single integration step*, we can thus calculate, how much extra time we need to allot, in order to handle single state events within an integration step adequately.

Discontinuous Systems V

- As no iteration takes place, the amount of work, i.e., the total number of floating-point operations needed, can be estimated accurately.
- Assuming that only *one state event is allowed to occur within a single integration step*, we can thus calculate, how much extra time we need to allot, in order to handle single state events within an integration step adequately.
- We perform three integration steps instead of only one, and we have to accommodate the additional computations needed to process the event actions themselves.

Discontinuous Systems V

- As no iteration takes place, the amount of work, i.e., the total number of floating-point operations needed, can be estimated accurately.
- Assuming that only *one state event is allowed to occur within a single integration step*, we can thus calculate, how much extra time we need to allot, in order to handle single state events within an integration step adequately.
- We perform three integration steps instead of only one, and we have to accommodate the additional computations needed to process the event actions themselves.
- Thus, the total effort may grow by about a factor of four.

Discontinuous Systems V

- As no iteration takes place, the amount of work, i.e., the total number of floating-point operations needed, can be estimated accurately.
- Assuming that only *one state event is allowed to occur within a single integration step*, we can thus calculate, how much extra time we need to allot, in order to handle single state events within an integration step adequately.
- We perform three integration steps instead of only one, and we have to accommodate the additional computations needed to process the event actions themselves.
- Thus, the total effort may grow by about a factor of four.
- **For this reason, the allowed resource utilization for regular integration steps needs to drop from about 80% to about 20%.**

Discontinuous Systems V

- As no iteration takes place, the amount of work, i.e., the total number of floating-point operations needed, can be estimated accurately.
- Assuming that only *one state event is allowed to occur within a single integration step*, we can thus calculate, how much extra time we need to allot, in order to handle single state events within an integration step adequately.
- We perform three integration steps instead of only one, and we have to accommodate the additional computations needed to process the event actions themselves.
- Thus, the total effort may grow by about a factor of four.
- **For this reason, the allowed resource utilization for regular integration steps needs to drop from about 80% to about 20%.**

Internal time events are handled essentially in the same manner, except that their time of occurrence is known in advance, i.e., we only need two partial integration steps within the time allotted for one step instead of three.

Simulation Architecture

We haven't discussed yet, how the simulation engine is physically connected to the hardware.

Simulation Architecture

We haven't discussed yet, how the simulation engine is physically connected to the hardware.

Although it is principally possible to connect directly the output signals of the *sensor units* with the input of the *A/D-converters*, which form part of the simulation engine, and the outputs of the *D/A-converters*, also integrated with the simulation engine, with the input signals of the *actuator units*, this is hardly ever done in today's world.

Simulation Architecture

We haven't discussed yet, how the simulation engine is physically connected to the hardware.

Although it is principally possible to connect directly the output signals of the *sensor units* with the input of the *A/D-converters*, which form part of the simulation engine, and the outputs of the *D/A-converters*, also integrated with the simulation engine, with the input signals of the *actuator units*, this is hardly ever done in today's world.

Instead, commercial converters have their own computer chips built in, that perform the necessary computations and store the digital signals in *mailboxes*.

Simulation Architecture

We haven't discussed yet, how the simulation engine is physically connected to the hardware.

Although it is principally possible to connect directly the output signals of the *sensor units* with the input of the *A/D-converters*, which form part of the simulation engine, and the outputs of the *D/A-converters*, also integrated with the simulation engine, with the input signals of the *actuator units*, this is hardly ever done in today's world.

Instead, commercial converters have their own computer chips built in, that perform the necessary computations and store the digital signals in *mailboxes*.

Thus, an A/D-converter is really a converter together with a built-in *zero-order hold (ZOH)* unit. Once the analog signal has been converted, it is available for whichever process needs it, until it is overridden by the next *sample-and-hold (S/H)* cycle.

Simulation Architecture

We haven't discussed yet, how the simulation engine is physically connected to the hardware.

Although it is principally possible to connect directly the output signals of the *sensor units* with the input of the *A/D-converters*, which form part of the simulation engine, and the outputs of the *D/A-converters*, also integrated with the simulation engine, with the input signals of the *actuator units*, this is hardly ever done in today's world.

Instead, commercial converters have their own computer chips built in, that perform the necessary computations and store the digital signals in *mailboxes*.

Thus, an A/D-converter is really a converter together with a built-in *zero-order hold (ZOH)* unit. Once the analog signal has been converted, it is available for whichever process needs it, until it is overridden by the next *sample-and-hold (S/H)* cycle.

A D/A-converter doesn't take its data from the simulation directly, but instead, takes it out of its own mailbox.

Simulation Architecture

We haven't discussed yet, how the simulation engine is physically connected to the hardware.

Although it is principally possible to connect directly the output signals of the *sensor units* with the input of the *A/D-converters*, which form part of the simulation engine, and the outputs of the *D/A-converters*, also integrated with the simulation engine, with the input signals of the *actuator units*, this is hardly ever done in today's world.

Instead, commercial converters have their own computer chips built in, that perform the necessary computations and store the digital signals in *mailboxes*.

Thus, an A/D-converter is really a converter together with a built-in *zero-order hold (ZOH)* unit. Once the analog signal has been converted, it is available for whichever process needs it, until it is overridden by the next *sample-and-hold (S/H)* cycle.

A D/A-converter doesn't take its data from the simulation directly, but instead, takes it out of its own mailbox.

Even the sensor and actuator units contain their own hardware-built sample-and-hold equipment.

Simulation Architecture II

Handshaking mechanisms are needed to prevent the simulator from replacing the data in the mailbox of the D/A-converter, while the converter tries to read out the data from its own mailbox.

Simulation Architecture II

Handshaking mechanisms are needed to prevent the simulator from replacing the data in the mailbox of the D/A-converter, while the converter tries to read out the data from its own mailbox.

Similarly, handshaking mechanisms are also needed to prevent the simulator from reading the data from the mailbox of the A/D-converter, while these data are in the process of being updated by the converter.

Simulation Architecture II

Handshaking mechanisms are needed to prevent the simulator from replacing the data in the mailbox of the D/A-converter, while the converter tries to read out the data from its own mailbox.

Similarly, handshaking mechanisms are also needed to prevent the simulator from reading the data from the mailbox of the A/D-converter, while these data are in the process of being updated by the converter.

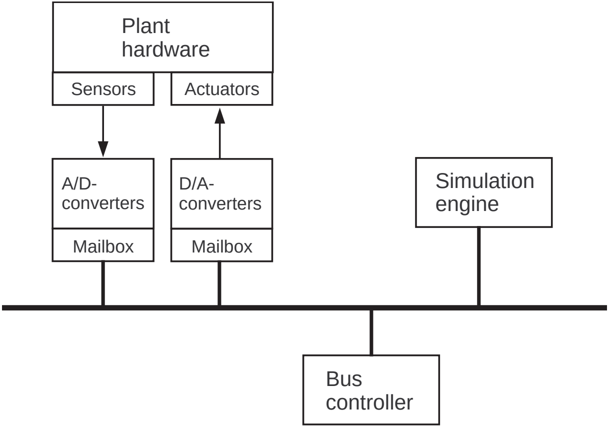
This is usually accomplished by making read and write operations out of and into the mailboxes *non-interruptible*.

Simulation Architecture III

A possible physical configuration of a *hardware-in-the-loop (HIL)* simulation is shown below:

Simulation Architecture III

A possible physical configuration of a *hardware-in-the-loop (HIL)* simulation is shown below:



Simulation Architecture IV

- Protocols have been designed to ensure that these handshaking mechanisms always work correctly. To this end, the *High-Level Architecture (HLA)* standard was created in the U.S., whereas Europe developed its own standard with *CORBA*.

Simulation Architecture IV

- Protocols have been designed to ensure that these handshaking mechanisms always work correctly. To this end, the *High-Level Architecture (HLA)* standard was created in the U.S., whereas Europe developed its own standard with *CORBA*.

- Consequently, the first figure of this presentation needs to be updated. The time intervals needed for the A/D-conversions and D/A-conversions are no longer part of the computational load associated with advancing the simulation by one time step, as these activities are performed in parallel by separate computational units. Instead, we must include the time needed for the read and write requests from and to the mailboxes across the architecture.

Simulation Architecture IV

- Protocols have been designed to ensure that these handshaking mechanisms always work correctly. To this end, the *High-Level Architecture (HLA)* standard was created in the U.S., whereas Europe developed its own standard with *CORBA*.

- Consequently, the first figure of this presentation needs to be updated. The time intervals needed for the A/D-conversions and D/A-conversions are no longer part of the computational load associated with advancing the simulation by one time step, as these activities are performed in parallel by separate computational units. Instead, we must include the time needed for the read and write requests from and to the mailboxes across the architecture.

- Since the total time needed for computing all activities associated with a single integration step must be known, both HLA and CORBA offer mechanisms for specifying the *maximum allowed latency* in answering requests for information transfer across the architecture using the established communication channels and protocols.

Overruns

Overruns are defined as situations, where, in spite of our best efforts, the simulation engine is unable to perform all of the required computations in time to advance its state to the next clock time, before the real-time clock interrupt is received.

Overruns

Overruns are defined as situations, where, in spite of our best efforts, the simulation engine is unable to perform all of the required computations in time to advance its state to the next clock time, before the real-time clock interrupt is received.

This may happen, because it cannot be guaranteed that no more than one *state event* will ever occur within a single integration step.

Overruns

Overruns are defined as situations, where, in spite of our best efforts, the simulation engine is unable to perform all of the required computations in time to advance its state to the next clock time, before the real-time clock interrupt is received.

This may happen, because it cannot be guaranteed that no more than one *state event* will ever occur within a single integration step.

As all events must be processed, it can happen that the simulation falls behind.

Overruns

Overruns are defined as situations, where, in spite of our best efforts, the simulation engine is unable to perform all of the required computations in time to advance its state to the next clock time, before the real-time clock interrupt is received.

This may happen, because it cannot be guaranteed that no more than one *state event* will ever occur within a single integration step.

As all events must be processed, it can happen that the simulation falls behind.

Most real-time simulations specify the *maximum percentage of overruns* as e.g. 1% or 2%.

Overruns II

What happens, when the simulation falls behind?

Overruns II

What happens, when the simulation falls behind?

Thanks to the buffers implemented in the form of the mailboxes, the *hardware* will hardly notice it. It simply receives the same actuator values for a second time in a row.

Overruns II

What happens, when the simulation falls behind?

Thanks to the buffers implemented in the form of the mailboxes, the *hardware* will hardly notice it. It simply receives the same actuator values for a second time in a row.

For the *simulation software*, the situation may be worse, because it may need to know, what time it is.

Overruns II

What happens, when the simulation falls behind?

Thanks to the buffers implemented in the form of the mailboxes, the *hardware* will hardly notice it. It simply receives the same actuator values for a second time in a row.

For the *simulation software*, the situation may be worse, because it may need to know, what time it is.

Thus, the following procedure is recommended in the case of an overrun.

Overruns II

What happens, when the simulation falls behind?

Thanks to the buffers implemented in the form of the mailboxes, the *hardware* will hardly notice it. It simply receives the same actuator values for a second time in a row.

For the *simulation software*, the situation may be worse, because it may need to know, what time it is.

Thus, the following procedure is recommended in the case of an overrun.

- **If the next real-time interrupt arrives, before the computations have been completed, the subsequent integration step is doubled in length to catch up with real time.**

Overruns II

What happens, when the simulation falls behind?

Thanks to the buffers implemented in the form of the mailboxes, the *hardware* will hardly notice it. It simply receives the same actuator values for a second time in a row.

For the *simulation software*, the situation may be worse, because it may need to know, what time it is.

Thus, the following procedure is recommended in the case of an overrun.

- **If the next real-time interrupt arrives, before the computations have been completed, the subsequent integration step is doubled in length to catch up with real time.**

In this way, we allow an integration step to be computed less accurately once in a while in order to stay synchronous with the real-time clock.

Conclusions

- In this presentation, we looked at the special requirements of *real-time simulation*.

Conclusions

- In this presentation, we looked at the special requirements of *real-time simulation*.
- As real-time simulation is not conceptually different from general-purpose simulations, many of the observations made were rather practical, and not highly mathematical.

Conclusions

- In this presentation, we looked at the special requirements of *real-time simulation*.
- As real-time simulation is not conceptually different from general-purpose simulations, many of the observations made were rather practical, and not highly mathematical.
- Consequently, this presentation contains much more text and much less formulae than any of the previous presentations.

Conclusions

- In this presentation, we looked at the special requirements of *real-time simulation*.
- As real-time simulation is not conceptually different from general-purpose simulations, many of the observations made were rather practical, and not highly mathematical.
- Consequently, this presentation contains much more text and much less formulae than any of the previous presentations.
- Yet, real-time simulation is a very important branch of simulation, and consequently, its special demands require often solutions that we wouldn't ever consider apart from a real-time simulation task.

Conclusions II

- We introduced a new (derived) class of integration algorithms, the *linearly implicit integration algorithms*. These can be designed as variants of practically all implicit integration schemes, although we concentrated on one algorithm only, the *linearly implicit Backward Euler (LIBE)* algorithm. Whereas such algorithms could also be used for general-purpose simulation, this is hardly ever done.

Conclusions II

- We introduced a new (derived) class of integration algorithms, the *linearly implicit integration algorithms*. These can be designed as variants of practically all implicit integration schemes, although we concentrated on one algorithm only, the *linearly implicit Backward Euler (LIBE)* algorithm. Whereas such algorithms could also be used for general-purpose simulation, this is hardly ever done.

- We then looked at *multi-rate integration schemes*. They have a role to play in real-time simulations of systems with clearly separate groups of eigenvalues, such as in the real-time simulation of physical systems including multiple energy domains. Once again, multi-rate integration schemes are hardly ever used outside the world of real-time simulation.

Conclusions II

- We introduced a new (derived) class of integration algorithms, the *linearly implicit integration algorithms*. These can be designed as variants of practically all implicit integration schemes, although we concentrated on one algorithm only, the *linearly implicit Backward Euler (LIBE)* algorithm. Whereas such algorithms could also be used for general-purpose simulation, this is hardly ever done.

- We then looked at *multi-rate integration schemes*. They have a role to play in real-time simulations of systems with clearly separate groups of eigenvalues, such as in the real-time simulation of physical systems including multiple energy domains. Once again, multi-rate integration schemes are hardly ever used outside the world of real-time simulation.

- We then showed that *inline integration methods* and *mixed-mode integration schemes* can be very useful for speeding up real-time simulations.

Conclusions II

- We introduced a new (derived) class of integration algorithms, the *linearly implicit integration algorithms*. These can be designed as variants of practically all implicit integration schemes, although we concentrated on one algorithm only, the *linearly implicit Backward Euler (LIBE)* algorithm. Whereas such algorithms could also be used for general-purpose simulation, this is hardly ever done.

- We then looked at *multi-rate integration schemes*. They have a role to play in real-time simulations of systems with clearly separate groups of eigenvalues, such as in the real-time simulation of physical systems including multiple energy domains. Once again, multi-rate integration schemes are hardly ever used outside the world of real-time simulation.

- We then showed that *inline integration methods* and *mixed-mode integration schemes* can be very useful for speeding up real-time simulations.

- The presentation ended with some general remarks about *real-time simulation architectures*.

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. and M. Krebs (2007), "Analysis and Simulation of Variable Structure Systems Using Bond Graphs and Inline Integration," *Proc. ICBGM07, 8^{th} SCS Intl. Conf. on Bond Graph Modeling and Simulation*, San Diego, California, pp. 29-34.

3. Krebs, Matthias (1997), *Modeling of Conditional Index Changes*, MS Thesis, Dept. of Electr. & Comp. Engr., University of Arizona, Tucson, AZ.