Numerical Simulation of Dynamic Systems XXVII

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

May 28, 2013

Introduction

This 27^{th} and final presentation of the class on the *Numerical Simulation of Dynamic Systems* deals with "Chapter 13 of a 12-chapter book," i.e., it discusses a number of issues related to *Quantized State System (QSS) solvers* that were discovered only after the book went to print.

Introduction

This 27^{th} and final presentation of the class on the *Numerical Simulation of Dynamic Systems* deals with "Chapter 13 of a 12-chapter book," i.e., it discusses a number of issues related to *Quantized State System (QSS) solvers* that were discovered only after the book went to print.

- Until now, we have always controlled the *absolute integration error*. It may be more robust to control the *relative integration error*, as this quantity doesn't need to be set for each simulation separately. Hence being able to specify the relative error tolerance makes the software more robust and user-friendly.

Introduction

This 27th and final presentation of the class on the *Numerical Simulation of Dynamic Systems* deals with "Chapter 13 of a 12-chapter book," i.e., it discusses a number of issues related to *Quantized State System (QSS) solvers* that were discovered only after the book went to print.

- Until now, we have always controlled the *absolute integration error*. It may be more robust to control the *relative integration error*, as this quantity doesn't need to be set for each simulation separately. Hence being able to specify the relative error tolerance makes the software more robust and user-friendly.
- Until now, we have only dealt with *non-stiff QSS-based integrators*. Yet, we know from the earlier presentations that *large systems are almost invariably stiff*, and consequently, we shall need also *stiff QSS-based integrators*. Such algorithms have meanwhile been developed and shall be discussed in this presentation.

Introduction

This 27^{th} and final presentation of the class on the *Numerical Simulation of Dynamic Systems* deals with "Chapter 13 of a 12-chapter book," i.e., it discusses a number of issues related to *Quantized State System (QSS) solvers* that were discovered only after the book went to print.

- Until now, we have always controlled the *absolute integration error*. It may be more robust to control the *relative integration error*, as this quantity doesn't need to be set for each simulation separately. Hence being able to specify the relative error tolerance makes the software more robust and user-friendly.
- Until now, we have only dealt with *non-stiff QSS-based integrators*. Yet, we know from the earlier presentations that *large systems are almost invariably stiff*, and consequently, we shall need also *stiff QSS-based integrators*. Such algorithms have meanwhile been developed and shall be discussed in this presentation.
- Also of importance are special solvers for dealing with *marginally stable systems*, and indeed, an *F-stable QSS-based solver* has also been developed since the book went to press.

Introduction II

- A type of models that we haven't dealt with in this class is *models that contain delays*. We call a set of ordinary differential equations with delays a *delay differential equation (DDE) model*, and we call algorithms that are suitable for the simulation of DDE models *numerical DDE solvers*. A QSS-based DDE solver has meanwhile also been developed and shall be discussed.

Introduction II

- A type of models that we haven't dealt with in this class is *models that contain delays*. We call a set of ordinary differential equations with delays a *delay differential equation (DDE) model*, and we call algorithms that are suitable for the simulation of DDE models *numerical DDE solvers*. A QSS-based DDE solver has meanwhile also been developed and shall be discussed.

- Finally, a few remarks shall be offered on the use of QSS-based algorithms in *real-time simulations*.

Choosing the Quantum Size

Let us suppose that we wish to simulate a system in which a state variable x_1 grows from 0 to 1000, while x_2 grows from 0 to 1.

Choosing the Quantum Size

Let us suppose that we wish to simulate a system in which a state variable x_1 grows from 0 to 1000, while x_2 grows from 0 to 1.

- Using QSS, if we use a quantum of $\Delta Q_1 = \Delta Q_2 = 0.1$, variable q_1 will change $10,000$ times, while variable q_2 will change only 10 times.

Choosing the Quantum Size

Let us suppose that we wish to simulate a system in which a state variable x_1 grows from 0 to 1000, while x_2 grows from 0 to 1.

- Using QSS, if we use a quantum of $\Delta Q_1 = \Delta Q_2 = 0.1$, variable q_1 will change $10,000$ times, while variable q_2 will change only 10 times.
- Using QSS2 or QSS3, the number of steps differs in a less dramatic way, but it still depends on the amplitude of the signals to be integrated.

Choosing the Quantum Size

Let us suppose that we wish to simulate a system in which a state variable x_1 grows from 0 to 1000, while x_2 grows from 0 to 1.

- Using QSS, if we use a quantum of $\Delta Q_1 = \Delta Q_2 = 0.1$, variable q_1 will change $10,000$ times, while variable q_2 will change only 10 times.
- Using QSS2 or QSS3, the number of steps differs in a less dramatic way, but it still depends on the amplitude of the signals to be integrated.
- Consequently, an appropriate value of the quantum used for each state variable depends on the variability of that state variable.

Choosing the Quantum Size

Let us suppose that we wish to simulate a system in which a state variable x_1 grows from 0 to 1000, while x_2 grows from 0 to 1.

- Using QSS, if we use a quantum of $\Delta Q_1 = \Delta Q_2 = 0.1$, variable q_1 will change $10,000$ times, while variable q_2 will change only 10 times.
- Using QSS2 or QSS3, the number of steps differs in a less dramatic way, but it still depends on the amplitude of the signals to be integrated.
- Consequently, an appropriate value of the quantum used for each state variable depends on the variability of that state variable.
- Unfortunately, before performing the simulation, we normally do not know, how the state variables will develop over time.

Choosing the Quantum Size

Let us suppose that we wish to simulate a system in which a state variable x_1 grows from 0 to 1000, while x_2 grows from 0 to 1.

- Using QSS, if we use a quantum of $\Delta Q_1 = \Delta Q_2 = 0.1$, variable q_1 will change $10,000$ times, while variable q_2 will change only 10 times.
- Using QSS2 or QSS3, the number of steps differs in a less dramatic way, but it still depends on the amplitude of the signals to be integrated.
- Consequently, an appropriate value of the quantum used for each state variable depends on the variability of that state variable.
- Unfortunately, before performing the simulation, we normally do not know, how the state variables will develop over time.

We need some mechanism to adjust the size of the quantum automatically during the simulation in accordance with the values that the state variable assumes over time.

Logarithmic Quantization

We shall set the quantum *proportional to the absolute value of each state*. Also, to avoid problems around zero, we shall limit the minimum quantum value.

Logarithmic Quantization

We shall set the quantum *proportional to the absolute value of each state*. Also, to avoid problems around zero, we shall limit the minimum quantum value.

Thus, we shall express the quantum size as:

$$\Delta Q_i(t) = \max(\Delta Q_{rel_i} \cdot |x_i(t_k)|, \Delta Q_{min_i})$$

where t_k is the time of the last change of $q_i(t)$.

Logarithmic Quantization

We shall set the quantum *proportional to the absolute value of each state*. Also, to avoid problems around zero, we shall limit the minimum quantum value.

Thus, we shall express the quantum size as:

$$\Delta Q_i(t) = \max(\Delta Q_{rel_i} \cdot |x_i(t_k)|, \Delta Q_{min_i})$$

where t_k is the time of the last change of $q_i(t)$.

The parameters Q_{rel} and Q_{min} can be specified for each hysteretic quantized integrator separately.

Logarithmic Quantization

We shall set the quantum *proportional to the absolute value of each state*. Also, to avoid problems around zero, we shall limit the minimum quantum value.

Thus, we shall express the quantum size as:

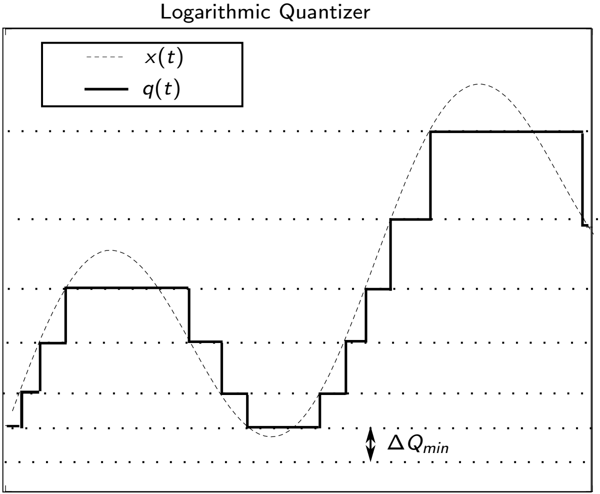
$$\Delta Q_i(t) = \max(\Delta Q_{rel_i} \cdot |x_i(t_k)|, \Delta Q_{min_i})$$

where t_k is the time of the last change of $q_i(t)$.

The parameters Q_{rel} and Q_{min} can be specified for each hysteretic quantized integrator separately.

This idea can be applied to all QSS methods.

Logarithmic Quantization II



Perturbed Representation

Recall that the QSS approximation of a linear time-invariant ODE:

$$\dot{\mathbf{x}}_{\mathbf{a}}(t) = \mathbf{A} \cdot \mathbf{x}_{\mathbf{a}}(t) + \mathbf{B} \cdot \mathbf{u}(t)$$

can be written as:

$$\dot{\mathbf{x}}(t) = \mathbf{A} \cdot \mathbf{q}(t) + \mathbf{B} \cdot \mathbf{u}(t)$$

Perturbed Representation

Recall that the QSS approximation of a linear time-invariant ODE:

$$\dot{\mathbf{x}}_{\mathbf{a}}(t) = \mathbf{A} \cdot \mathbf{x}_{\mathbf{a}}(t) + \mathbf{B} \cdot \mathbf{u}(t)$$

can be written as:

$$\dot{\mathbf{x}}(t) = \mathbf{A} \cdot \mathbf{q}(t) + \mathbf{B} \cdot \mathbf{u}(t)$$

Defining $\boldsymbol{\Delta}\mathbf{x}(t) \triangleq \mathbf{q}(t) - \mathbf{x}(t)$, we have:

$$\dot{\mathbf{x}}(t) = \mathbf{A} \cdot (\mathbf{x}(t) + \boldsymbol{\Delta}\mathbf{x}(t)) + \mathbf{B} \cdot \mathbf{u}(t)$$

Perturbed Representation

Recall that the QSS approximation of a linear time-invariant ODE:

$$\dot{\mathbf{x}}_{\mathbf{a}}(t) = \mathbf{A} \cdot \mathbf{x}_{\mathbf{a}}(t) + \mathbf{B} \cdot \mathbf{u}(t)$$

can be written as:

$$\dot{\mathbf{x}}(t) = \mathbf{A} \cdot \mathbf{q}(t) + \mathbf{B} \cdot \mathbf{u}(t)$$

Defining $\mathbf{\Delta x}(t) \triangleq \mathbf{q}(t) - \mathbf{x}(t)$, we have:

$$\dot{\mathbf{x}}(t) = \mathbf{A} \cdot (\mathbf{x}(t) + \mathbf{\Delta x}(t)) + \mathbf{B} \cdot \mathbf{u}(t)$$

This perturbed representation does not depend on the type of quantization. However, with *logarithmic quantization*, the upper bound of $|\mathbf{\Delta x}(t))|$ depends on the state value $\mathbf{x}(t)$.

Stability and Error Bound

Assume that matrix $\mathbf{A} = \mathbf{V} \cdot \mathbf{\Lambda} \cdot \mathbf{V}^{-1}$ is Hurwitz, and define:

$$\mathbf{R} \triangleq |\mathbf{V}| \cdot |\mathbb{R}\mathrm{e}(\mathbf{\Lambda})^{-1}\mathbf{\Lambda}| \cdot |\mathbf{V}^{-1}|$$

Stability and Error Bound

Assume that matrix $\mathbf{A} = \mathbf{V} \cdot \mathbf{\Lambda} \cdot \mathbf{V}^{-1}$ is Hurwitz, and define:

$$\mathbf{R} \triangleq |\mathbf{V}| \cdot |\mathbb{R}\mathrm{e}(\mathbf{\Lambda})^{-1}\mathbf{\Lambda}| \cdot |\mathbf{V}^{-1}|$$

Let $\mathbf{\Delta Q_{rel}}$ be a diagonal matrix with entries ΔQ_{rel_i}, and let $\mathbf{\Delta Q_{min}}$ be a column vector with components ΔQ_{min_i}. Then, provided that all eigenvalues of the matrix $\mathbf{R} \cdot \mathbf{\Delta Q_{rel}}$ lie inside the unit circle, it can be shown that:

$$|\mathbf{e}(t)| \leq (\mathbf{I^{(n)}} - \mathbf{R} \cdot \mathbf{\Delta Q_{rel}})^{-1} \cdot \mathbf{R} \cdot \max(\mathbf{\Delta Q_{rel}} \cdot \mathbf{x_{max}}, \mathbf{\Delta Q_{min}})$$

where $\mathbf{x_{max}}$ is the column vector of the maximum absolute values reached by each component of the analytical solution $\mathbf{x_a}(t)$.

Logarithmic Quantization and Relative Error Control

- Generally, $\mathbf{\Delta Q_{rel}}$ is chosen small, which ensures that all eigenvalues of $\mathbf{R} \cdot \mathbf{\Delta Q_{rel}}$ lie inside the unit circle.

Logarithmic Quantization and Relative Error Control

- Generally, $\mathbf{\Delta Q_{rel}}$ is chosen small, which ensures that all eigenvalues of $\mathbf{R} \cdot \mathbf{\Delta Q_{rel}}$ lie inside the unit circle.
- In this case, we can approximate $(\mathbf{I^{(n)}} - \mathbf{R} \cdot \mathbf{\Delta Q_{rel}})^{-1} \approx \mathbf{I^{(n)}}$.

Logarithmic Quantization and Relative Error Control

- Generally, $\mathbf{\Delta Q_{rel}}$ is chosen small, which ensures that all eigenvalues of $\mathbf{R} \cdot \mathbf{\Delta Q_{rel}}$ lie inside the unit circle.
- In this case, we can approximate $(\mathbf{I^{(n)}} - \mathbf{R} \cdot \mathbf{\Delta Q_{rel}})^{-1} \approx \mathbf{I^{(n)}}$.
- Also, $\mathbf{x_{max}}$ is normally much bigger than $\mathbf{\Delta Q_{min}}$.

Logarithmic Quantization and Relative Error Control

- Generally, $\mathbf{\Delta Q_{rel}}$ is chosen small, which ensures that all eigenvalues of $\mathbf{R} \cdot \mathbf{\Delta Q_{rel}}$ lie inside the unit circle.
- In this case, we can approximate $(\mathbf{I^{(n)}} - \mathbf{R} \cdot \mathbf{\Delta Q_{rel}})^{-1} \approx \mathbf{I^{(n)}}$.
- Also, $\mathbf{x_{max}}$ is normally much bigger than $\mathbf{\Delta Q_{min}}$.
- Usually, we shall choose $\Delta Q_{rel_i} = tol_{rel}$ for all i.

Logarithmic Quantization and Relative Error Control

- Generally, $\mathbf{\Delta Q_{rel}}$ is chosen small, which ensures that all eigenvalues of $\mathbf{R} \cdot \mathbf{\Delta Q_{rel}}$ lie inside the unit circle.
- In this case, we can approximate $(\mathbf{I^{(n)}} - \mathbf{R} \cdot \mathbf{\Delta Q_{rel}})^{-1} \approx \mathbf{I^{(n)}}$.
- Also, $\mathbf{x_{max}}$ is normally much bigger than $\mathbf{\Delta Q_{min}}$.
- Usually, we shall choose $\Delta Q_{rel_i} = tol_{rel}$ for all i.

In this case, we obtain:

$$|\mathbf{e}(t)| \leq (\mathbf{I^{(n)}} - \mathbf{R} \cdot \mathbf{\Delta Q_{rel}})^{-1} \cdot \mathbf{R} \cdot \max(\mathbf{\Delta Q_{rel}} \cdot \mathbf{x_{max}}, \mathbf{\Delta Q_{min}}) \approx \mathbf{R} \cdot tol_{rel} \cdot |\mathbf{x_{max}}|$$

Logarithmic Quantization and Relative Error Control

- Generally, $\mathbf{\Delta Q_{rel}}$ is chosen small, which ensures that all eigenvalues of $\mathbf{R} \cdot \mathbf{\Delta Q_{rel}}$ lie inside the unit circle.
- In this case, we can approximate $(\mathbf{I^{(n)}} - \mathbf{R} \cdot \mathbf{\Delta Q_{rel}})^{-1} \approx \mathbf{I^{(n)}}$.
- Also, $\mathbf{x_{max}}$ is normally much bigger than $\mathbf{\Delta Q_{min}}$.
- Usually, we shall choose $\Delta Q_{rel_i} = tol_{rel}$ for all i.

In this case, we obtain:

$$|\mathbf{e}(t)| \leq (\mathbf{I^{(n)}} - \mathbf{R} \cdot \mathbf{\Delta Q_{rel}})^{-1} \cdot \mathbf{R} \cdot \max(\mathbf{\Delta Q_{rel}} \cdot \mathbf{x_{max}}, \mathbf{\Delta Q_{min}}) \approx \mathbf{R} \cdot tol_{rel} \cdot |\mathbf{x_{max}}|$$

The use of logarithmic quantization yields an intrinsic control of the *relative error*.

QSS Methods and Stiff Systems

The linear time-invariant system:

$$\dot{x}_1(t) = 0.01 \; x_2(t)$$
$$\dot{x}_2(t) = -100 \; x_1(t) - 100 \; x_2(t) + 2020$$

has eigenvalues $\lambda_1 \approx -0.01$ and $\lambda_2 \approx -99.99$, and consequently, *the model is stiff*.

QSS Methods and Stiff Systems

The linear time-invariant system:

$$\dot{x}_1(t) = 0.01\ x_2(t)$$
$$\dot{x}_2(t) = -100\ x_1(t) - 100\ x_2(t) + 2020$$

has eigenvalues $\lambda_1 \approx -0.01$ and $\lambda_2 \approx -99.99$, and consequently, *the model is stiff*.

The QSS method approximates this system by:

$$\dot{x}_1(t) = 0.01\ q_2(t)$$
$$\dot{x}_2(t) = -100\ q_1(t) - 100\ q_2(t) + 2020$$

QSS Methods and Stiff Systems

The linear time-invariant system:

$$\dot{x}_1(t) = 0.01\ x_2(t)$$
$$\dot{x}_2(t) = -100\ x_1(t) - 100\ x_2(t) + 2020$$

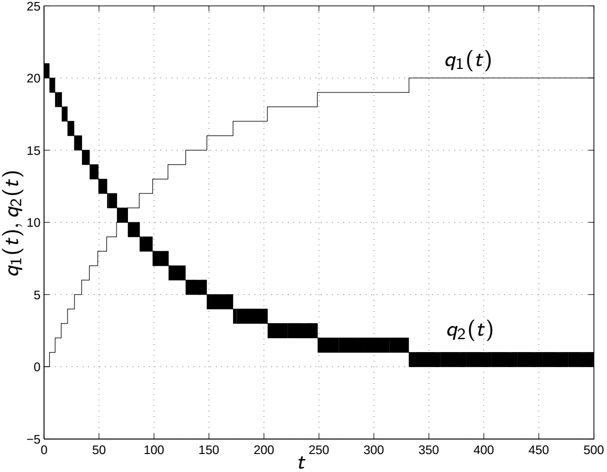
has eigenvalues $\lambda_1 \approx -0.01$ and $\lambda_2 \approx -99.99$, and consequently, *the model is stiff*.

The QSS method approximates this system by:

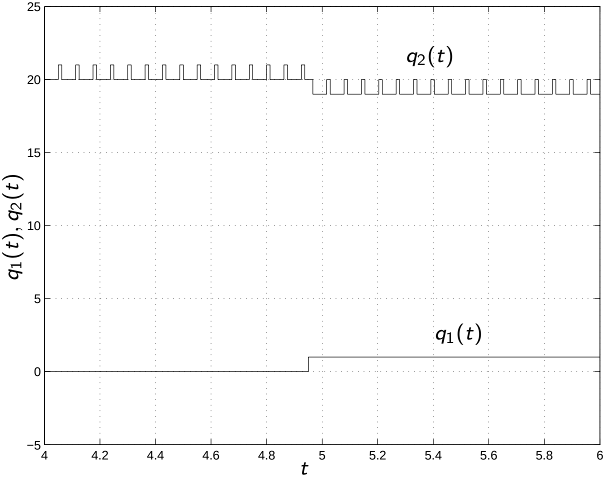
$$\dot{x}_1(t) = 0.01\ q_2(t)$$
$$\dot{x}_2(t) = -100\ q_1(t) - 100\ q_2(t) + 2020$$

Let us simulate this system using the QSS solver with initial conditions $x_1(0) = 0$ and $x_2(0) = 20$ and with the quantization $\Delta Q_1 = \Delta Q_2 = 1$.

QSS Methods and Stiff Systems II



QSS Methods and Stiff Systems III



QSS Methods and Stiff Systems IV

- Stiff systems usually provoke *fast oscillations* on the QSS solutions.

QSS Methods and Stiff Systems IV

- Stiff systems usually provoke *fast oscillations* on the QSS solutions.
- Consequently, the number of steps is very large.

QSS Methods and Stiff Systems IV

- Stiff systems usually provoke *fast oscillations* on the QSS solutions.
- Consequently, the number of steps is very large.
- In the simulated example, there were 21 changes in q_1 and $15,995$ changes in q_2 for a final simulation time of $t_f = 500$ *seconds*.

QSS Methods and Stiff Systems IV

- Stiff systems usually provoke *fast oscillations* on the QSS solutions.
- Consequently, the number of steps is very large.
- In the simulated example, there were 21 changes in q_1 and $15,995$ changes in q_2 for a final simulation time of $t_f = 500$ *seconds*.

Evidently, the QSS method is not appropriate for the simulation of stiff systems.

Backward QSS: Basic Idea

How can we prevent oscillations?

Backward QSS: Basic Idea

How can we prevent oscillations?

- In classical solvers, we had to use future values of state derivatives, i.e., *implicit methods*, in order to obtain *stiff system solvers*.

Backward QSS: Basic Idea

How can we prevent oscillations?

- In classical solvers, we had to use future values of state derivatives, i.e., *implicit methods*, in order to obtain *stiff system solvers*.
- Let us try to do the same also in QSS-based methods. We shall choose q_j such that it represents a *future value* of x_j.

Backward QSS: Basic Idea

How can we prevent oscillations?

- In classical solvers, we had to use future values of state derivatives, i.e., *implicit methods*, in order to obtain *stiff system solvers*.
- Let us try to do the same also in QSS-based methods. We shall choose q_j such that it represents a *future value* of x_j.
- In QSS, after a step in q_j, we set $q_j(t) = x_j(t)$. Now, we shall set $q_j(t) = x_j(t) \pm \Delta Q_j$ depending on the sign of $\dot{x}_j(t)$.

Backward QSS: Basic Idea

How can we prevent oscillations?

- In classical solvers, we had to use future values of state derivatives, i.e., *implicit methods*, in order to obtain *stiff system solvers*.
- Let us try to do the same also in QSS-based methods. We shall choose q_j such that it represents a *future value* of x_j.
- In QSS, after a step in q_j, we set $q_j(t) = x_j(t)$. Now, we shall set $q_j(t) = x_j(t) \pm \Delta Q_j$ depending on the sign of $\dot{x}_j(t)$.
- It may happen that $\dot{x}_j(t)\big|_{x_j+\Delta Q_j} < 0$ and $\dot{x}_j(t)\big|_{x_j-\Delta Q_j} > 0$. In that case, we cannot choose a suitable value for q_j. However in that case, there must exist a value $\hat{q}_j \in (x_j - \Delta Q_j, x_j + \Delta Q_j)$ for which $\dot{x}_j = 0$.

Backward QSS: Basic Idea

How can we prevent oscillations?

- In classical solvers, we had to use future values of state derivatives, i.e., *implicit methods*, in order to obtain *stiff system solvers*.
- Let us try to do the same also in QSS-based methods. We shall choose q_j such that it represents a *future value* of x_j.
- In QSS, after a step in q_j, we set $q_j(t) = x_j(t)$. Now, we shall set $q_j(t) = x_j(t) \pm \Delta Q_j$ depending on the sign of $\dot{x}_j(t)$.
- It may happen that $\dot{x}_j(t)\big|_{x_j+\Delta Q_j} < 0$ and $\dot{x}_j(t)\big|_{x_j-\Delta Q_j} > 0$. In that case, we cannot choose a suitable value for q_j. However in that case, there must exist a value $\hat{q}_j \in (x_j - \Delta Q_j, x_j + \Delta Q_j)$ for which $\dot{x}_j = 0$.
- If this happens, we keep $q_j(t) = x_j(t)$, but set $\dot{x}_j(t) = 0$, i.e., no further internal transition gets scheduled for the state variable in question.

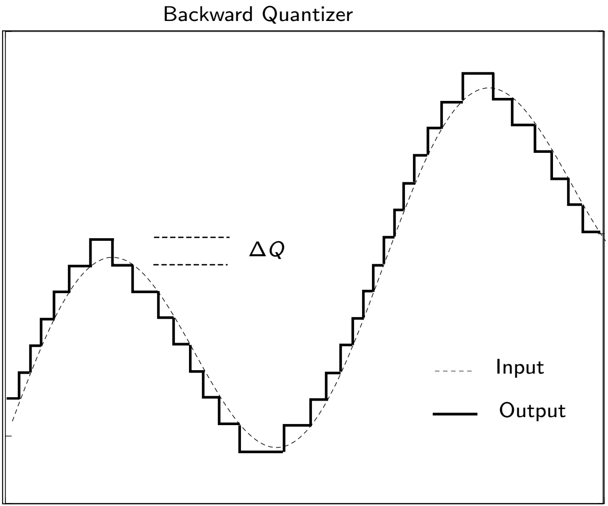
Backward QSS: Basic Idea

How can we prevent oscillations?

- In classical solvers, we had to use future values of state derivatives, i.e., *implicit methods*, in order to obtain *stiff system solvers*.
- Let us try to do the same also in QSS-based methods. We shall choose q_j such that it represents a *future value* of x_j.
- In QSS, after a step in q_j, we set $q_j(t) = x_j(t)$. Now, we shall set $q_j(t) = x_j(t) \pm \Delta Q_j$ depending on the sign of $\dot{x}_j(t)$.
- It may happen that $\dot{x}_j(t)\big|_{x_j+\Delta Q_j} < 0$ and $\dot{x}_j(t)\big|_{x_j-\Delta Q_j} > 0$. In that case, we cannot choose a suitable value for q_j. However in that case, there must exist a value $\hat{q}_j \in (x_j - \Delta Q_j, x_j + \Delta Q_j)$ for which $\dot{x}_j = 0$.
- If this happens, we keep $q_j(t) = x_j(t)$, but set $\dot{x}_j(t) = 0$, i.e., no further internal transition gets scheduled for the state variable in question.

This is an implicit algorithm, but it *does not require a Newton iteration*, as $q_j(t)$ can only assume one of two values at each step. The method is called *Backward QSS (BQSS) solver*.

Backward Quantization



Backward Quantization II

- In *non-stiff QSS solvers*, the trajectories of $q(t)$ and $x(t)$ at the beginning of the step always *move away from each other*. The next *internal transition* is scheduled to occur at time $\hat{t}$, where $|q(\hat{t}^-) - x(\hat{t})| = \Delta Q$. At that time, $q(t)$ is reset to $q(\hat{t}^+) = x(\hat{t})$.

Backward Quantization II

- In *non-stiff QSS solvers*, the trajectories of $q(t)$ and $x(t)$ at the beginning of the step always *move away from each other*. The next *internal transition* is scheduled to occur at time $\hat{t}$, where $|q(\hat{t}^-) - x(\hat{t})| = \Delta Q$. At that time, $q(t)$ is reset to $q(\hat{t}^+) = x(\hat{t})$.

- In *stiff QSS solvers*, the trajectories of $q(t)$ and $x(t)$ at the beginning of the step always *move toward each other*. The next internal transition is scheduled to occur at time $\hat{t}$, where $|q(\hat{t}^-) - x(\hat{t})| = 0$. At that time, $q(t)$ is reset to $q(\hat{t}^+) = x(\hat{t}) \pm \Delta Q$.

Backward Quantization II

- In *non-stiff QSS solvers*, the trajectories of $q(t)$ and $x(t)$ at the beginning of the step always *move away from each other*. The next *internal transition* is scheduled to occur at time $\hat{t}$, where $|q(\hat{t}^-) - x(\hat{t})| = \Delta Q$. At that time, $q(t)$ is reset to $q(\hat{t}^+) = x(\hat{t})$.

- In *stiff QSS solvers*, the trajectories of $q(t)$ and $x(t)$ at the beginning of the step always *move toward each other*. The next internal transition is scheduled to occur at time $\hat{t}$, where $|q(\hat{t}^-) - x(\hat{t})| = 0$. At that time, $q(t)$ is reset to $q(\hat{t}^+) = x(\hat{t}) \pm \Delta Q$.

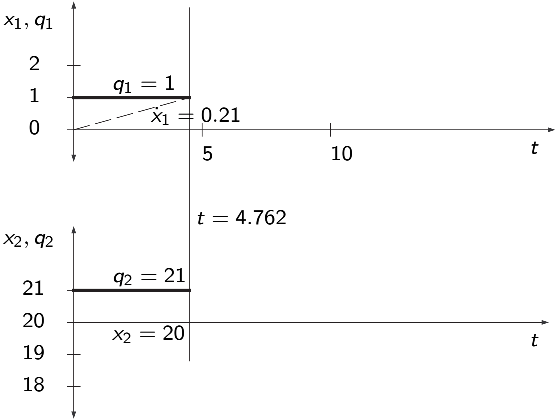
- In **BQSS**, if $q(\hat{t}^+)$ cannot be chosen such that the trajectories move toward each other, then no internal transition is scheduled, i.e., the *time advance function* is set to ∞, and $\dot{x}(t)$ is set to $\dot{x}(t) = 0$ irrespective of the current value of the input signal to the hysteretic quantized integrator.

Backward Quantization II

- In *non-stiff QSS solvers*, the trajectories of $q(t)$ and $x(t)$ at the beginning of the step always *move away from each other*. The next *internal transition* is scheduled to occur at time $\hat{t}$, where $|q(\hat{t}^-) - x(\hat{t})| = \Delta Q$. At that time, $q(t)$ is reset to $q(\hat{t}^+) = x(\hat{t})$.
- In *stiff QSS solvers*, the trajectories of $q(t)$ and $x(t)$ at the beginning of the step always *move toward each other*. The next internal transition is scheduled to occur at time $\hat{t}$, where $|q(\hat{t}^-) - x(\hat{t})| = 0$. At that time, $q(t)$ is reset to $q(\hat{t}^+) = x(\hat{t}) \pm \Delta Q$.
- In **BQSS**, if $q(\hat{t}^+)$ cannot be chosen such that the trajectories move toward each other, then no internal transition is scheduled, i.e., the *time advance function* is set to ∞, and $\dot{x}(t)$ is set to $\dot{x}(t) = 0$ irrespective of the current value of the input signal to the hysteretic quantized integrator.
- *Higher-order stiff QSS solvers* need additional provisions that shall be discussed in due course.

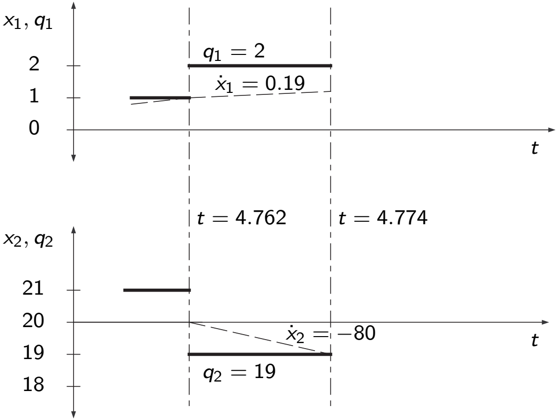
Backward QSS: An Example

$\dot{x}_1 = 0.01 q_2 \qquad \dot{x}_2 = -100 q_1 - 100 q_2 + 2020$

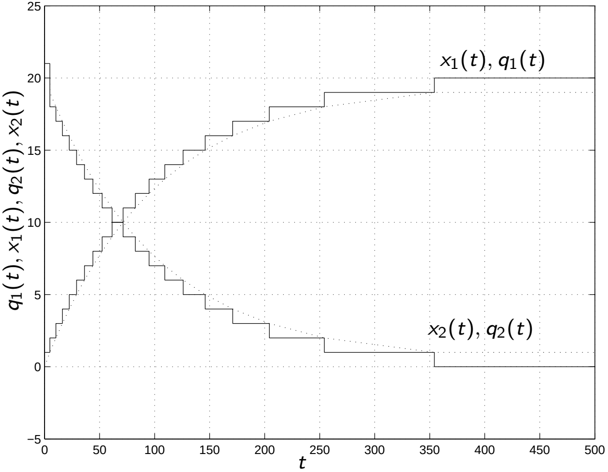


Backward QSS: An Example

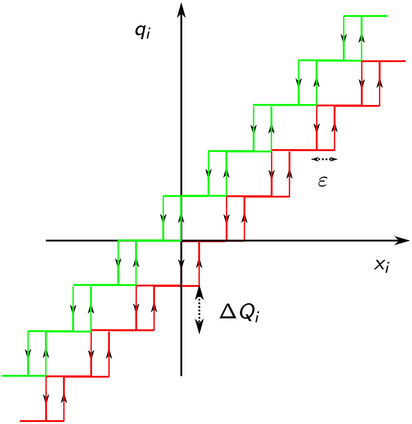
$\dot{x}_1 = 0.01 q_2$ $\qquad$ $\dot{x}_2 = -100 q_1 - 100 q_2 + 2020$



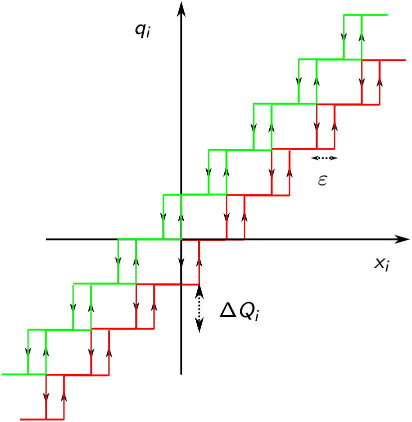
Backward QSS: An Example II



Quantization Functions in BQSS

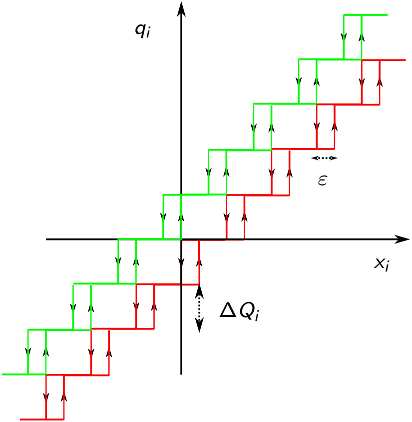


Quantization Functions in BQSS



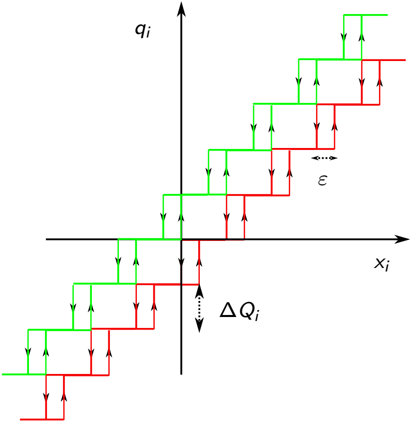
- **BQSS** uses a *lower and an upper hysteretic quantization function* for each state.

Quantization Functions in BQSS



- **BQSS** uses a *lower and an upper hysteretic quantization function* for each state.
- They compute a lower and an upper quantized state, $\underline{q}_j$ and $\overline{q}_j$.

Quantization Functions in BQSS



- **BQSS** uses a *lower and an upper hysteretic quantization function* for each state.
- They compute a lower and an upper quantized state, $\underline{q}_j$ and $\overline{q}_j$.
- The quantized state q_j is chosen from them in accordance with the sign of $\dot{x}_j$.

BQSS Definition

Given the system:

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

BQSS Definition

Given the system:

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

BQSS approximates it as:

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{q}(t), \mathbf{u}(t)) + \mathbf{\Delta f}$$

where the components q_j of $\mathbf{q}$ take the values of either the lower quantized state $\underline{q}_j(t)$ or the upper quantized state $\overline{q}_j(t)$, such that:

$$f_j(\mathbf{q}(t), \mathbf{u}(t)) \cdot (q_j(t) - x_j(t)) > 0$$

BQSS Definition

Given the system:

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

BQSS approximates it as:

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{q}(t), \mathbf{u}(t)) + \mathbf{\Delta f}$$

where the components q_j of $\mathbf{q}$ take the values of either the lower quantized state $\underline{q}_j(t)$ or the upper quantized state $\overline{q}_j(t)$, such that:

$$f_j(\mathbf{q}(t), \mathbf{u}(t)) \cdot (q_j(t) - x_j(t)) > 0$$

$$\vee$$

$$\exists \hat{q}_j \in (\underline{q}_j(t), \overline{q}_j(t))\Big|_{f_j(\hat{\mathbf{q}}^{(j)}(t), \mathbf{u}(t))} = 0$$

with $\hat{\mathbf{q}}^{(j)}(t) = [q_1, \ldots, \hat{q}_j, \ldots, q_n]^T$.

BQSS Definition II

In the BQSS approximation:

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{q}(t), \mathbf{u}(t)) + \mathbf{\Delta f}$$

the terms Δf_j will satisfy:

$$\Delta f_j = \begin{cases} 0, \text{ if } f_j(\mathbf{q}(t), \mathbf{u}(t)) \cdot (q_j - x_j) > 0 \\ -f_j(\mathbf{q}(t), \mathbf{u}(t)), \quad \text{otherwise} \end{cases}$$

Algorithm to Obtain $q(t)$

When a state $x_i(t)$ reaches the quantized state $q_i(t)$, we proceed as follows:

Algorithm to Obtain $q(t)$

When a state $x_i(t)$ reaches the quantized state $q_i(t)$, we proceed as follows:

Algorithm:

1. We update $\underline{q}_i$ and $\overline{q}_i$ and choose $q_i = \overline{q}_i$ or $q_i = \underline{q}_i$ depending on the sign of $\dot{x}_i(t^-)$.
2. We evaluate the functions f_j depending on q_i.
3. **If** some f_j changes its sign:
   - We change q_j according to the new sign of $\dot{x}_j$.

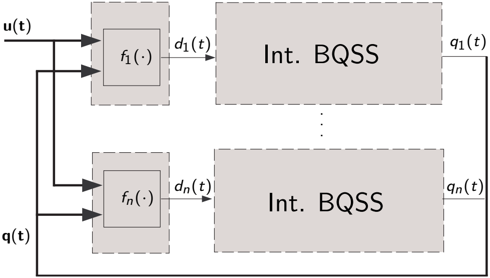
   Otherwise

   - End of the Algorithm.
4. We evaluate the functions f_j depending on the quantized states q_i that changed and we return to step 3.

Restriction: We do not allow any q_j to change more than once in the process.

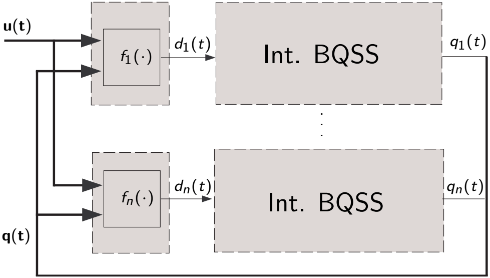
BQSS and PowerDEVS

- The *block diagram structure* of BQSS is the same as that of QSS.



BQSS and PowerDEVS

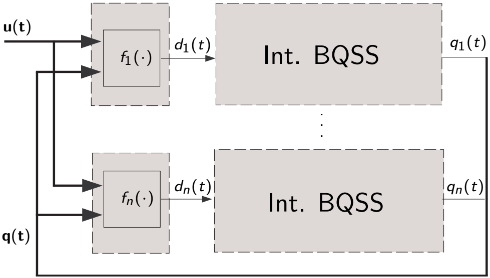
- The *block diagram structure* of BQSS is the same as that of QSS.



- The *static functions* are the same as for QSS.

BQSS and PowerDEVS

- The *block diagram structure* of BQSS is the same as that of QSS.



- The *static functions* are the same as for QSS.
- The *hysteretic quantized integrators* are similar, but:
  - they compute q_i in accordance with the algorithm just outlined,
  - when the calculated value of q_i does not agree with the sign of $\dot{x}_i$, they act as if $\dot{x}_i = 0$.

Stability and Error Bound of BQSS

BQSS is very similar to **QSS**, but it has an additional *perturbation* term that enforces that $\dot{x}_i = 0$ when a consistent value for q_i cannot be found.

$$\dot{\mathbf{x}}(t) = \mathbf{f}[\mathbf{q}(t), t] + \mathbf{\Delta f}(t) = \mathbf{f}[\mathbf{x}(t) + \mathbf{\Delta x}(t), t] + \mathbf{\Delta f}(t)$$

where $|\Delta x_i| \leq \Delta Q_i + \varepsilon_i$.

Stability and Error Bound of BQSS

BQSS is very similar to **QSS**, but it has an additional *perturbation* term that enforces that $\dot{x}_i = 0$ when a consistent value for q_i cannot be found.

$$\dot{\mathbf{x}}(t) = \mathbf{f}[\mathbf{q}(t), t] + \mathbf{\Delta f}(t) = \mathbf{f}[\mathbf{x}(t) + \mathbf{\Delta x}(t), t] + \mathbf{\Delta f}(t)$$

where $|\Delta x_i| \leq \Delta Q_i + \varepsilon_i$.

As a consequence:

Stability and Error Bound of BQSS

BQSS is very similar to **QSS**, but it has an additional *perturbation* term that enforces that $\dot{x}_i = 0$ when a consistent value for q_i cannot be found.

$$\dot{\mathbf{x}}(t) = \mathbf{f}[\mathbf{q}(t), t] + \mathbf{\Delta f}(t) = \mathbf{f}[\mathbf{x}(t) + \mathbf{\Delta x}(t), t] + \mathbf{\Delta f}(t)$$

where $|\Delta x_i| \leq \Delta Q_i + \varepsilon_i$.

As a consequence:

- The global error formula of **BQSS** for linear time-invariant stable systems now becomes:

$$|\mathbf{x_a}(t) - \mathbf{x}(t)| \leq |\mathbf{V}| \cdot |\mathbb{R}\mathrm{e}(\mathbf{\Lambda})^{-1}\mathbf{V}^{-1}| \cdot |\mathbf{A}| \cdot (\mathbf{\Delta Q} + \varepsilon) \quad ; \quad \forall t \geq 0$$

Stability and Error Bound of BQSS

BQSS is very similar to **QSS**, but it has an additional *perturbation* term that enforces that $\dot{x}_i = 0$ when a consistent value for q_i cannot be found.

$$\dot{\mathbf{x}}(t) = \mathbf{f}[\mathbf{q}(t), t] + \boldsymbol{\Delta}\mathbf{f}(t) = \mathbf{f}[\mathbf{x}(t) + \boldsymbol{\Delta}\mathbf{x}(t), t] + \boldsymbol{\Delta}\mathbf{f}(t)$$

where $|\Delta x_i| \leq \Delta Q_i + \varepsilon_i$.

As a consequence:

- The global error formula of **BQSS** for linear time-invariant stable systems now becomes:

$$|\mathbf{x_a}(t) - \mathbf{x}(t)| \leq |\mathbf{V}| \cdot |\mathbb{R}\text{e}(\boldsymbol{\Lambda})^{-1}\mathbf{V}^{-1}| \cdot |\mathbf{A}| \cdot (\boldsymbol{\Delta}\mathbf{Q} + \varepsilon) \quad ; \quad \forall t \geq 0$$

- The hysteresis width ε should now be chosen smaller than the quantum, so it does not introduce unnecessary errors.

Stability and Error Bound of BQSS

BQSS is very similar to **QSS**, but it has an additional *perturbation* term that enforces that $\dot{x}_i = 0$ when a consistent value for q_i cannot be found.

$$\dot{\mathbf{x}}(t) = \mathbf{f}[\mathbf{q}(t), t] + \mathbf{\Delta f}(t) = \mathbf{f}[\mathbf{x}(t) + \mathbf{\Delta x}(t), t] + \mathbf{\Delta f}(t)$$

where $|\Delta x_i| \leq \Delta Q_i + \varepsilon_i$.

As a consequence:

- The global error formula of **BQSS** for linear time-invariant stable systems now becomes:

$$|\mathbf{x_a}(t) - \mathbf{x}(t)| \leq |\mathbf{V}| \cdot |\mathbb{R}\text{e}(\mathbf{\Lambda})^{-1}\mathbf{V}^{-1}| \cdot |\mathbf{A}| \cdot (\mathbf{\Delta Q} + \varepsilon) \quad ; \quad \forall t \geq 0$$

- The hysteresis width ε should now be chosen smaller than the quantum, so it does not introduce unnecessary errors.
- The simulation trajectories are numerically stable for any quantum adopted.

Example: Second-order Linear System

- Eigenvalues

  $\lambda_1 = -0.01;$
  $\lambda_2 = -99.99$

Example: Second-order Linear System

- Eigenvalues

  $\lambda_1 = -0.01;$
  $\lambda_2 = -99.99$

- Initial states

  $x_1(0) = 0$; $x_2(0) = 20$

Example: Second-order Linear System

- Eigenvalues

  $\lambda_1 = -0.01;$
  $\lambda_2 = -99.99$

- Initial states

  $x_1(0) = 0;\ x_2(0) = 20$

- Quantization

  S1 $\Delta Q_{1,2} = 1$
  S2 $\Delta Q_{1,2} = 0.1$
  S3 $\Delta Q_{1,2} = 0.01$

Example: Second-order Linear System

- Eigenvalues

  $\lambda_1 = -0.01;$
  $\lambda_2 = -99.99$

- Initial states

  $x_1(0) = 0;\ x_2(0) = 20$

- Quantization

  S1 $\Delta Q_{1,2} = 1$
  S2 $\Delta Q_{1,2} = 0.1$
  S3 $\Delta Q_{1,2} = 0.01$

- Number of Events

  S1 x_1 : 20; x_2 : 22
  S2 x_1 : 201; x_2 : 201
  S3 x_1 : 2006; x_2 : 2024

Example: Second-order Linear System

- Eigenvalues

  $\lambda_1 = -0.01;$
  $\lambda_2 = -99.99$

- Initial states

  $x_1(0) = 0;\ x_2(0) = 20$

- Quantization

S1	$\Delta Q_{1,2} = 1$
S2	$\Delta Q_{1,2} = 0.1$
S3	$\Delta Q_{1,2} = 0.01$

- Number of Events

S1	x_1 : 20;	x_2 : 22
S2	x_1 : 201;	x_2 : 201
S3	x_1 : 2006;	x_2 : 2024

- Error Bound

S1	$e_1 \leq 3;$	$e_2 \leq 5$
S2	$e_1 \leq 0.3;$	$e_2 \leq 0.5$
S3	$e_1 \leq 0.03;$	$e_2 \leq 0.05$

Example: Second-order Linear System

- Eigenvalues

  $\lambda_1 = -0.01;$
  $\lambda_2 = -99.99$

- Initial states

  $x_1(0) = 0;\ x_2(0) = 20$

- Quantization

S1	$\Delta Q_{1,2} = 1$
S2	$\Delta Q_{1,2} = 0.1$
S3	$\Delta Q_{1,2} = 0.01$

- Number of Events

S1	x_1 : 20;	x_2 : 22
S2	x_1 : 201;	x_2 : 201
S3	x_1 : 2006;	x_2 : 2024

- Error Bound

S1	$e_1 \leq 3;$	$e_2 \leq 5$
S2	$e_1 \leq 0.3;$	$e_2 \leq 0.5$
S3	$e_1 \leq 0.03;$	$e_2 \leq 0.05$

$$\dot{x}_1(t) = 0.01\ x_2(t)$$
$$\dot{x}_2(t) = -100\ x_1(t) - 100\ x_2(t) + 2020$$

Example: Second-order Linear System

- Eigenvalues

  $\lambda_1 = -0.01;$
  $\lambda_2 = -99.99$

- Initial states

  $x_1(0) = 0;\ x_2(0) = 20$

- Quantization

  S1 $\Delta Q_{1,2} = 1$
  S2 $\Delta Q_{1,2} = 0.1$
  S3 $\Delta Q_{1,2} = 0.01$

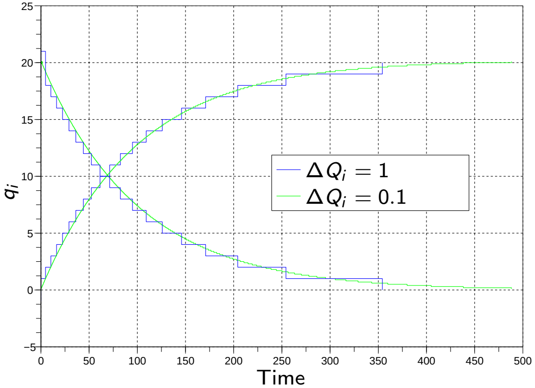
- Number of Events

  S1 x_1 : 20; x_2 : 22
  S2 x_1 : 201; x_2 : 201
  S3 x_1 : 2006; x_2 : 2024

- Error Bound

  S1 $e_1 \leq 3$; $e_2 \leq 5$
  S2 $e_1 \leq 0.3$; $e_2 \leq 0.5$
  S3 $e_1 \leq 0.03$; $e_2 \leq 0.05$

$$\dot{x}_1(t) = 0.01\ x_2(t)$$
$$\dot{x}_2(t) = -100\ x_1(t) - 100\ x_2(t) + 2020$$



Example: Second-order Linear System II

Some remarks:

Example: Second-order Linear System II

Some remarks:

- As there are no oscillations, the number of events is the division between each signal amplitude and the corresponding quantum.

Example: Second-order Linear System II

Some remarks:

- As there are no oscillations, the number of events is the division between each signal amplitude and the corresponding quantum.
- Consequently, the number of steps grows inversely proportional to the quantization.

Example: Second-order Linear System II

Some remarks:

- As there are no oscillations, the number of events is the division between each signal amplitude and the corresponding quantum.
- Consequently, the number of steps grows inversely proportional to the quantization.
- The simulation reaches a stable situation some time before $t = 500$ *seconds*, and after that, the simulation stops, as no further events are being scheduled.

Example: Second-order Linear System II

Some remarks:

- As there are no oscillations, the number of events is the division between each signal amplitude and the corresponding quantum.
- Consequently, the number of steps grows inversely proportional to the quantization.
- The simulation reaches a stable situation some time before $t = 500$ *seconds*, and after that, the simulation stops, as no further events are being scheduled.
- The theoretical error bound is conservative in this example.

Main Features of BQSS

Advantages:

Main Features of BQSS

Advantages:

- **BQSS** solves the problem of *high-frequency oscillations* exhibited when simulating stiff systems using explicit QSS methods.

Main Features of BQSS

Advantages:

- **BQSS** solves the problem of *high-frequency oscillations* exhibited when simulating stiff systems using explicit QSS methods.
- **BQSS** does not require *Newton iterations*, which permits the efficient numerical simulation of many stiff systems.

Main Features of BQSS

Advantages:

- **BQSS** solves the problem of *high-frequency oscillations* exhibited when simulating stiff systems using explicit QSS methods.
- **BQSS** does not require *Newton iterations*, which permits the efficient numerical simulation of many stiff systems.
- **BQSS** is also efficient for *discontinuity handling*.

Main Features of BQSS

Advantages:

- **BQSS** solves the problem of *high-frequency oscillations* exhibited when simulating stiff systems using explicit QSS methods.
- **BQSS** does not require *Newton iterations*, which permits the efficient numerical simulation of many stiff systems.
- **BQSS** is also efficient for *discontinuity handling*.
- **BQSS** can be used for *real-time simulation of stiff systems*.

Main Features of BQSS

Advantages:

- **BQSS** solves the problem of *high-frequency oscillations* exhibited when simulating stiff systems using explicit QSS methods.
- **BQSS** does not require *Newton iterations*, which permits the efficient numerical simulation of many stiff systems.
- **BQSS** is also efficient for *discontinuity handling*.
- **BQSS** can be used for *real-time simulation of stiff systems*.

Disadvantages:

Main Features of BQSS

Advantages:

- **BQSS** solves the problem of *high-frequency oscillations* exhibited when simulating stiff systems using explicit QSS methods.
- **BQSS** does not require *Newton iterations*, which permits the efficient numerical simulation of many stiff systems.
- **BQSS** is also efficient for *discontinuity handling*.
- **BQSS** can be used for *real-time simulation of stiff systems*.

Disadvantages:

- Spurious equilibrium points may appear in nonlinear systems due to the terms Δf_i.

Main Features of BQSS

Advantages:

- **BQSS** solves the problem of *high-frequency oscillations* exhibited when simulating stiff systems using explicit QSS methods.
- **BQSS** does not require *Newton iterations*, which permits the efficient numerical simulation of many stiff systems.
- **BQSS** is also efficient for *discontinuity handling*.
- **BQSS** can be used for *real-time simulation of stiff systems*.

Disadvantages:

- Spurious equilibrium points may appear in nonlinear systems due to the terms Δf_i.
- The number of steps grows linearly with the accuracy. Unfortunately, we have not found any way to extend the **BQSS** algorithm to higher orders of approximation accuracy.

LIQSS: Basic Idea

The *Linearly Implicit QSS (LIQSS)* algorithm is based on an idea very similar to that of BQSS.

LIQSS: Basic Idea

The *Linearly Implicit QSS (LIQSS)* algorithm is based on an idea very similar to that of BQSS.

- Just as in BQSS, we shall choose q_j such that it represents a *future value* of x_j.

LIQSS: Basic Idea

The *Linearly Implicit QSS (LIQSS)* algorithm is based on an idea very similar to that of BQSS.

- Just as in BQSS, we shall choose q_j such that it represents a *future value* of x_j.
- Just as in BQSS, we set $q_j(t) = x_j(t) \pm \Delta Q_j$ depending on the sign of $\dot{x}_j(t)$.

LIQSS: Basic Idea

The *Linearly Implicit QSS (LIQSS)* algorithm is based on an idea very similar to that of BQSS.

- Just as in BQSS, we shall choose q_j such that it represents a *future value* of x_j.
- Just as in BQSS, we set $q_j(t) = x_j(t) \pm \Delta Q_j$ depending on the sign of $\dot{x}_j(t)$.
- The main difference with BQSS is that, when we cannot find q_j such that x_j moves towards it, we now attempt to determine the value of $\hat{q}_j$ that makes $\dot{x}_j = 0$, instead of enforcing $\dot{x}_j = 0$ by adding a perturbation term Δf_j.

LIQSS: Basic Idea

The *Linearly Implicit QSS (LIQSS)* algorithm is based on an idea very similar to that of BQSS.

- Just as in BQSS, we shall choose q_j such that it represents a *future value* of x_j.
- Just as in BQSS, we set $q_j(t) = x_j(t) \pm \Delta Q_j$ depending on the sign of $\dot{x}_j(t)$.
- The main difference with BQSS is that, when we cannot find q_j such that x_j moves towards it, we now attempt to determine the value of $\hat{q}_j$ that makes $\dot{x}_j = 0$, instead of enforcing $\dot{x}_j = 0$ by adding a perturbation term Δf_j.
- The search for $\hat{q}_j$ is done in a linear manner.

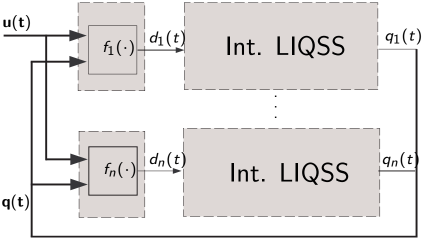
LIQSS: Basic Idea

The *Linearly Implicit QSS (LIQSS)* algorithm is based on an idea very similar to that of BQSS.

- Just as in BQSS, we shall choose q_j such that it represents a *future value* of x_j.
- Just as in BQSS, we set $q_j(t) = x_j(t) \pm \Delta Q_j$ depending on the sign of $\dot{x}_j(t)$.
- The main difference with BQSS is that, when we cannot find q_j such that x_j moves towards it, we now attempt to determine the value of $\hat{q}_j$ that makes $\dot{x}_j = 0$, instead of enforcing $\dot{x}_j = 0$ by adding a perturbation term Δf_j.
- The search for $\hat{q}_j$ is done in a linear manner.
- Contrary to BQSS, once we have computed a new value for q_j, no further internal transition will be scheduled, until x_j has advanced by $\pm\Delta Q_j$, even if the sign of $\dot{x}_j$ changes during the step.

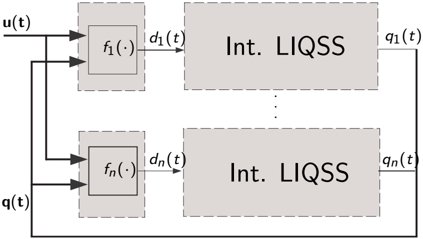
LIQSS and PowerDEVS

- The *block diagram structure* of LIQSS is the same as that of QSS and BQSS.



LIQSS and PowerDEVS

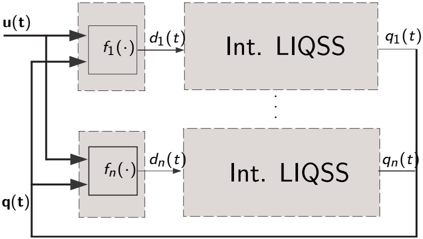
- The *block diagram structure* of LIQSS is the same as that of QSS and BQSS.



- The *static functions* are the same as for QSS and BQSS.

LIQSS and PowerDEVS

- The *block diagram structure* of LIQSS is the same as that of QSS and BQSS.



- The *static functions* are the same as for QSS and BQSS.
- The *hysteretic quantized integrators* are similar, but:
  - before they output a new value for q_i, they estimate the value of the derivative $\dot{x}_i$ that each of the two options would produce;
  - if one of them has the correct sign, i.e., $x_i \rightarrow q_i$, that value is chosen;
  - otherwise, the value $\hat{q}_i$ is estimated that makes $\dot{x}_i = 0$.

Stability and Error Bound of LIQSS

Given the system:

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

Stability and Error Bound of LIQSS

Given the system:

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

Its **LIQSS** approximation is governed by the same expression as that of QSS:

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{q}(t), \mathbf{u}(t)) = \mathbf{f}(\mathbf{x}(t) + \mathbf{\Delta x}(t), \mathbf{u}(t))$$

Stability and Error Bound of LIQSS

Given the system:

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

Its **LIQSS** approximation is governed by the same expression as that of QSS:

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{q}(t), \mathbf{u}(t)) = \mathbf{f}(\mathbf{x}(t) + \boldsymbol{\Delta}\mathbf{x}(t), \mathbf{u}(t))$$

However, q_i can now differ from x_i by an amount of $2 \cdot \Delta Q_i$ (after sign changes in $\dot{x}_i$).

Stability and Error Bound of LIQSS

Given the system:

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

Its **LIQSS** approximation is governed by the same expression as that of QSS:

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{q}(t), \mathbf{u}(t)) = \mathbf{f}(\mathbf{x}(t) + \boldsymbol{\Delta}\mathbf{x}(t), \mathbf{u}(t))$$

However, q_i can now differ from x_i by an amount of $2 \cdot \Delta Q_i$ (after sign changes in $\dot{x}_i$).

The properties of LIQSS are almost identical to those of QSS, but the error bound is twice the error bound of QSS.

Example: Second-order Linear System

- Eigenvalues

  $\lambda_1 = -0.01;$
  $\lambda_2 = -99.99$

Example: Second-order Linear System

- Eigenvalues

  $\lambda_1 = -0.01;$
  $\lambda_2 = -99.99$

- Initial states

  $x_1(0) = 0;\ x_2(0) = 20$

Example: Second-order Linear System

- Eigenvalues

  $\lambda_1 = -0.01;$
  $\lambda_2 = -99.99$

- Initial states

  $x_1(0) = 0;\ x_2(0) = 20$

- Quantization
  S1 $\Delta Q_{1,2} = 1$
  S2 $\Delta Q_{1,2} = 0.1$
  S3 $\Delta Q_{1,2} = 0.01$

Example: Second-order Linear System

- Eigenvalues

  $\lambda_1 = -0.01;$
  $\lambda_2 = -99.99$

- Initial states

  $x_1(0) = 0;\ x_2(0) = 20$

- Quantization

S1	$\Delta Q_{1,2} = 1$
S2	$\Delta Q_{1,2} = 0.1$
S3	$\Delta Q_{1,2} = 0.01$

- Number of Events

S1	x_1 : 21;	x_2 : 25
S2	x_1 : 201;	x_2 : 203
S3	x_1 : 2006;	x_2 : 2026

Example: Second-order Linear System

- Eigenvalues

  $\lambda_1 = -0.01;$
  $\lambda_2 = -99.99$

- Initial states

  $x_1(0) = 0;\ x_2(0) = 20$

- Quantization

  S1 $\Delta Q_{1,2} = 1$
  S2 $\Delta Q_{1,2} = 0.1$
  S3 $\Delta Q_{1,2} = 0.01$

- Number of Events

  S1 $x_1 : 21;$ $x_2 : 25$
  S2 $x_1 : 201;$ $x_2 : 203$
  S3 $x_1 : 2006;$ $x_2 : 2026$

- Error Bound

  S1 $e_1 \leq 2;$ $e_2 \leq 6$
  S2 $e_1 \leq 0.2;$ $e_2 \leq 0.6$
  S3 $e_1 \leq 0.02;$ $e_2 \leq 0.06$

Example: Second-order Linear System

- Eigenvalues

  $\lambda_1 = -0.01;$
  $\lambda_2 = -99.99$

- Initial states

  $x_1(0) = 0;\ x_2(0) = 20$

- Quantization

S1	$\Delta Q_{1,2} = 1$
S2	$\Delta Q_{1,2} = 0.1$
S3	$\Delta Q_{1,2} = 0.01$

- Number of Events

S1	x_1 : 21;	x_2 : 25
S2	x_1 : 201;	x_2 : 203
S3	x_1 : 2006;	x_2 : 2026

- Error Bound

S1	$e_1 \leq 2;$	$e_2 \leq 6$
S2	$e_1 \leq 0.2;$	$e_2 \leq 0.6$
S3	$e_1 \leq 0.02;$	$e_2 \leq 0.06$

$$\dot{x}_1(t) = 0.01\ x_2(t)$$
$$\dot{x}_2(t) = -100\ x_1(t) - 100\ x_2(t) + 2020$$

Example: Second-order Linear System

- Eigenvalues

  $\lambda_1 = -0.01;$
  $\lambda_2 = -99.99$

- Initial states

  $x_1(0) = 0;\ x_2(0) = 20$

- Quantization

  S1 $\Delta Q_{1,2} = 1$
  S2 $\Delta Q_{1,2} = 0.1$
  S3 $\Delta Q_{1,2} = 0.01$

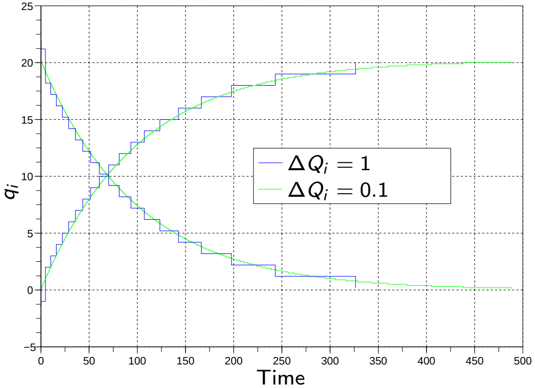
- Number of Events

  S1 x_1 : 21; x_2 : 25
  S2 x_1 : 201; x_2 : 203
  S3 x_1 : 2006; x_2 : 2026

- Error Bound

  S1 $e_1 \leq 2;$ $e_2 \leq 6$
  S2 $e_1 \leq 0.2;$ $e_2 \leq 0.6$
  S3 $e_1 \leq 0.02;$ $e_2 \leq 0.06$

$$\dot{x}_1(t) = 0.01\ x_2(t)$$
$$\dot{x}_2(t) = -100\ x_1(t) - 100\ x_2(t) + 2020$$



Example: Second-order Linear System II

Some remarks:

Example: Second-order Linear System II

Some remarks:

- As there are no oscillations, the number of events is the division between each signal amplitude and the corresponding quantum.

Example: Second-order Linear System II

Some remarks:

- As there are no oscillations, the number of events is the division between each signal amplitude and the corresponding quantum.
- Consequently, the number of steps grows inversely proportional to the quantization.

Example: Second-order Linear System II

Some remarks:

- As there are no oscillations, the number of events is the division between each signal amplitude and the corresponding quantum.
- Consequently, the number of steps grows inversely proportional to the quantization.
- The simulation reaches a stable situation some time before $t = 500$ *seconds*, and after that, the simulation stops, as no further events are being scheduled.

Example: Second-order Linear System II

Some remarks:

- As there are no oscillations, the number of events is the division between each signal amplitude and the corresponding quantum.
- Consequently, the number of steps grows inversely proportional to the quantization.
- The simulation reaches a stable situation some time before $t = 500\ seconds$, and after that, the simulation stops, as no further events are being scheduled.
- The theoretical error bound is conservative in this example.

Main Features of LIQSS

Advantages:

Main Features of LIQSS

Advantages:

- **LIQSS** shares the main advantages of **BQSS**.

Main Features of LIQSS

Advantages:

- **LIQSS** shares the main advantages of **BQSS**.
- **LIQSS** solves the problem of spurious equilibrium points of **BQSS**.

Main Features of LIQSS

Advantages:

- **LIQSS** shares the main advantages of **BQSS**.
- **LIQSS** solves the problem of spurious equilibrium points of **BQSS**.

Disadvantages:

Main Features of LIQSS

Advantages:

- **LIQSS** shares the main advantages of **BQSS**.
- **LIQSS** solves the problem of spurious equilibrium points of **BQSS**.

Disadvantages:

- The number of steps still grows linearly with the accuracy. However, it is possible to obtain *higher-order LIQSS methods*.

Main Features of LIQSS

Advantages:

- **LIQSS** shares the main advantages of **BQSS**.
- **LIQSS** solves the problem of spurious equilibrium points of **BQSS**.

Disadvantages:

- The number of steps still grows linearly with the accuracy. However, it is possible to obtain *higher-order LIQSS methods*.
- **LIQSS** only looks at the main diagonal of the Jacobian matrix, allowing it sometimes to happen that $\dot{x}_j$ changes its direction without causing a modification of q_j. In such cases, **LIQSS** can provoke oscillations depending on the system structure, i.e., the algorithm may *lose its stiffly-stable nature*.

LIQSS2: Basic Idea

- **LIQSS2** is similar to **QSS2**. The quantized variables $q_j(t)$ are *piecewise linear*.

LIQSS2: Basic Idea

- **LIQSS2** is similar to **QSS2**. The quantized variables $q_j(t)$ are *piecewise linear*.
- Each section of $q_j(t)$ starts at $x_j(t) \pm \Delta Q_j$ with a slope of $\dot{q}_j(t) = \dot{x}_j(t)|_{q_j(t)}$.

LIQSS2: Basic Idea

- **LIQSS2** is similar to **QSS2**. The quantized variables $q_j(t)$ are *piecewise linear*.
- Each section of $q_j(t)$ starts at $x_j(t) \pm \Delta Q_j$ with a slope of $\dot{q}_j(t) = \dot{x}_j(t)|_{q_j(t)}$.
- We shall attempt to ensure, just as in the case of **LIQSS**, that $x_j(t)$ moves toward $q_j(t)$. That is, we demand that $\ddot{x}_j(t) \cdot (q_j(t) - x_j(t)) > 0$.

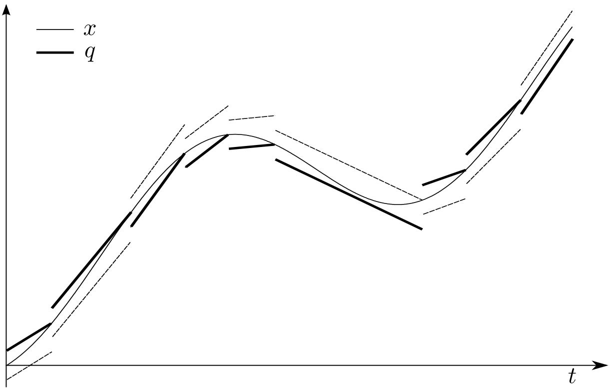
LIQSS2: Basic Idea

- **LIQSS2** is similar to **QSS2**. The quantized variables $q_j(t)$ are *piecewise linear*.
- Each section of $q_j(t)$ starts at $x_j(t) \pm \Delta Q_j$ with a slope of $\dot{q}_j(t) = \dot{x}_j(t)|_{q_j(t)}$.
- We shall attempt to ensure, just as in the case of **LIQSS**, that $x_j(t)$ moves toward $q_j(t)$. That is, we demand that $\ddot{x}_j(t) \cdot (q_j(t) - x_j(t)) > 0$.
- When this is not possible, we shall use a linear approximation to find the value $\hat{q}_j(t)$ with slope $\dot{q}_j(t) = \dot{x}_j(t)|_{\hat{q}_j(t)}$, for which $\ddot{x}_j(t) = 0$.

LIQSS2: Basic Idea

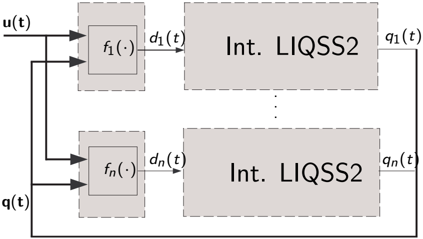
- **LIQSS2** is similar to **QSS2**. The quantized variables $q_j(t)$ are *piecewise linear*.
- Each section of $q_j(t)$ starts at $x_j(t) \pm \Delta Q_j$ with a slope of $\dot{q}_j(t) = \dot{x}_j(t)|_{q_j(t)}$.
- We shall attempt to ensure, just as in the case of **LIQSS**, that $x_j(t)$ moves toward $q_j(t)$. That is, we demand that $\ddot{x}_j(t) \cdot (q_j(t) - x_j(t)) > 0$.
- When this is not possible, we shall use a linear approximation to find the value $\hat{q}_j(t)$ with slope $\dot{q}_j(t) = \dot{x}_j(t)|_{\hat{q}_j(t)}$, for which $\ddot{x}_j(t) = 0$.
- Although this is an implicit algorithm, there is no need for Newton iterations to take place, as only three possible trajectories can be chosen for $q_j(t)$.

LIQSS2: Trajectories



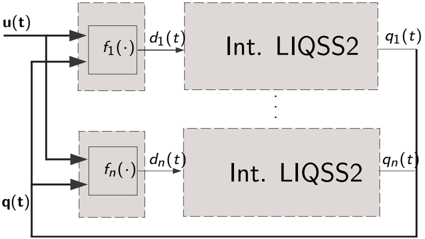
LIQSS2 and PowerDEVS

- The *block diagram structure* of LIQSS2 is the same as that of QSS2.



LIQSS2 and PowerDEVS

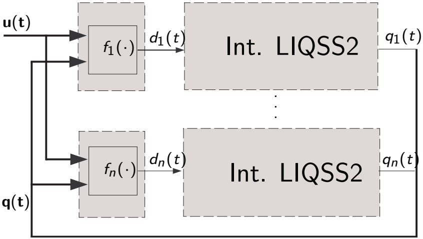
- The *block diagram structure* of LIQSS2 is the same as that of QSS2.



- The *static functions* are the same as for QSS2.

LIQSS2 and PowerDEVS

- The *block diagram structure* of LIQSS2 is the same as that of QSS2.



- The *static functions* are the same as for QSS2.
- The *hysteretic quantized integrators* are similar, but:
  - before they output a new value for q_i, they estimate the value of the second derivative $\ddot{x}_i$ that each of the two options would produce;
  - if one of them has the correct sign, i.e., $x_i \rightarrow q_i$, that value is chosen;
  - otherwise, the value $\hat{q}_i$ is estimated that makes $\ddot{x}_i = 0$.

Stability and Error Bound of LIQSS2

Given the system:

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

Stability and Error Bound of LIQSS2

Given the system:

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

Its **LIQSS2** approximation is governed by the same expression as that of **QSS**:

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{q}(t), \mathbf{u}(t)) = \mathbf{f}(\mathbf{x}(t) + \mathbf{\Delta x}(t), \mathbf{u}(t))$$

Stability and Error Bound of LIQSS2

Given the system:

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

Its **LIQSS2** approximation is governed by the same expression as that of **QSS**:

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{q}(t), \mathbf{u}(t)) = \mathbf{f}(\mathbf{x}(t) + \mathbf{\Delta x}(t), \mathbf{u}(t))$$

However, just as in the case of **LIQSS**, q_i can now differ from x_i by an amount of $2 \cdot \Delta Q_i$ (after sign changes in $\ddot{x}_i$).

Stability and Error Bound of LIQSS2

Given the system:

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

Its **LIQSS2** approximation is governed by the same expression as that of **QSS**:

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{q}(t), \mathbf{u}(t)) = \mathbf{f}(\mathbf{x}(t) + \mathbf{\Delta x}(t), \mathbf{u}(t))$$

However, just as in the case of **LIQSS**, q_i can now differ from x_i by an amount of $2 \cdot \Delta Q_i$ (after sign changes in $\ddot{x}_i$).

The properties of LIQSS2 are identical to those of LIQSS.

Example: Second-order Linear System

- Eigenvalues

  $\lambda_1 = -0.01;$
  $\lambda_2 = -99.99$

Example: Second-order Linear System

- Eigenvalues

  $\lambda_1 = -0.01;$
  $\lambda_2 = -99.99$

- Initial states

  $x_1(0) = 0;\ x_2(0) = 20$

Example: Second-order Linear System

- Eigenvalues

  $\lambda_1 = -0.01;$
  $\lambda_2 = -99.99$

- Initial states

  $x_1(0) = 0;\ x_2(0) = 20$

- Quantization
  S1 $\Delta Q_{1,2} = 1$
  S2 $\Delta Q_{1,2} = 0.1$
  S3 $\Delta Q_{1,2} = 0.01$

Example: Second-order Linear System

- Eigenvalues

  $\lambda_1 = -0.01;$
  $\lambda_2 = -99.99$

- Initial states

  $x_1(0) = 0;\ x_2(0) = 20$

- Quantization

  S1 $\Delta Q_{1,2} = 1$
  S2 $\Delta Q_{1,2} = 0.1$
  S3 $\Delta Q_{1,2} = 0.01$

- Number of Events

  S1 $x_1 : 5;$ $x_2 : 8$
  S2 $x_1 : 18;$ $x_2 : 22$
  S3 $x_1 : 57;$ $x_2 : 65$

Example: Second-order Linear System

- Eigenvalues

  $\lambda_1 = -0.01;$
  $\lambda_2 = -99.99$

- Initial states

  $x_1(0) = 0;\ x_2(0) = 20$

- Quantization

S1	$\Delta Q_{1,2} = 1$
S2	$\Delta Q_{1,2} = 0.1$
S3	$\Delta Q_{1,2} = 0.01$

- Number of Events

S1	$x_1 : 5;$	$x_2 : 8$
S2	$x_1 : 18;$	$x_2 : 22$
S3	$x_1 : 57;$	$x_2 : 65$

- Error Bound

S1	$e_1 \leq 2;$	$e_2 \leq 6$
S2	$e_1 \leq 0.2;$	$e_2 \leq 0.6$
S3	$e_1 \leq 0.02;$	$e_2 \leq 0.06$

Example: Second-order Linear System

- Eigenvalues

  $\lambda_1 = -0.01;$
  $\lambda_2 = -99.99$

- Initial states

  $x_1(0) = 0;\ x_2(0) = 20$

- Quantization

S1	$\Delta Q_{1,2} = 1$
S2	$\Delta Q_{1,2} = 0.1$
S3	$\Delta Q_{1,2} = 0.01$

- Number of Events

S1	x_1 : 5;	x_2 : 8
S2	x_1 : 18;	x_2 : 22
S3	x_1 : 57;	x_2 : 65

- Error Bound

S1	$e_1 \leq 2;$	$e_2 \leq 6$
S2	$e_1 \leq 0.2;$	$e_2 \leq 0.6$
S3	$e_1 \leq 0.02;$	$e_2 \leq 0.06$

$$\dot{x}_1(t) = 0.01\ x_2(t)$$
$$\dot{x}_2(t) = -100\ x_1(t) - 100\ x_2(t) + 2020$$

Example: Second-order Linear System

- Eigenvalues

  $\lambda_1 = -0.01;$
  $\lambda_2 = -99.99$

- Initial states

  $x_1(0) = 0;\ x_2(0) = 20$

- Quantization

  S1 $\Delta Q_{1,2} = 1$
  S2 $\Delta Q_{1,2} = 0.1$
  S3 $\Delta Q_{1,2} = 0.01$

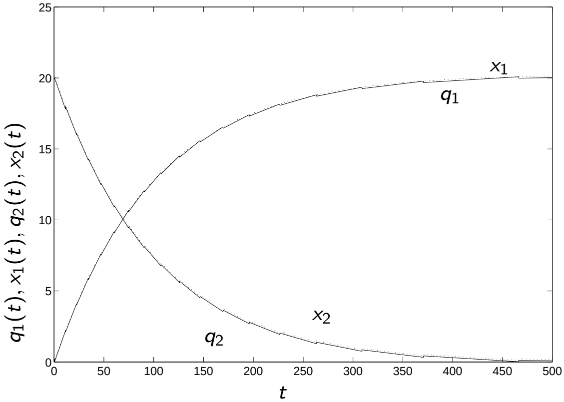
- Number of Events

  S1 $x_1 : 5;$ $x_2 : 8$
  S2 $x_1 : 18;$ $x_2 : 22$
  S3 $x_1 : 57;$ $x_2 : 65$

- Error Bound

  S1 $e_1 \leq 2;$ $e_2 \leq 6$
  S2 $e_1 \leq 0.2;$ $e_2 \leq 0.6$
  S3 $e_1 \leq 0.02;$ $e_2 \leq 0.06$

$$\dot{x}_1(t) = 0.01\ x_2(t)$$
$$\dot{x}_2(t) = -100\ x_1(t) - 100\ x_2(t) + 2020$$



Example: Second-order Linear System II

Some remarks:

Example: Second-order Linear System II

Some remarks:

- The number of steps grows inversely proportional to the *square root of the quantization*.

Example: Second-order Linear System II

Some remarks:

- The number of steps grows inversely proportional to the *square root of the quantization*.
- The theoretical error bound is conservative in this example.

Main Features of LIQSS2

Advantages:

Main Features of LIQSS2

Advantages:

- **LIQSS2** shares all the advantages of **LIQSS**.

Main Features of LIQSS2

Advantages:

- **LIQSS2** shares all the advantages of **LIQSS**.
- **LIQSS2** is *second-order accurate*.

Main Features of LIQSS2

Advantages:

- **LIQSS2** shares all the advantages of **LIQSS**.
- **LIQSS2** is *second-order accurate*.
- The number of steps grows with the *square root of the accuracy*.

Main Features of LIQSS2

Advantages:

- **LIQSS2** shares all the advantages of **LIQSS**.
- **LIQSS2** is *second-order accurate*.
- The number of steps grows with the *square root of the accuracy*.
- Recent results indicate that **LIQSS2** is particularly well suited for simulating *power electronics circuits*.

Main Features of LIQSS2

Advantages:

- **LIQSS2** shares all the advantages of **LIQSS**.
- **LIQSS2** is *second-order accurate*.
- The number of steps grows with the *square root of the accuracy*.
- Recent results indicate that **LIQSS2** is particularly well suited for simulating *power electronics circuits*.

Disadvantages:

Main Features of LIQSS2

Advantages:

- **LIQSS2** shares all the advantages of **LIQSS**.
- **LIQSS2** is *second-order accurate*.
- The number of steps grows with the *square root of the accuracy*.
- Recent results indicate that **LIQSS2** is particularly well suited for simulating *power electronics circuits*.

Disadvantages:

- **LIQSS2** only looks at the main diagonal of the Jacobian matrix, allowing it sometimes to happen that $\ddot{x}_j$ changes its direction without causing a modification of q_j. In such cases, **LIQSS2** can provoke oscillations depending on the system structure, i.e., the algorithm may *lose its stiffly-stable nature*.

LIQSS3: Basic Idea

- **LIQSS3** is similar to **QSS3**. The quantized variables $q_j(t)$ are *piecewise parabolic*.

LIQSS3: Basic Idea

- **LIQSS3** is similar to **QSS3**. The quantized variables $q_j(t)$ are *piecewise parabolic*.
- Each section of $q_j(t)$ starts at $x_j(t) \pm \Delta Q_j$ with a slope of $\dot{q}_j(t) = \dot{x}_j(t)|_{q_j(t)}$ and a second derivative of $\ddot{q}_j(t) = \ddot{x}_j(t)|_{q_j(t)}$.

LIQSS3: Basic Idea

- **LIQSS3** is similar to **QSS3**. The quantized variables $q_j(t)$ are *piecewise parabolic*.
- Each section of $q_j(t)$ starts at $x_j(t) \pm \Delta Q_j$ with a slope of $\dot{q}_j(t) = \dot{x}_j(t)|_{q_j(t)}$ and a second derivative of $\ddot{q}_j(t) = \ddot{x}_j(t)|_{q_j(t)}$.
- We shall attempt to ensure, just as in the cases of **LIQSS** and **LIQSS2**, that $x_j(t)$ moves toward $q_j(t)$. That is, we demand that $\dddot{x}_j(t) \cdot (q_j(t) - x_j(t)) > 0$.

LIQSS3: Basic Idea

- **LIQSS3** is similar to **QSS3**. The quantized variables $q_j(t)$ are *piecewise parabolic*.

- Each section of $q_j(t)$ starts at $x_j(t) \pm \Delta Q_j$ with a slope of $\dot{q}_j(t) = \dot{x}_j(t)|_{q_j(t)}$ and a second derivative of $\ddot{q}_j(t) = \ddot{x}_j(t)|_{q_j(t)}$.

- We shall attempt to ensure, just as in the cases of **LIQSS** and **LIQSS2**, that $x_j(t)$ moves toward $q_j(t)$. That is, we demand that $\dddot{x}_j(t) \cdot (q_j(t) - x_j(t)) > 0$.

- When this is not possible, we shall use a linear approximation to find the value $\hat{q}_j(t)$ with slope $\dot{q}_j(t) = \dot{x}_j(t)|_{\hat{q}_j(t)}$ and with second derivative $\ddot{q}_j(t) = \ddot{x}_j(t)|_{\hat{q}_j(t)}$, for which $\dddot{x}_j(t) = 0$.

LIQSS3: Basic Idea

- **LIQSS3** is similar to **QSS3**. The quantized variables $q_j(t)$ are *piecewise parabolic*.
- Each section of $q_j(t)$ starts at $x_j(t) \pm \Delta Q_j$ with a slope of $\dot{q}_j(t) = \dot{x}_j(t)|_{q_j(t)}$ and a second derivative of $\ddot{q}_j(t) = \ddot{x}_j(t)|_{q_j(t)}$.
- We shall attempt to ensure, just as in the cases of **LIQSS** and **LIQSS2**, that $x_j(t)$ moves toward $q_j(t)$. That is, we demand that $\dddot{x}_j(t) \cdot (q_j(t) - x_j(t)) > 0$.
- When this is not possible, we shall use a linear approximation to find the value $\hat{q}_j(t)$ with slope $\dot{q}_j(t) = \dot{x}_j(t)|_{\hat{q}_j(t)}$ and with second derivative $\ddot{q}_j(t) = \ddot{x}_j(t)|_{\hat{q}_j(t)}$, for which $\dddot{x}_j(t) = 0$.
- Although this is an implicit algorithm, there is no need for Newton iterations to take place, as only three possible trajectories can be chosen for $q_j(t)$.

Main Features of LIQSS3

Advantages:

Main Features of LIQSS3

Advantages:

- **LIQSS3** shares all the advantages of **LIQSS2**.

Main Features of LIQSS3

Advantages:

- **LIQSS3** shares all the advantages of **LIQSS2**.
- **LIQSS3** is *third-order accurate*.

Main Features of LIQSS3

Advantages:

- **LIQSS3** shares all the advantages of **LIQSS2**.
- **LIQSS3** is *third-order accurate*.
- The number of steps grows with the *cubic root of the accuracy*.

Main Features of LIQSS3

Advantages:

- **LIQSS3** shares all the advantages of **LIQSS2**.
- **LIQSS3** is *third-order accurate*.
- The number of steps grows with the *cubic root of the accuracy*.
- Recent results indicate that **LIQSS3** is particularly well suited for simulating *power electronics circuits*.

Main Features of LIQSS3

Advantages:

- **LIQSS3** shares all the advantages of **LIQSS2**.
- **LIQSS3** is *third-order accurate*.
- The number of steps grows with the *cubic root of the accuracy*.
- Recent results indicate that **LIQSS3** is particularly well suited for simulating *power electronics circuits*.

Disadvantages:

Main Features of LIQSS3

Advantages:

- **LIQSS3** shares all the advantages of **LIQSS2**.
- **LIQSS3** is *third-order accurate*.
- The number of steps grows with the *cubic root of the accuracy*.
- Recent results indicate that **LIQSS3** is particularly well suited for simulating *power electronics circuits*.

Disadvantages:

- **LIQSS3** only looks at the main diagonal of the Jacobian matrix, allowing it sometimes to happen that $\dddot{x}_j$ changes its direction without causing a modification of q_j. In such cases, **LIQSS3** can provoke oscillations depending on the system structure, i.e., the algorithm may *lose its stiffly-stable nature*.

Marginally Stable Systems

The following system is known as the *harmonic oscillator*:

$$\dot{x}_1(t) = x_2(t)$$
$$\dot{x}_2(t) = -x_1(t)$$

The harmonic oscillator is the simplest marginally stable system.

Marginally Stable Systems

The following system is known as the *harmonic oscillator*:

$$\dot{x}_1(t) = x_2(t)$$
$$\dot{x}_2(t) = -x_1(t)$$

The harmonic oscillator is the simplest marginally stable system.

If we consider, for instance, the initial states $x_1(0) = 4$ and $x_2(0) = 0$, we obtain the analytical solution:

$$x_1(t) = 4\cos(t)$$
$$x_2(t) = 4\sin(t)$$

Marginally Stable Systems

The following system is known as the *harmonic oscillator*:

$$\dot{x}_1(t) = x_2(t)$$
$$\dot{x}_2(t) = -x_1(t)$$

The harmonic oscillator is the simplest marginally stable system.

If we consider, for instance, the initial states $x_1(0) = 4$ and $x_2(0) = 0$, we obtain the analytical solution:

$$x_1(t) = 4\cos(t)$$
$$x_2(t) = 4\sin(t)$$

We already know that *F-stable methods* are needed to properly integrate such systems.

Marginally Stable Systems

The following system is known as the *harmonic oscillator*:

$$\dot{x}_1(t) = x_2(t)$$
$$\dot{x}_2(t) = -x_1(t)$$

The harmonic oscillator is the simplest marginally stable system.

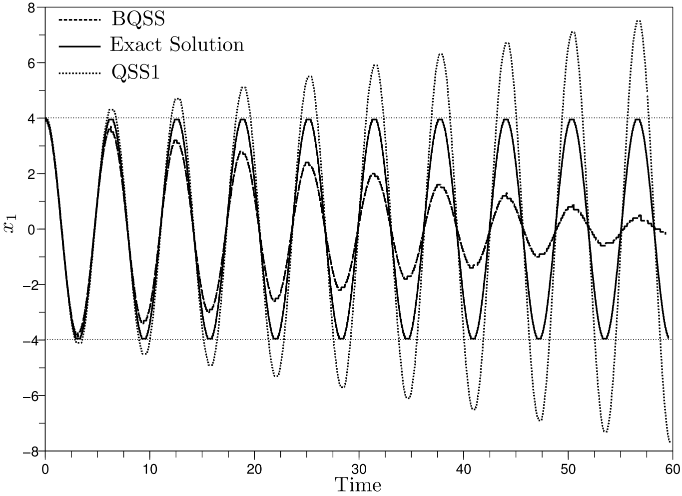
If we consider, for instance, the initial states $x_1(0) = 4$ and $x_2(0) = 0$, we obtain the analytical solution:

$$x_1(t) = 4\cos(t)$$
$$x_2(t) = 4\sin(t)$$

We already know that *F-stable methods* are needed to properly integrate such systems.

Let us check what happen with QSS and BQSS.

QSS, BQSS and Marginally Stable Systems



QSS, BQSS and Marginally Stable Systems II

The results are consistent with what we know about classic time-slicing methods:

QSS, BQSS and Marginally Stable Systems II

The results are consistent with what we know about classic time-slicing methods:

- The numerical solution with the *forward QSS* method gains energy and generates output trajectories that are numerically unstable, just as *Forward Euler* would.

QSS, BQSS and Marginally Stable Systems II

The results are consistent with what we know about classic time-slicing methods:

- The numerical solution with the *forward QSS* method gains energy and generates output trajectories that are numerically unstable, just as *Forward Euler* would.
- The numerical solution with the *backward QSS* method loses energy and generates output trajectories that are asymptotically stable, just as *Backward Euler* would.

QSS, BQSS and Marginally Stable Systems II

The results are consistent with what we know about classic time-slicing methods:

- The numerical solution with the *forward QSS* method gains energy and generates output trajectories that are numerically unstable, just as *Forward Euler* would.
- The numerical solution with the *backward QSS* method loses energy and generates output trajectories that are asymptotically stable, just as *Backward Euler* would.

Evidently, neither of the two methods is suited for the simulation of marginally stable systems.

Centered QSS Method

The simplest F-stable time-slicing method is the trapezoidal rule, which combines Forward and Backward Euler. So, it may make sense to try some combination of QSS and BQSS.

Centered QSS Method

The simplest F-stable time-slicing method is the trapezoidal rule, which combines Forward and Backward Euler. So, it may make sense to try some combination of QSS and BQSS.

Recall that:

Centered QSS Method

The simplest F-stable time-slicing method is the trapezoidal rule, which combines Forward and Backward Euler. So, it may make sense to try some combination of QSS and BQSS.

Recall that:

- In QSS, we take q_j as the *last* quantized value intersected by x_j.

Centered QSS Method

The simplest F-stable time-slicing method is the trapezoidal rule, which combines Forward and Backward Euler. So, it may make sense to try some combination of QSS and BQSS.

Recall that:

- In QSS, we take q_j as the *last* quantized value intersected by x_j.
- In BQSS, we take q_j as the *next* quantized value intersected by x_j.

Centered QSS Method

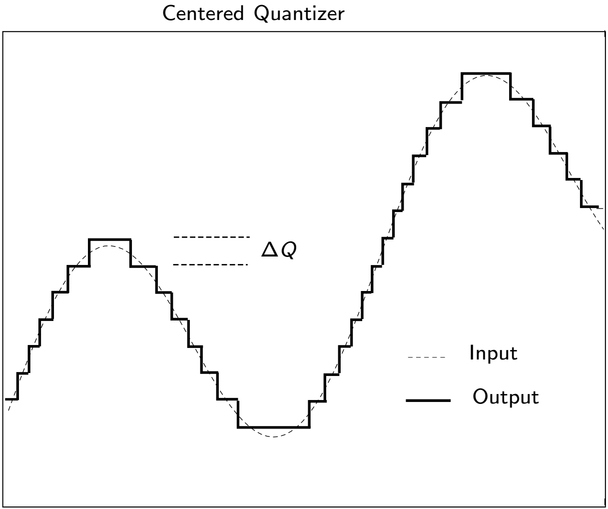
The simplest F-stable time-slicing method is the trapezoidal rule, which combines Forward and Backward Euler. So, it may make sense to try some combination of QSS and BQSS.

Recall that:

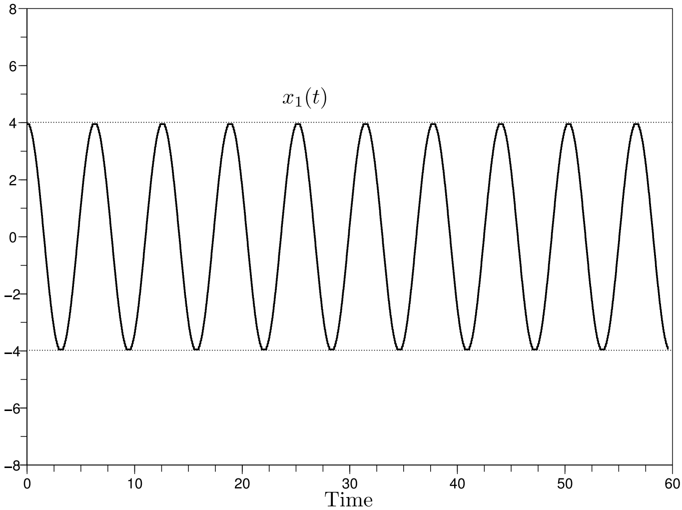
- In QSS, we take q_j as the *last* quantized value intersected by x_j.
- In BQSS, we take q_j as the *next* quantized value intersected by x_j.

Let us see what happens if we take q_j as the *mean* value between the last and the next quantized values, i.e., if we choose q_j in the middle of the quantization interval. We shall call this new method *Centered QSS (CQSS) solver*.

Centered Quantization



CQSS Simulation of the Harmonic Oscillator



CQSS Features

- **CQSS** preserves the periodical behavior of the harmonic oscillator.

CQSS Features

- **CQSS** preserves the periodical behavior of the harmonic oscillator.
- It has been proven that **CQSS** preserves the *geometrical properties* of general *Hamiltonian (conservative, frictionless) systems*: the method is *symmetric*, *symplectic*, and it preserves *ρ-reversibility*.

CQSS Features

- **CQSS** preserves the periodical behavior of the harmonic oscillator.
- It has been proven that **CQSS** preserves the *geometrical properties* of general *Hamiltonian (conservative, frictionless) systems*: the method is *symmetric*, *symplectic*, and it preserves *ρ-reversibility*.
- Just like **QSS** and **BQSS**, the method does not call for Newton iterations. In fact, it is the only known solver that exhibits *F-stability* while terminating the computations associated with a single step in a fixed amount of real time that can be calculated beforehand.

CQSS Features

- **CQSS** preserves the periodical behavior of the harmonic oscillator.
- It has been proven that **CQSS** preserves the *geometrical properties* of general *Hamiltonian (conservative, frictionless) systems*: the method is *symmetric*, *symplectic*, and it preserves *ρ-reversibility*.
- Just like **QSS** and **BQSS**, the method does not call for Newton iterations. In fact, it is the only known solver that exhibits *F-stability* while terminating the computations associated with a single step in a fixed amount of real time that can be calculated beforehand.
- Consequently, **CQSS** is well suited for *real-time simulation of marginally stable systems* with low to modest accuracy requirements.

CQSS Features

- **CQSS** preserves the periodical behavior of the harmonic oscillator.
- It has been proven that **CQSS** preserves the *geometrical properties* of general *Hamiltonian (conservative, frictionless) systems*: the method is *symmetric*, *symplectic*, and it preserves *ρ-reversibility*.
- Just like **QSS** and **BQSS**, the method does not call for Newton iterations. In fact, it is the only known solver that exhibits *F-stability* while terminating the computations associated with a single step in a fixed amount of real time that can be calculated beforehand.
- Consequently, **CQSS** is well suited for *real-time simulation of marginally stable systems* with low to modest accuracy requirements.
- Unfortunately, **CQSS** is only first-order accurate. There are reasons to believe that *higher-order CQSS methods* cannot be constructed.

CQSS Features

- **CQSS** preserves the periodical behavior of the harmonic oscillator.
- It has been proven that **CQSS** preserves the *geometrical properties* of general *Hamiltonian (conservative, frictionless) systems*: the method is *symmetric*, *symplectic*, and it preserves *ρ-reversibility*.
- Just like **QSS** and **BQSS**, the method does not call for Newton iterations. In fact, it is the only known solver that exhibits *F-stability* while terminating the computations associated with a single step in a fixed amount of real time that can be calculated beforehand.
- Consequently, **CQSS** is well suited for *real-time simulation of marginally stable systems* with low to modest accuracy requirements.
- Unfortunately, **CQSS** is only first-order accurate. There are reasons to believe that *higher-order CQSS methods* cannot be constructed.
- It is certainly possible to design higher-order QSS-based solvers that are decently well suited for the simulation of marginally stable systems, e.g., by creating a *cyclic method*, in which steps of $\textbf{QSS}_i$ toggle with steps of $\textbf{LIQSS}_i$. Unfortunately, these methods will not preserve the geometric properties of **CQSS** in a strict sense, and they will not be truly F-stable.

Delay Differential Equations

Delay differential equations (DDEs) are similar to ODEs, but their right-hand functions also depend explicitly on past state values.

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), \mathbf{x}(t - \tau_1(\mathbf{x}, t)), \ldots, \mathbf{x}(t - \tau_m(\mathbf{x}, t)), \mathbf{u}(t))$$

Delay Differential Equations

Delay differential equations (DDEs) are similar to ODEs, but their right-hand functions also depend explicitly on past state values.

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), \mathbf{x}(t - \tau_1(\mathbf{x}, t)), \ldots, \mathbf{x}(t - \tau_m(\mathbf{x}, t)), \mathbf{u}(t))$$

The *delay functions* $\tau_j(\cdot)$ can be constant, or they can depend on time and/or the state itself. Instead of an initial state, DDEs need an *initial history* to be solvable.

Delay Differential Equations

Delay differential equations (DDEs) are similar to ODEs, but their right-hand functions also depend explicitly on past state values.

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), \mathbf{x}(t - \tau_1(\mathbf{x}, t)), \ldots, \mathbf{x}(t - \tau_m(\mathbf{x}, t)), \mathbf{u}(t))$$

The *delay functions* $\tau_j(\cdot)$ can be constant, or they can depend on time and/or the state itself. Instead of an initial state, DDEs need an *initial history* to be solvable.

- DDEs require numerical integration algorithms similar to those for ODEs.

Delay Differential Equations

Delay differential equations (DDEs) are similar to ODEs, but their right-hand functions also depend explicitly on past state values.

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), \mathbf{x}(t - \tau_1(\mathbf{x}, t)), \ldots, \mathbf{x}(t - \tau_m(\mathbf{x}, t)), \mathbf{u}(t))$$

The *delay functions* $\tau_j(\cdot)$ can be constant, or they can depend on time and/or the state itself. Instead of an initial state, DDEs need an *initial history* to be solvable.

- DDEs require numerical integration algorithms similar to those for ODEs.
- The algorithms are however more complex, as they must look backward in time in order to compute the state derivatives.

Delay Differential Equations

Delay differential equations (DDEs) are similar to ODEs, but their right-hand functions also depend explicitly on past state values.

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), \mathbf{x}(t - \tau_1(\mathbf{x}, t)), \ldots, \mathbf{x}(t - \tau_m(\mathbf{x}, t)), \mathbf{u}(t))$$

The *delay functions* $\tau_j(\cdot)$ can be constant, or they can depend on time and/or the state itself. Instead of an initial state, DDEs need an *initial history* to be solvable.

- DDEs require numerical integration algorithms similar to those for ODEs.
- The algorithms are however more complex, as they must look backward in time in order to compute the state derivatives.
- From the point of view of the numerical integration, DDEs have some features in common with discontinuous ODEs.

DDEs and QSS Methods

Recently obtained results show that QSS methods provide an efficient and comparatively simple solution to the numerical integration of DDEs.

DDEs and QSS Methods

Recently obtained results show that QSS methods provide an efficient and comparatively simple solution to the numerical integration of DDEs.

Given a DDE:

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), \mathbf{x}(t - \tau_1(\mathbf{x}, t)), \ldots, \mathbf{x}(t - \tau_m(\mathbf{x}, t)), \mathbf{u}(t))$$

DDEs and QSS Methods

Recently obtained results show that QSS methods provide an efficient and comparatively simple solution to the numerical integration of DDEs.

Given a DDE:

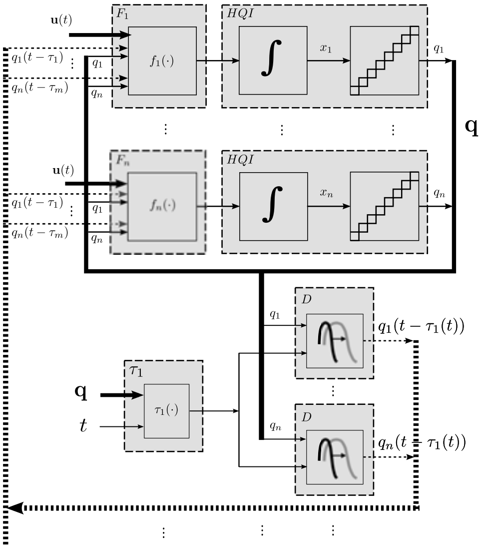
$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), \mathbf{x}(t - \tau_1(\mathbf{x}, t)), \ldots, \mathbf{x}(t - \tau_m(\mathbf{x}, t)), \mathbf{u}(t))$$

The so-called *DQSS methods (DQSS1, DQSS2, or DQSS3)* approximate it as:

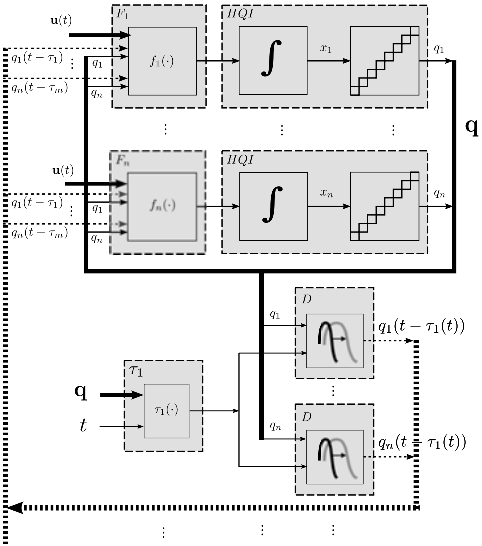
$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{q}(t), \mathbf{q}(t - \tau_1(\mathbf{q}, t)), \ldots, \mathbf{q}(t - \tau_m(\mathbf{q}, t)), \mathbf{u}(t))$$

where the components of $\mathbf{q}$ and $\mathbf{x}$ are related by quantization functions of the given approximation order.

DQSS and PowerDEVS

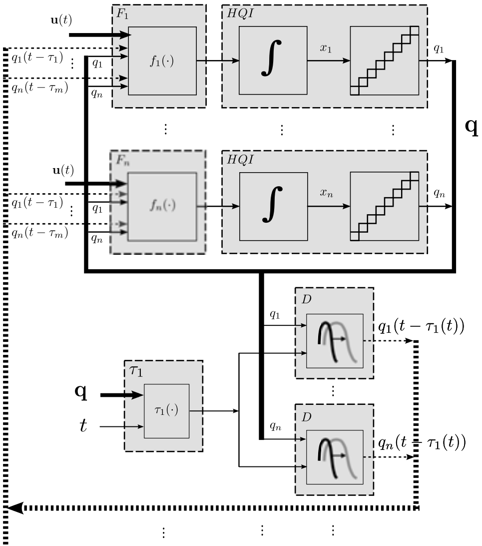


DQSS and PowerDEVS



The *block diagram of a DQSS* contains:

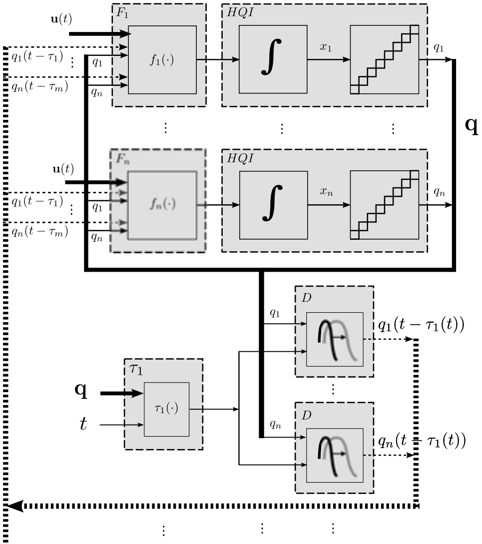
DQSS and PowerDEVS



The *block diagram of a DQSS* contains:

- *hysteretic quantized integrators* to compute $q_i(t)$,

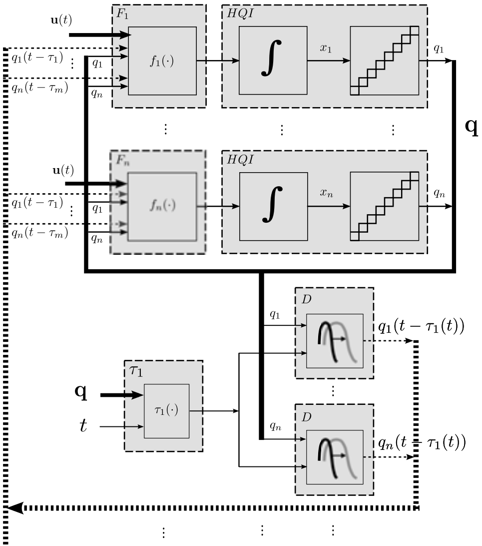
DQSS and PowerDEVS



The *block diagram of a DQSS* contains:

- *hysteretic quantized integrators* to compute $q_i(t)$,
- *static functions* to compute $\dot{x}_i(t)$,

DQSS and PowerDEVS

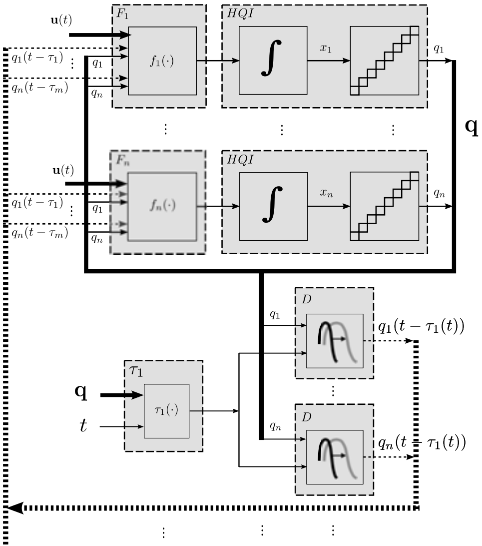


The *block diagram of a DQSS* contains:

- *hysteretic quantized integrators* to compute $q_i(t)$,
- *static functions* to compute $\dot{x}_i(t)$,

and also:

DQSS and PowerDEVS



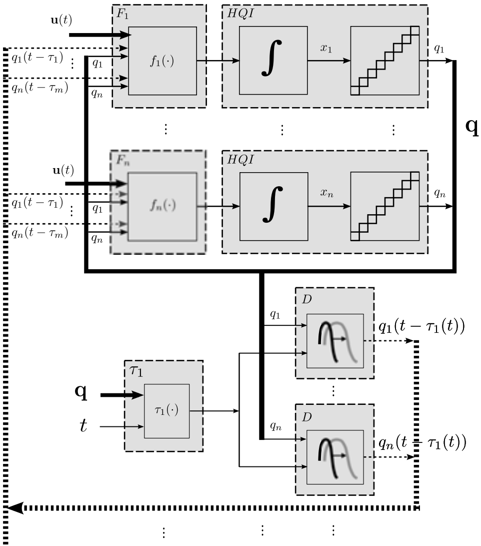
The *block diagram of a DQSS* contains:

- *hysteretic quantized integrators* to compute $q_i(t)$,
- *static functions* to compute $\dot{x}_i(t)$,

and also:

- *static functions* to compute $\tau_j(t)$, and

DQSS and PowerDEVS



The *block diagram of a DQSS* contains:

- *hysteretic quantized integrators* to compute $q_i(t)$,
- *static functions* to compute $\dot{x}_i(t)$,

and also:

- *static functions* to compute $\tau_j(t)$, and
- a new type of *delay blocks* that compute $q_i(t - \tau_j(t))$ given $q_i(t)$ and $\tau_j(t)$.

DQSS and PowerDEVS II

Each of the **PowerDEVS** blocks can be implemented as a *DEVS model*.

DQSS and PowerDEVS II

Each of the **PowerDEVS** blocks can be implemented as a *DEVS model*.

- We already have seen the DEVS models for the *hysteretic quantized integrators*.

DQSS and PowerDEVS II

Each of the **PowerDEVS** blocks can be implemented as a *DEVS model*.

- We already have seen the DEVS models for the *hysteretic quantized integrators*.
- We also have seen the DEVS models for *static functions*.

DQSS and PowerDEVS II

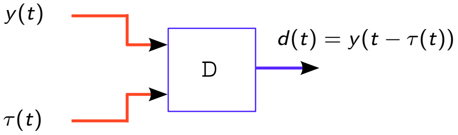
Each of the **PowerDEVS** blocks can be implemented as a *DEVS model*.

- We already have seen the DEVS models for the *hysteretic quantized integrators*.
- We also have seen the DEVS models for *static functions*.
- We only need to still build a DEVS model for the *delay block*.

DQSS and PowerDEVS II

Each of the **PowerDEVS** blocks can be implemented as a *DEVS model*.

- We already have seen the DEVS models for the *hysteretic quantized integrators*.
- We also have seen the DEVS models for *static functions*.
- We only need to still build a DEVS model for the *delay block*.



The Delay Block

The inputs of the *delay block* $y(t)$ and $\tau(t)$ and the output $d(t) = y(t - \tau(t))$ are *piecewise polynomial trajectories*. The maximum order of the polynomials depends on the approximation order of the method.

The Delay Block

The inputs of the *delay block* $y(t)$ and $\tau(t)$ and the output $d(t) = y(t - \tau(t))$ are *piecewise polynomial trajectories*. The maximum order of the polynomials depends on the approximation order of the method.

- The first input $y(t)$ will be represented as a sequence of events at times t_k^y, carrying values $y_{0,k}$, $y_{1,k}$, $y_{2,k}$, so that:

$$y(t) = y_{0,k} + y_{1,k} \cdot (t - t_k^y) + y_{2,k} \cdot (t - t_k^y)^2; \quad \text{for } t_k^y < t < t_{k+1}^y$$

The Delay Block

The inputs of the *delay block* $y(t)$ and $\tau(t)$ and the output $d(t) = y(t - \tau(t))$ are *piecewise polynomial trajectories*. The maximum order of the polynomials depends on the approximation order of the method.

- The first input $y(t)$ will be represented as a sequence of events at times t_k^y, carrying values $y_{0,k}$, $y_{1,k}$, $y_{2,k}$, so that:

$$y(t) = y_{0,k} + y_{1,k} \cdot (t - t_k^y) + y_{2,k} \cdot (t - t_k^y)^2; \quad \text{for } t_k^y < t < t_{k+1}^y$$

- The second input $\tau(t)$ will be represented as a sequence of events at times t_j^τ, carrying values $\tau_{0,j}$, $\tau_{1,j}$, $\tau_{2,j}$, so that:

$$\tau(t) = \tau_{0,j} + \tau_{1,j} \cdot (t - t_j^\tau) + \tau_{2,j} \cdot (t - t_j^\tau)^2; \quad \text{for } t_j^\tau < t < t_{j+1}^\tau$$

The Delay Block

The inputs of the *delay block* $y(t)$ and $\tau(t)$ and the output $d(t) = y(t - \tau(t))$ are *piecewise polynomial trajectories*. The maximum order of the polynomials depends on the approximation order of the method.

- The first input $y(t)$ will be represented as a sequence of events at times t_k^y, carrying values $y_{0,k}$, $y_{1,k}$, $y_{2,k}$, so that:

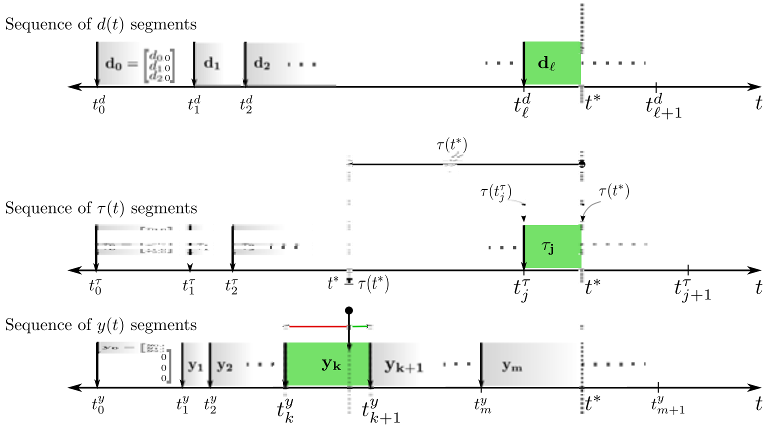
$$y(t) = y_{0,k} + y_{1,k} \cdot (t - t_k^y) + y_{2,k} \cdot (t - t_k^y)^2; \quad \text{for } t_k^y < t < t_{k+1}^y$$

- The second input $\tau(t)$ will be represented as a sequence of events at times t_j^τ, carrying values $\tau_{0,j}$, $\tau_{1,j}$, $\tau_{2,j}$, so that:

$$\tau(t) = \tau_{0,j} + \tau_{1,j} \cdot (t - t_j^\tau) + \tau_{2,j} \cdot (t - t_j^\tau)^2; \quad \text{for } t_j^\tau < t < t_{j+1}^\tau$$

- The *DEVS delay model* computes the corresponding event sequences of $d(t) = y(t - \tau(t))$.

The Delay Algorithm



Theoretical Properties of DQSS

Given a *DDE with constant delays* of the form:

$$\dot{\mathbf{x}}(t) = \mathbf{A} \cdot \mathbf{x}(t) + \sum_{i=1}^{m} \mathbf{A_i} \cdot \mathbf{x}(t - \tau_i)$$

Theoretical Properties of DQSS

Given a *DDE with constant delays* of the form:

$$\dot{\mathbf{x}}(t) = \mathbf{A} \cdot \mathbf{x}(t) + \sum_{i=1}^{m} \mathbf{A_i} \cdot \mathbf{x}(t - \tau_i)$$

where the analytical solution $\mathbf{x}(t) = 0$ corresponding to the trivial initial history is asymptotically stable, then:

Theoretical Properties of DQSS

Given a *DDE with constant delays* of the form:

$$\dot{\mathbf{x}}(t) = \mathbf{A} \cdot \mathbf{x}(t) + \sum_{i=1}^{m} \mathbf{A_i} \cdot \mathbf{x}(t - \tau_i)$$

where the analytical solution $\mathbf{x}(t) = 0$ corresponding to the trivial initial history is asymptotically stable, then:

- The absolute global error committed by any DQSS method is bounded for any quanta adopted.

Theoretical Properties of DQSS

Given a *DDE with constant delays* of the form:

$$\dot{\mathbf{x}}(t) = \mathbf{A} \cdot \mathbf{x}(t) + \sum_{i=1}^{m} \mathbf{A_i} \cdot \mathbf{x}(t - \tau_i)$$

where the analytical solution $\mathbf{x}(t) = 0$ corresponding to the trivial initial history is asymptotically stable, then:

- The absolute global error committed by any DQSS method is bounded for any quanta adopted.
- The error goes to zero when the quantization goes to zero.

Theoretical Properties of DQSS

Given a *DDE with constant delays* of the form:

$$\dot{\mathbf{x}}(t) = \mathbf{A} \cdot \mathbf{x}(t) + \sum_{i=1}^{m} \mathbf{A_i} \cdot \mathbf{x}(t - \tau_i)$$

where the analytical solution $\mathbf{x}(t) = 0$ corresponding to the trivial initial history is asymptotically stable, then:

- The absolute global error committed by any DQSS method is bounded for any quanta adopted.
- The error goes to zero when the quantization goes to zero.

There is a similar property for *general nonlinear DDE models with state and time dependent delays*. In that case, a stronger condition (*input-to-state stability*) is required for the original system, and the global error boundedness is only guaranteed for sufficiently small quanta.

DQSS Features

- The usage of *QSS methods for DDE simulations* provides *highly efficient algorithms* that often improve on the results obtained by classic time-slicing methods.

DQSS Features

- The usage of *QSS methods for DDE simulations* provides *highly efficient algorithms* that often improve on the results obtained by classic time-slicing methods.

- While conventional methods for ODEs must be almost entirely reformulated to deal with DDEs, QSS methods reduce the *problem of the delays* to *polynomial manipulations locally managed by a new atomic DEVS block*. The static functions and hysteretic quantized integrators remain unchanged.

DQSS Features

- The usage of *QSS methods for DDE simulations* provides *highly efficient algorithms* that often improve on the results obtained by classic time-slicing methods.

- While conventional methods for ODEs must be almost entirely reformulated to deal with DDEs, QSS methods reduce the *problem of the delays* to *polynomial manipulations locally managed by a new atomic DEVS block*. The static functions and hysteretic quantized integrators remain unchanged.

- DQSS solvers have *strong theoretical properties* regarding *stability*, *convergence*, and *accuracy* that *guarantee the correctness of the results*.

QSS Methods in Real Time

QSS methods have features that make them particularly suitable for *running simulations in real time*:

QSS Methods in Real Time

QSS methods have features that make them particularly suitable for *running simulations in real time*:

- They *detect and handle discontinuities* without iterations.

QSS Methods in Real Time

QSS methods have features that make them particularly suitable for *running simulations in real time*:

- They *detect and handle discontinuities* without iterations.
- They can integrate many *stiff systems* in predetermined time, i.e., *without requiring Newton iterations*.

QSS Methods in Real Time

QSS methods have features that make them particularly suitable for *running simulations in real time*:

- They *detect and handle discontinuities* without iterations.
- They can integrate many *stiff systems* in predetermined time, i.e., *without requiring Newton iterations*.
- If real-time speed cannot be attained, *accuracy can be sacrificed* by increasing the quantum. This would *only affect accuracy*, whereas *stability will be preserved*.

QSS Methods in Real Time

QSS methods have features that make them particularly suitable for *running simulations in real time*:

- They *detect and handle discontinuities* without iterations.
- They can integrate many *stiff systems* in predetermined time, i.e., *without requiring Newton iterations*.
- If real-time speed cannot be attained, *accuracy can be sacrificed* by increasing the quantum. This would *only affect accuracy*, whereas *stability will be preserved*.
- They offer *dense output*. Thus, the simulation outputs can be communicated at any instant of time.

QSS Methods in Real Time

QSS methods have features that make them particularly suitable for *running simulations in real time*:

- They *detect and handle discontinuities* without iterations.
- They can integrate many *stiff systems* in predetermined time, i.e., *without requiring Newton iterations*.
- If real-time speed cannot be attained, *accuracy can be sacrificed* by increasing the quantum. This would *only affect accuracy*, whereas *stability will be preserved*.
- They offer *dense output*. Thus, the simulation outputs can be communicated at any instant of time.
- These algorithms are *naturally asynchronous*. Hence distributing QSS simulations across a *parallel multi-processor architecture* is trivial.

PowerDEVS in Real Time

A version of **PowerDEVS** runs on a *real-time operative system* called **Linux RTAI**, with the following features:

PowerDEVS in Real Time

A version of **PowerDEVS** runs on a *real-time operative system* called **Linux RTAI**, with the following features:

- Simulations can be *synchronized with the real-time clock* with a precision of roughly $1\mu s$.

PowerDEVS in Real Time

A version of **PowerDEVS** runs on a *real-time operative system* called **Linux RTAI**, with the following features:

- Simulations can be *synchronized with the real-time clock* with a precision of roughly $1\mu s$.
- **PowerDEVS-RT** offers special modules to *detect and process hardware interrupts*.

PowerDEVS in Real Time

A version of **PowerDEVS** runs on a *real-time operative system* called **Linux RTAI**, with the following features:

- Simulations can be *synchronized with the real-time clock* with a precision of roughly $1\mu s$.
- **PowerDEVS-RT** offers special modules to *detect and process hardware interrupts*.
- **PowerDEVS-RT** permits to *communicate with input and output hardware ports* in a very simple fashion.

PowerDEVS in Real Time

A version of **PowerDEVS** runs on a *real-time operative system* called **Linux RTAI**, with the following features:

- Simulations can be *synchronized with the real-time clock* with a precision of roughly $1\mu s$.
- **PowerDEVS-RT** offers special modules to *detect and process hardware interrupts*.
- **PowerDEVS-RT** permits to *communicate with input and output hardware ports* in a very simple fashion.
- **PowerDEVS-RT** has been successfully used in the *real-time simulation of power electronic converters*, working at frequencies where most real-time simulation tools fail.

Conclusions

- A *broad palette of different QSS-based algorithms* has been presented in the last three presentations.

Conclusions

- A *broad palette of different QSS-based algorithms* has been presented in the last three presentations.

- QSS-based algorithms have meanwhile proven themselves to offer an *interesting and innovative new approach to numerical ODE, DAE, and DDE solutions* for many different problems.

Conclusions

- A *broad palette of different QSS-based algorithms* has been presented in the last three presentations.
- QSS-based algorithms have meanwhile proven themselves to offer an *interesting and innovative new approach to numerical ODE, DAE, and DDE solutions* for many different problems.
- QSS-based algorithms can be *highly competitive* when used for applications with *low to average accuracy requirements*. Problems calling for high-order algorithms cannot be dealt with effectively using QSS-based solvers.

Conclusions

- A *broad palette of different QSS-based algorithms* has been presented in the last three presentations.
- QSS-based algorithms have meanwhile proven themselves to offer an *interesting and innovative new approach to numerical ODE, DAE, and DDE solutions* for many different problems.
- QSS-based algorithms can be *highly competitive* when used for applications with *low to average accuracy requirements*. Problems calling for high-order algorithms cannot be dealt with effectively using QSS-based solvers.
- Already QSS4 is rarely more cost-effective than QSS3, because the increased overhead eats into the benefit to be reaped from the larger step sizes. Already QSS5 will require iterations in every step, because a fifth-order polynomial will have to be solved, which can only be done iteratively.

Conclusions II

- QSS-based algorithms are of particular interest for the *simulation of systems exhibiting frequent discontinuities*, as state events can be handled much more efficiently by state-quantization algorithms than by time-slicing algorithms.

Conclusions II

- QSS-based algorithms are of particular interest for the *simulation of systems exhibiting frequent discontinuities*, as state events can be handled much more efficiently by state-quantization algorithms than by time-slicing algorithms.

- The benefit of QSS-based solvers becomes even more pronounced when the *discontinuous system to be simulated is furthermore stiff*. A typical application of stiff discontinuous systems are *models of switching power electronic circuits*.

Conclusions II

- QSS-based algorithms are of particular interest for the *simulation of systems exhibiting frequent discontinuities*, as state events can be handled much more efficiently by state-quantization algorithms than by time-slicing algorithms.

- The benefit of QSS-based solvers becomes even more pronounced when the *discontinuous system to be simulated is furthermore stiff*. A typical application of stiff discontinuous systems are *models of switching power electronic circuits*.

- QSS-based solvers are very promising for the *simulation of large-scale models*, as they exploit the sparsity inherent in these models naturally and directly.

Conclusions II

- QSS-based algorithms are of particular interest for the *simulation of systems exhibiting frequent discontinuities*, as state events can be handled much more efficiently by state-quantization algorithms than by time-slicing algorithms.
- The benefit of QSS-based solvers becomes even more pronounced when the *discontinuous system to be simulated is furthermore stiff*. A typical application of stiff discontinuous systems are *models of switching power electronic circuits*.
- QSS-based solvers are very promising for the *simulation of large-scale models*, as they exploit the sparsity inherent in these models naturally and directly.
- QSS-based solvers perform equally well as classical solvers when dealing with *small-scale models of systems without discontinuities*. Yet in those cases, there is nothing that the QSS solvers can exploit that would make them more attractive than the much simpler classical ODE solvers.

Conclusions III

- QSS-based algorithms are of particular interest for *real-time simulation*.

Conclusions III

- QSS-based algorithms are of particular interest for *real-time simulation*.
- Since *QSS solvers are naturally asynchronous*, they can be easily *distributed over a multi-processor architecture*.

Conclusions III

- QSS-based algorithms are of particular interest for *real-time simulation*.
- Since *QSS solvers are naturally asynchronous*, they can be easily *distributed over a multi-processor architecture*.
- The *communication band-width is minimized*, because communication between two processors only takes place at event times. As long as not much is happening, there is no need to communicate at all. Thus, also the communication occurs in a naturally asynchronous fashion.

Conclusions III

- QSS-based algorithms are of particular interest for *real-time simulation*.
- Since *QSS solvers are naturally asynchronous*, they can be easily *distributed over a multi-processor architecture*.
- The *communication band-width is minimized*, because communication between two processors only takes place at event times. As long as not much is happening, there is no need to communicate at all. Thus, also the communication occurs in a naturally asynchronous fashion.
- The communication band-width is furthermore minimized, because *only a single bit needs to be communicated* between processors to indicate a state change. A message of "1" means that the sending state incremented its level, whereas a message of "0" means that it decremented its level. As long as the state variable doesn't change its level, it doesn't send an output event through its output channel.

References

1. Bergero, F., E. Kofman, and F.E. Cellier (2013), "A Novel Parallelization Technique for DEVS Simulation of Continuous and Hybrid Systems," *Simulation*, in press.

2. Migoni, G., M. Bortolotto, E. Kofman, and F.E. Cellier (2013), "Linearly Implicit Quantization–based Integration Methods for Stiff Ordinary Differential Equations," *Simulation Modelling Practice and Theory*, **35**(6), pp.118-136.

3. Migoni, G., E. Kofman, and F.E. Cellier (2012), "Quantization-based New Integration Methods for Stiff ODEs," *Simulation*, **88**(4), pp.387-407.

4. Castro, R., E. Kofman, and F.E. Cellier (2011), "Quantization-based Integration Methods for Delay Differential Equations," *Simulation Modelling Practice and Theory*, **19**(1), pp.314-336.

5. Kofman, E., F.E. Cellier, and G. Migoni (2010), "Continuous System Simulation and Control," in: *Discrete Event Simulation and Modeling: Theory and Applications (Model-Based Design)*, (G.A. Wainer and P.J. Mosterman, eds.), CRC Press, Boca Raton, FL, pp.75-107.

References II

1. Bergero, Federico (2012), *Simulaciones de Sistemas Híbridos por Eventos Discretos: Tiempo Real y Paralelismo*, Ph.D. Dissertation, Faculty of Exact Sciences, Engineering, and Land-surveying, National University of Rosario, Argentina.

2. Migoni, Gustavo (2010), *Simulación por Cuantificación de Sistemas Stiff*, Ph.D. Dissertation, Faculty of Exact Sciences, Engineering, and Land-surveying, National University of Rosario, Argentina.