Numerical Simulation of Dynamic Systems XXI

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

May 7, 2013

Introduction

All of the simulation methods that we encountered until now operate, in one form or another, on *polynomial extrapolations* using *Taylor series expansions*.

A fundamental property of *polynomials* is that they *don't exhibit discontinuities*.

Consequently, a model that contains discontinuities cannot be simulated across these discontinuities using polynomial extrapolations.

A large majority of *engineering systems exhibit many discontinuities* that need to be included in their models.

The numerical integration methods for dynamic systems that we developed until now can thus not be used for the simulation of *hybrid systems*. We shall need something better.

Abusing the Step-size Control

What happens if we simply close our eyes and integrate across a discontinuity of the model using any one of the numerical ODE solvers with step-size control introduced earlier?

The algorithm doesn't know that there exists a discontinuity. What it does notice is a *rapid change in the trajectory*.

The algorithm concludes that a *new eigenvalue appeared* far out to the left in the complex plane.

Consequently, the algorithm *reduces the step size* in order to capture this eigenvalue in its *accuracy domain*.

However, the new "eigenvalue" is a joker. It doesn't allow itself to be captured. Irrespective of how much the step size is being reduced, the eigenvalue remains outside the accuracy domain.

Abusing the Step-size Control II

When does the iteration on the rejected step end?

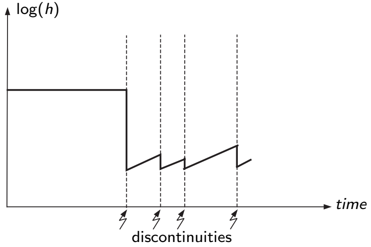
The step-size reduction continues until one of two things happens:

- The step size is reduced to the *smallest allowed value* specified in the step-size control algorithm.
- The step-size control algorithm decides that the *integration accuracy is acceptable*. This will eventually happen, as by reducing the step size, the non-linear terms in the Taylor series expansion lose their importance. With a sufficiently small step size, *every algorithm behaves like either Forward or Backward Euler*.

Once the discontinuity lies in the past, the evasive eigenvalue disappears as miraculously as it had shown up before.

The step-size control algorithm slowly increases the step size again, until it reaches its optimal value . . . or until it encounters the next discontinuity, whichever happens first.

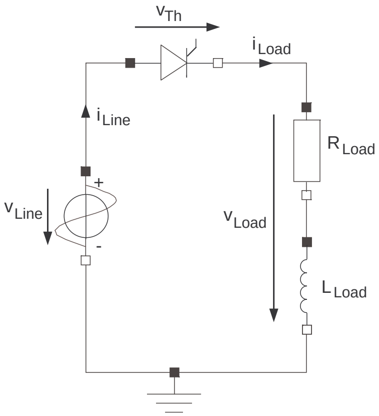
Abusing the Step-size Control III



Abuse of the step-size control algorithm for the *localization of discontinuities* often works quite well, and it is for this reason that many of the more primitive environments for the modeling and simulation of dynamic systems don't offer any special provisions for handling discontinuities in the model.

However, this approach is neither efficient nor robust. Sometimes, it fails miserably.

Speed Control in Train Engines



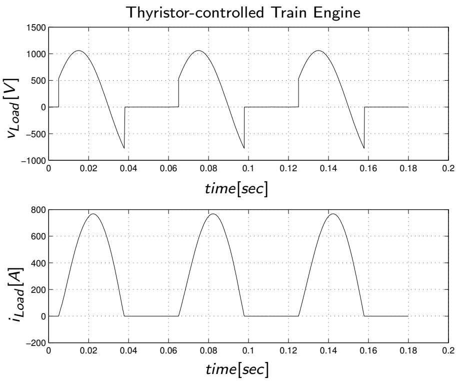
Swiss trains are running on *AC voltage* at $16\frac{2}{3}\,\mathrm{Hz}$.

Electrical thyristor (controlled rectifier) circuits are being used for *speed control*.

The most simple speed control circuit is shown to the left. The resistor with an inductor in series represents the load (the train). The thyristor blocks negative current (operating as a diode) and also blocks an additional part of every period.

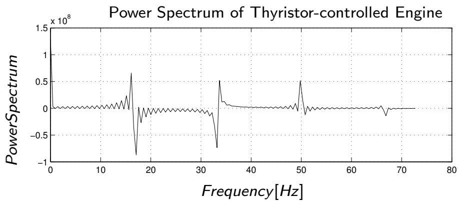
The percentage of the period that is not being blocked is controlled by the *firing angle* of the thyristor.

Speed Control in Train Engines II



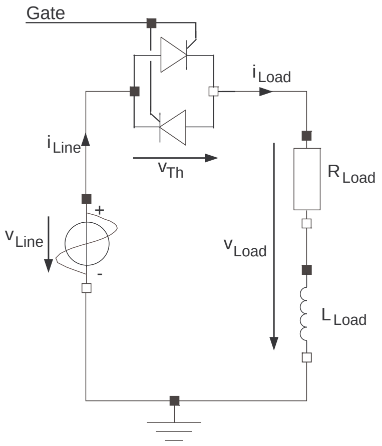
- The simulation trajectories exhibit discontinuities.
- The electric power $P = v_{\text{Load}} \cdot i_{\text{Load}}$ depends on the *firing angle*.
- For this simulation, we used a firing angle of 30^{o}. The speed of the train gets reduced with larger firing angles.

Speed Control in Train Engines III



- There is much power contained in the *third harmonic* at $50\mathrm{Hz}$.
- **In the past, when trains with three thyristor-controlled locomotives slowly climbed up the St. Gotthard mountain, the electric counters of the houses near to the rails were reset to zero, to everyone's content ... except for the local power company of the Canton of Uri.**

Speed Control in Train Engines IV



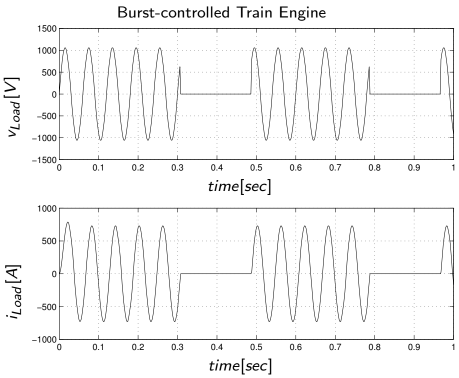
During some years, another type of thyristor-based speed control circuit was used in the trains of the line Zurich-Meilen-Rapperswil.

The thyristors were controlled in such a way that a number of periods were let through, whereas others were blocked.

This control strategy is called *burst control strategy*.

The trains made use of *packets of eight periods*.

Speed Control in Train Engines V



- The simulation trajectories exhibit less serious discontinuities than those observed using the previous thyristor-controlled circuit.
- Consequently, this control strategy doesn't interfere with the electric grid operated at 50Hz.
- However, this type of control only offers eight distinct velocities.

- **When the trains departed from the stations, the passengers noticed the abrupt velocity changes and received a free massage of their stomach muscles.**

Speed Control in Train Engines VI

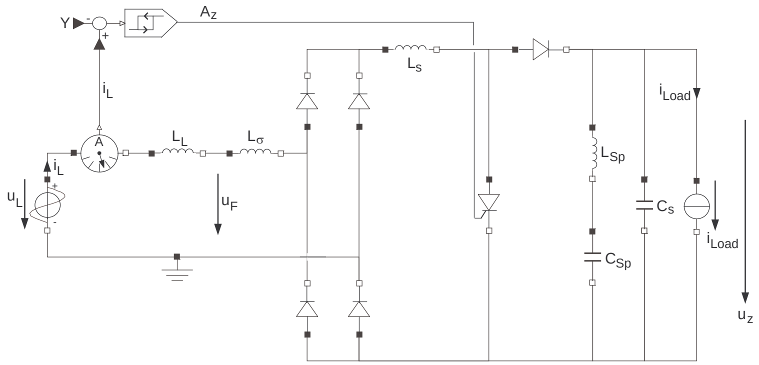
- The signal here is not $16\frac{2}{3}\mathrm{Hz}$-periodic, but rather $2\frac{1}{12}\mathrm{Hz}$-periodic.

- There is power in the higher harmonics at $4\frac{1}{6}\mathrm{Hz}$ and at $6\frac{1}{4}\mathrm{Hz}$, but nothing at $50\mathrm{Hz}$. Hence there is no problem with cross-talk on the electric commercial grid.

- To avoid the problem with the discrete velocities, the burst would have to be made longer, e.g. including 16 or even 32 periods. However, this is not possible either, as the periodicity of the signal would then shrink further. The lowest periodicity that is allowed for security reasons is $2\mathrm{Hz}$, because otherwise, the trains would not react fast enough, e.g. when passing a closed semaphore.

Speed Control in Train Engines VII

To avoid the problems encountered before, more advanced control circuits were developed that made use of a *four-quadrant rectifier* with *commutation*.

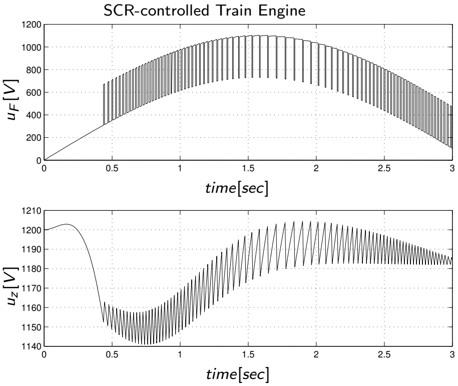


Speed Control in Train Engines VIII

- The line current, i_L, is controlled in such a way that it always remains in the vicinity of
$$Y(t) = \frac{15 \cdot 10^6}{u_L} \sin \omega t$$
- For $A_z = 0.0$, the line current, i_L, grows rapidly until it crosses $(Y + B_T)$ in the positive direction.
- At that moment, A_z assumes a value of $A_z = 1.0$, and i_L decays quickly again until it reaches $(Y - B_T)$, where A_z takes a value of $A_Z = 0.0$ as before.
- $$B_T = 200.0 \text{ Amps}$$
is the allowed tolerance around $Y(t)$, within which i_L is supposed to operate.

Speed Control in Train Engines IX

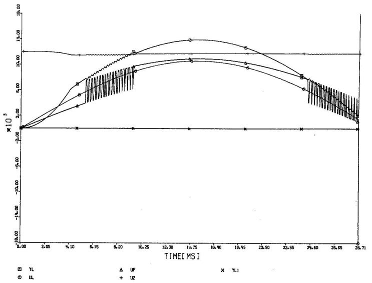
We expect to obtain the following trajectories from the simulation:

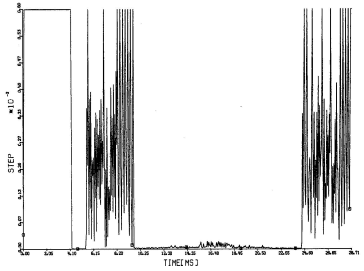


- The trajectories exhibit very serious discontinuities. For this reason, the simulation requires a *highly efficient and accurate step-size control*.
- These signals contain much power at high frequencies in their power spectrum. Yet, these contributions are at frequencies much higher than 50Hz.
- Thus, we don't experience problems with cross-talk.

Speed Control in Train Engines X

The circuit was originally simulated in **CSMP-III** using *RK4 with abuse of step-size control* to handle the discontinuities. The trajectories found were:





During some time interval, the *the simulation trajectories obtained were incorrect*. During those periods, **the simulation was creeping along** using the *smallest integration step size permitted by the software*.

Speed Control in Train Engines XI

What happened?

- We used an *explicit RK4 algorithm with abuse of step-size control* for handling the discontinuities.
- It was an algorithm in four stages.
- The effects of a commutation are so serious in this model that it sometimes happens that more than one commutation takes place within a single integration step.
- If the number of commutations within a step was *even*, we ended up at the end of the step on the same side of the fence and had to try jumping over the fence once again during the next step.
- The simulation proceeded during much time with the smallest allowed step size, unable to cross the fence.

Abuse of integration step-size control for the handling of discontinuities is not a good idea. The resulting algorithms aren't robust. Sometimes, the technique works quite well, but at other times, it fails miserably.

Time Events

Our problems arose from the fact that *we didn't instruct* the ODE solver that there were discontinuities. The ODE solver is incapable of *reading and interpreting* models. It can only *execute* them.

What we need is a *syntactical element* in the *model description language* that enables us to explicitly inform the integrator of occurrences of discontinuities.

From now on, we shall call discontinuities *discrete events*. What we need are explicit mechanisms for the description and handling of events.

In some cases, the time of occurrence of an event is known in advance. In this case, we talk about *time events*.

Time Events II

Let us consider once more the train speed control by a single thyristor. The time when the thyristor closes is known beforehand. The thyristor closes for the first time α degrees after the beginning of the period.

We can calculate:

$$\begin{aligned}\Delta t_{\text{period}} &= \frac{1}{2\pi f} \\ \Delta t_{\text{event}} &= \frac{\alpha}{360} \cdot \Delta t_{\text{period}}\end{aligned}$$

The first occurrence of this event can be *planned ahead*. Thus, this is a time event.

One of the actions associated with the closing event is the planning (scheduling) of the next occurrence of the same event at time $t + \Delta t_{\text{period}}$, where t denotes the current time.

Time Events III

One way to advise the simulation of forthcoming time events is by offering in the language a schedule statement that can be used to schedule future time events.

For example, we might include in the *initial section* of the program the following statements:

```
Gate = open;
schedule CloseGateEvent at $\Delta t_{\mathrm{event}}$;
```

and in the code of the *CloseGateEvent function*:

```
Gate = closed;
schedule CloseGateEvent at $t + \Delta t_{period}$;
```

The event, during which the thyristor opens again, is a different type of event. We shall talk about that class of events later.

Time Events IV

It should be mentioned that the *discontinuity associated with the event* is not part of the continuous-time model. All simulation trajectories in between events are perfectly continuous. Only the *conditions under which events are to occur* are known during the continuous simulation.

For this reason, the ODE solver won't encounter any problems. It never attempts to simulate across a discontinuity in the model.

The *localization of a time event* is trivial. All we need in order to localize time events accurately is an ODE solver that provides *dense output*. We proceed in exactly the same manner that we use to localize a communication instant, at which we wish to *report the values of the output variables*.

If the integration algorithm reduces the step in order to hit the *communication instants* (common for single-step algorithms), we shall do the same in order to arrive at the time of the next time event.

On the other hand, if the integration algorithm uses interpolation for the purpose of calculating the values of the output variables at communication instants (common for multi-step algorithms), we shall do the same for localizing time events.

Time Events V

Once an event has been localized, the actions associated with the event are being executed.

Afterwards, we are dealing with a new continuous-time simulation with new initial conditions.

The *time events* are stored in an *event calendar*, a linear linked list, in such a way that the *next time event* is always stored at the beginning of the list.

The continuous simulation only needs to know the *instant of time when the next time event is scheduled to occur*.

The continuous simulation proceeds until the time of occurrence of the next event. At that moment, the continuous simulation terminates, the actions associated with the event (i.e., the discontinuity) are being processed, and a new continuous simulation starts with new initial conditions.

Time Events VI

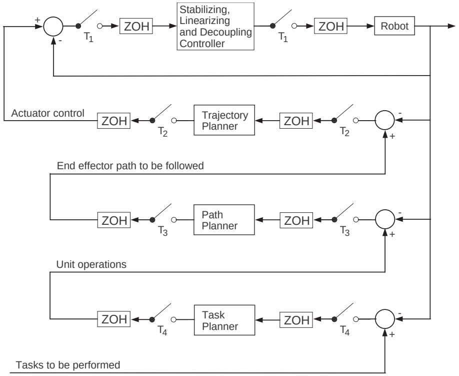
Consequently, the continuous simulations are now executed in segments. The individual continuous-time simulation segments are interrupted for handling discrete events that occur.

A model that specifies explicitly the discrete events is called *hybrid model*.

The simulation of a hybrid model is called *hybrid simulation*.

Simulation of Sampled-data Systems

Digital control systems can be modeled by means of hybrid models. They can then be simulated using a hybrid simulation engine.



- There are four closed loops employing different *sampling rates* $T_1 \leq T_2 \leq T_3 \ll T_4$.
- Every sampler represents an infinite series of self-scheduling time events.
- The event calendar maintains the four types of events and decides at every moment, which is the next event that needs to be executed.

State Events

Often the *time of occurrence* of an event is not known in advance. What is known is only the *condition under which the event is to take place*.

Such events are called *state events*.

State events *cannot be planned*.

In the thyristor control example, the gate opening event is a state event. We don't know ahead of time, when it happens. We only know under what conditions it happens, i.e., when the load current passes through zero.

To this end, we might include in the section where the *simulation equations* are being described the following statement:

schedule *OpenGateEvent* **when** $i_{\mathrm{Load}} < 0$;

and the code of the *OpenGateEvent function* consists of:

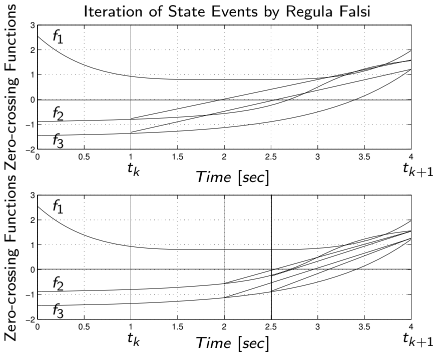
$Gate = open$;

State Events II

- State events are specified using *event conditions*, usually formulated in the form of *zero-crossing functions*.
- Such a function is called *event detection function*.
- There can be multiple event detection functions active simultaneously.
- During the continuous simulation, all active event detection functions must be constantly monitored to check whether any of the zero crossings has taken place.
- Once one of the event detection functions has found a zero crossing, an *event localization* algorithm is triggered that iterates on the exact time of the zero crossing.
- In the mathematical literature, the event localization algorithm is often also referred to as *root finding algorithm*.

Multiple Zero Crossings

If more than one event detection function is triggered during a single integration step, we need an algorithm to determine, which of the zero crossing happens first.



- Using the *Regula Falsi* method, the end points of every triggered zero-crossing function are connected.
- The next evaluation time instant is calculated using the formula:

$$t_{\text{next}} = \min_{\forall i} \left[\frac{f_i(t_{k+1}) \cdot t_k - f_i(t_k) \cdot t_{k+1}}{f_i(t_{k+1}) - f_i(t_k)} \right]$$

- If there is no zero crossing in the interval $t \in [t_k, t_{\text{next}}]$, we set $t_k = t_{\text{next}}$ and repeat the algorithm.
- On the other hand, if there are multiple zero crossings in the interval $t \in [t_k, t_{\text{next}}]$, we set $t_{k+1} = t_{\text{next}}$ and repeat the algorithm.
- *If there is exactly one zero crossing in the interval $t \in [t_k, t_{next}]$, we simplified the problem to that of localizing a single zero crossing.*

Multiple Zero Crossings II

- The *event isolation* using the *Regula Falsi* method converges always; the time interval gets reduced in every iteration step, but we cannot say beforehand, how many iterations are needed to obtain convergence.
- There exists another method that also converges always, but that offers the advantage that the time interval is reduced in every iteration step by 38.2%.
- This technique is called the *golden section* method.

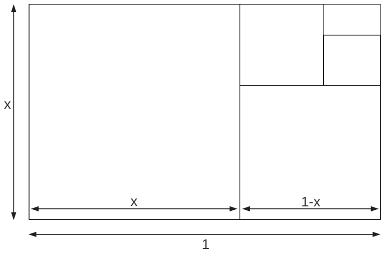
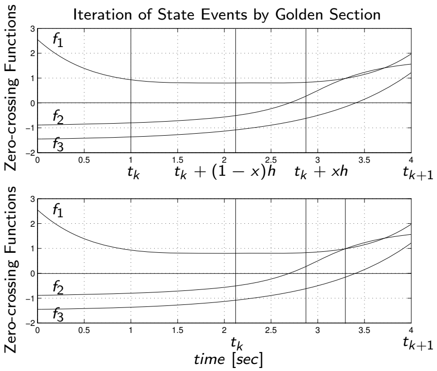


Figure: Golden Section.

$$\frac{x}{1} = \frac{1-x}{x} \quad \Rightarrow \quad x = 0.618$$

Multiple Zero Crossings III

How can the *golden section method* be used for *event isolation*?



- In the *golden section* method, two intermediate points are calculated in the time interval $t \in [t_k, t_{k+1}]$ cutting the interval using golden section.
- If there is no zero crossing in the interval $t \in [t_k, t_k + (1-x) \cdot h]$, we set $t_k = t_k + (1-x) \cdot h$ and repeat the algorithm with one new intermediate point.
- On the other hand, if there are multiple zero crossings in the interval $t \in [t_k, t_k + (1-x) \cdot h]$, we set $t_{k+1} = t_k + x \cdot h$ and repeat the algorithm with one new intermediate point.
- *If there is exactly one zero crossing in the interval $t \in [t_k, t_k + (1-x) \cdot h]$, we simplified the problem to that of localizing a single zero crossing.*

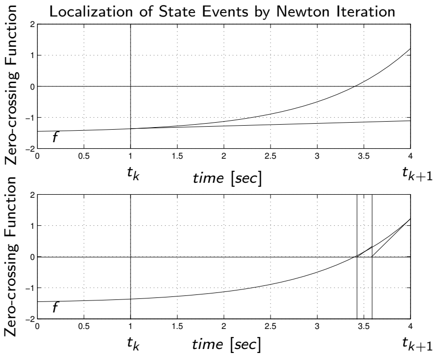
Multiple Zero Crossings IV

- We have meanwhile solved the problem of *event isolation*.
- If more than one zero crossing occurs within a single integration step, we reduce the interval $t \in [t_k, t_{k+1}]$ using either one of the two proposed methods: *Regula Falsi* or *golden section*, until there remains exactly one *zero crossing* of a single *event detection function* within the time interval.
- It can of course happen that two separate zero crossings occur at exactly the same time instant. In that case, the event isolation algorithm continues until the time interval has been sufficiently reduced so that we can consider the event localized, i.e., the *event isolation* algorithm is then also used for *event localization*.
- Thus, the original problem has thus been simplified to the problem of *localizing a single event that has already been identified to occur within a given fixed time interval*.

Event Localization

Single-step Algorithms

- Evidently, the *event isolation* techniques discussed previously can also be used for *event localization*.
- They may, however, be unnecessarily inefficient due to their *linear convergence speed*.
- We could suspect that *Newton iteration* might work better due to the *quadratic convergence speed* of this method.



Event Localization II

Single-step Algorithms

- Unfortunately, Newton iteration does not always converge. In the given example, if we start from the right, we obtain quick convergence, whereas if we start from the left, already the next iteration step carries us outside the interval.
- As we already know that the zero crossing occurs within a defined time interval, we are not interested in ever leaving the interval in search of the zero crossing.
- If the next Newton iteration step carries us outside the interval when starting on both ends of the interval, it may be advisable to reduce the width of the interval a bit more using one of the two event isolation algorithms introduced earlier before switching to Newton iteration for increasing the convergence speed.
- Once the time interval has been sufficiently reduced, Newton iteration will converge, and will do so more rapidly than either Regula Falsi or golden section.

Event Localization III

Single-step Algorithms

- Another algorithm that is possibly even more efficient for event localization than *Newton iteration* is *polynomial interpolation*.
- As every event detection function can be expressed as a function of state variables, we know not only the values of the event detection functions at both ends of the time interval, but also their time derivatives.
- Consequently, we may propose an interpolation polynomial:

  $$p(t) = a \cdot t^3 + b \cdot t^2 + c \cdot t + d$$

  with the time derivative:

  $$\dot{p}(t) = 3a \cdot t^2 + 2b \cdot t + c$$

- We know that:

  $$\begin{aligned} p(t_k) &= a \cdot t_k^3 + b \cdot t_k^2 + c \cdot t_k + d = f_k \\ p(t_{k+1}) &= a \cdot t_{k+1}^3 + b \cdot t_{k+1}^2 + c \cdot t_{k+1} + d = f_{k+1} \\ \dot{p}(t_k) &= 3a \cdot t_k^2 + 2b \cdot t_k + c = \dot{f}_k = h_k \\ \dot{p}(t_{k+1}) &= 3a \cdot t_{k+1}^2 + 2b \cdot t_{k+1} + c = \dot{f}_{k+1} = h_{k+1} \end{aligned}$$

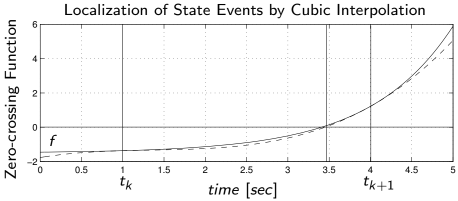
Event Localization IV

Single-step Algorithms

- We conclude:

$$\begin{pmatrix} f_k \\ f_{k+1} \\ h_k \\ h_{k+1} \end{pmatrix} = \begin{pmatrix} t_k^3 & t_k^2 & t_k & 1 \\ t_{k+1}^3 & t_{k+1}^2 & t_{k+1} & 1 \\ 3t_k^2 & 2t_k & 1 & 0 \\ 3t_{k+1}^2 & 2t_{k+1} & 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix}$$

- From the linear system of equations, we can easily determine the values of the four coefficients of the interpolation polynomial.



Event Localization V

Single-step Algorithms

- The *Cubic interpolation* algorithm is characterized by a *cubic convergence speed*. For this reason, the algorithm converges even faster than *Newton iteration*.

- It makes use of all of the information (the two function values and the two derivative values) available, and consequently, it is optimally suited for the task at hand.

- Furthermore, the algorithm encounters at least one solution inside the time interval. Consequently, the *convergence can be guaranteed* just as in the case of the *Regula Falsi* and *golden section* methods.

Event Localization

Multi-step Algorithms

- If we are using a *multi-step algorithm* for the simulation, the event localization algorithm can be improved even further, because we have access to the *Nordsieck vector*.
- We can write:

$$\mathcal{F}(\hat{h}) = \mathcal{F}_i(t_{\text{next}}) = \mathcal{F}_i(t_{k+1}) + \hat{h}\frac{d\mathcal{F}_i(t_{k+1})}{dt} + \frac{\hat{h}^2}{2}\frac{d^2\mathcal{F}_i(t_{k+1})}{dt^2} + \frac{\hat{h}^3}{6}\frac{d^3\mathcal{F}_i(t_{k+1})}{dt^3} + \dots = 0.0$$

- This is a function of the unknown $\hat{h}$ that can be solved for the unknown by Newton iteration.
- We begin with $\hat{h}^0 = 0.5 \cdot (t_k - t_{k+1})$ and iterate:

$$\hat{h}^{\ell+1} = \hat{h}^{\ell} - \frac{\mathcal{F}(\hat{h}^{\ell})}{\mathcal{H}(\hat{h}^{\ell})}$$

where:

$$\mathcal{H}(\hat{h}) = \frac{d\mathcal{F}(\hat{h})}{d\hat{h}} = \frac{d\mathcal{F}_i(t_{k+1})}{dt} + \hat{h}\frac{d^2\mathcal{F}_i(t_{k+1})}{dt^2} + \frac{\hat{h}^2}{2}\frac{d^3\mathcal{F}_i(t_{k+1})}{dt^3} + \dots$$

Event Localization II

Multi-step Algorithms

- As we have access to the *Nordsieck vector*, this iteration converges in a single iteration step with the same precision as the integration itself. We couldn't hope for anything more.

- We only have access to the Nordsieck vector for state variables. Thus, if the event detection function doesn't coincide with one of the state variables, it may be worth our while to augment the differential equation system with:

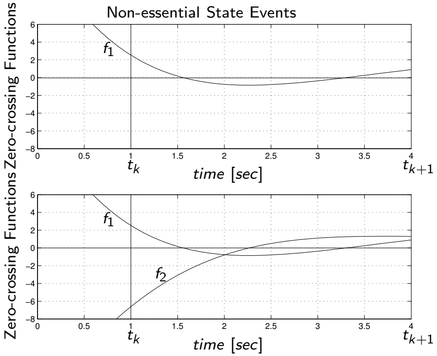
$$\dot{x}_{n+i} = \frac{d\mathcal{F}_i(\mathbf{x})}{dt}$$

- In this way, we shall need to integrate one additional state equation, but the technique simplifies and accelerates the localization of events.

- Since event localization may occupy much of the overall simulation time when dealing with a system with frequent discontinuities, such as in the case of the last version of the train speed control, augmenting the model in this fashion may turn out to be economical.

Non-essential State Events

Sometimes it is a good idea to augment the set of event detection functions by an additional function that is the derivative of another.



- Initially, we detect the zero crossing of function f_2.
- While iterating on this zero crossing, a second and earlier zero crossing of function f_1 is detected.
- The event associated with function f_2 is called *non-essential*, because no action is associated with this event.
- However, without the additional event detection function, f_2, we would have missed the two essential zero crossings of function f_1.

Conclusions

- In this presentation, we have discussed why *models containing discontinuities require special provisions* both on the side of the modeling language (the discontinuities should be declared in the model either directly or indirectly) and also on the side of the simulation engine (we should not simulate across discontinuities).

- It was shown that the step-size control algorithm may allow a model with discontinuities to be simulated successfully even without offering root-solving algorithms to the simulation engine, but we also demonstrated that this approach can fail.

- We then introduced the two types of events, the *time events* and the *state events*.

- In the final part of the presentation, we focused on the *root-solving algorithms* themselves. We decomposed the overall problem into an *event isolation* algorithm and an *event localization* algorithm.

References

1. Cellier, F.E. (1986), "Combined Continuous/Discrete Simulation - Applications, Techniques and Tools," *Proc. Winter Simulation Conference*, Washington, DC, pp.24-33.

2. Cellier, F.E., H. Elmqvist, M. Otter, and J.H. Taylor (1993), "Guidelines for Modeling and Simulation of Hybrid Systems," *Proc. IFAC World Congress*, Sydney, Australia, vol.8, pp.391-397.

3. Cellier, F.E. (1979), *Combined Continuous/Discrete System Simulation by Use of Digital Computers: Techniques and Tools*, Ph.D. Dissertation, ETH Zurich, Switzerland.