Numerical Simulation of Dynamic Systems XXVI

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

May 21, 2013

Introduction

We have shown in the course of this class that, in order to be able to trust the results of a simulation, it is crucial that we understand the properties of *numerical stability* and *accuracy* of the underlying solver.

Introduction

We have shown in the course of this class that, in order to be able to trust the results of a simulation, it is crucial that we understand the properties of *numerical stability* and *accuracy* of the underlying solver.

To this end, we usually analyzed the relationship between the *eigenvalues* of the **A**-matrix of the original continuous system and those of the **F**-matrix of the resulting numerical approximation.

Introduction

We have shown in the course of this class that, in order to be able to trust the results of a simulation, it is crucial that we understand the properties of *numerical stability* and *accuracy* of the underlying solver.

To this end, we usually analyzed the relationship between the *eigenvalues* of the **A**-matrix of the original continuous system and those of the **F**-matrix of the resulting numerical approximation.

Unfortunately, we cannot apply this same technique to QSS methods, because even if the original system happened to be linear, the resulting *discrete system* is not, and consequently, there is no **F**-matrix that could be analyzed.

Introduction

We have shown in the course of this class that, in order to be able to trust the results of a simulation, it is crucial that we understand the properties of *numerical stability* and *accuracy* of the underlying solver.

To this end, we usually analyzed the relationship between the *eigenvalues* of the **A**-matrix of the original continuous system and those of the **F**-matrix of the resulting numerical approximation.

Unfortunately, we cannot apply this same technique to QSS methods, because even if the original system happened to be linear, the resulting *discrete system* is not, and consequently, there is no **F**-matrix that could be analyzed.

However, we noticed that we can interpret the simulation results generated by the QSS solver as a *perturbation* around the analytical solution.

Introduction

We have shown in the course of this class that, in order to be able to trust the results of a simulation, it is crucial that we understand the properties of *numerical stability* and *accuracy* of the underlying solver.

To this end, we usually analyzed the relationship between the *eigenvalues* of the **A**-matrix of the original continuous system and those of the **F**-matrix of the resulting numerical approximation.

Unfortunately, we cannot apply this same technique to QSS methods, because even if the original system happened to be linear, the resulting *discrete system* is not, and consequently, there is no **F**-matrix that could be analyzed.

However, we noticed that we can interpret the simulation results generated by the QSS solver as a *perturbation* around the analytical solution.

We shall make use of this idea to approach the analysis of numerical stability and accuracy of these solvers.

Representation of the QSS Solution as a Perturbation

Given the *non-linear continuous system*:

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

Representation of the QSS Solution as a Perturbation

Given the *non-linear continuous system*:

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

Its *QSS approximation* can be written as:

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

where $\mathbf{x}(t)$ and $\mathbf{q}(t)$ are related componentwise by *hysteretic quantization functions*.

Representation of the QSS Solution as a Perturbation

Given the *non-linear continuous system*:

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

Its *QSS approximation* can be written as:

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

where $\mathbf{x}(t)$ and $\mathbf{q}(t)$ are related componentwise by *hysteretic quantization functions*.

Defining $\mathbf{\Delta x}(t) = \mathbf{q}(t) - \mathbf{x}(t)$, The QSS approximation can be rewritten as:

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

Representation of the QSS Solution as a Perturbation

Given the *non-linear continuous system*:

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

Its *QSS approximation* can be written as:

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

where $\mathbf{x}(t)$ and $\mathbf{q}(t)$ are related componentwise by *hysteretic quantization functions*.

Defining $\mathbf{\Delta x}(t) = \mathbf{q}(t) - \mathbf{x}(t)$, The QSS approximation can be rewritten as:

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

The simulation model is a *perturbed version* of the original continuous system.

Representation of the QSS Solution as a Perturbation

Given the *non-linear continuous system*:

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

Its *QSS approximation* can be written as:

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

where $\mathbf{x}(t)$ and $\mathbf{q}(t)$ are related componentwise by *hysteretic quantization functions*.

Defining $\mathbf{\Delta x}(t) = \mathbf{q}(t) - \mathbf{x}(t)$, The QSS approximation can be rewritten as:

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

The simulation model is a *perturbed version* of the original continuous system.

We also note that the perturbation is bounded:

$$|\Delta x_i(t)| = |q_i(t) - x_i(t)| \leq \max(\Delta Q_i, \varepsilon_i)$$

with ΔQ_i being the quantum of the i^{th} state variable, and ε_i being its associated hysteresis width.

Convergence, Accuracy, and Stability in QSS

We can study the *numerical stability* and the *approximation accuracy* of QSS methods by analyzing the *effects of the bounded perturbations*.

Convergence, Accuracy, and Stability in QSS

We can study the *numerical stability* and the *approximation accuracy* of QSS methods by analyzing the *effects of the bounded perturbations*.

- We shall demonstrate the *convergence* of the QSS approximation for non-linear systems. This means that, as the *quantum* and the *hysteresis width* are being reduced to *zero*, the trajectories of the QSS solution become in the limit identical to those of the original continuous system.

Convergence, Accuracy, and Stability in QSS

We can study the *numerical stability* and the *approximation accuracy* of QSS methods by analyzing the *effects of the bounded perturbations*.

- We shall demonstrate the *convergence* of the QSS approximation for non-linear systems. This means that, as the *quantum* and the *hysteresis width* are being reduced to *zero*, the trajectories of the QSS solution become in the limit identical to those of the original continuous system.

- Contrary to the classical solvers, where we usually limited the *stability analysis* to linear systems only, we shall now discuss the numerical stability of the QSS approximation for arbitrary *non-linear systems*. It shall be shown that, wherever the original system exhibits an *asymptotically stable equilibrium point*, there is a region around that equilibrium point such that, for any initial condition within that region, we can find a quantum of finite size that guarantees that the trajectories of the QSS approximation *end up bounded* in a region around that equilibrium point.

Convergence, Accuracy, and Stability in QSS II

Although these results are interesting from a theoretical point of view, as they allow us to offer assurances that the *QSS approximation of an asymptotically stable continuous system remains always numerically stable*, we don't know yet, how good the approximation actually is, as we cannot offer in general a quantitative value for the upper bound of the size of the region around the equilibrium point, in which the QSS simulation terminates.

Convergence, Accuracy, and Stability in QSS II

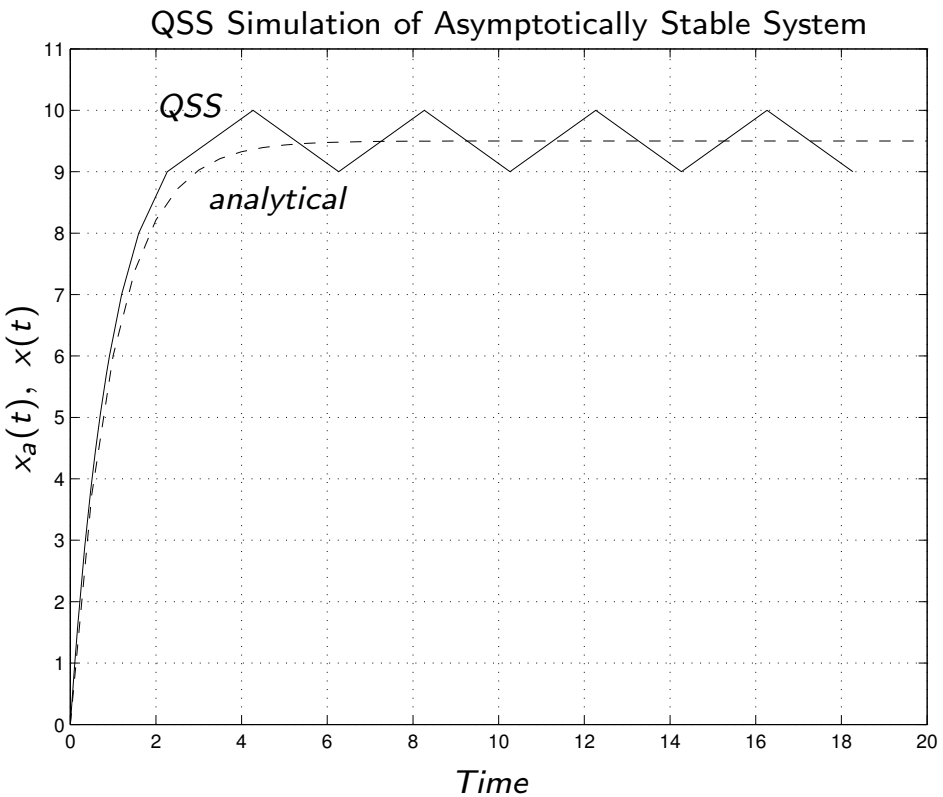
Although these results are interesting from a theoretical point of view, as they allow us to offer assurances that the *QSS approximation of an asymptotically stable continuous system remains always numerically stable*, we don't know yet, how good the approximation actually is, as we cannot offer in general a quantitative value for the upper bound of the size of the region around the equilibrium point, in which the QSS simulation terminates.

Unfortunately, even if the continuous system exhibits a *constant steady-state*, the QSS approximation will usually *oscillate around that steady state* as shown to the right for the scalar system $\dot{x}_a(t) = -x_a(t) + 9.5$ with $\Delta Q = 1$ and $\varepsilon = 1$:

Convergence, Accuracy, and Stability in QSS II

Although these results are interesting from a theoretical point of view, as they allow us to offer assurances that the *QSS approximation of an asymptotically stable continuous system remains always numerically stable*, we don't know yet, how good the approximation actually is, as we cannot offer in general a quantitative value for the upper bound of the size of the region around the equilibrium point, in which the QSS simulation terminates.

Unfortunately, even if the continuous system exhibits a *constant steady-state*, the QSS approximation will usually *oscillate around that steady state* as shown to the right for the scalar system $\dot{x}_a(t) = -x_a(t) + 9.5$ with $\Delta Q = 1$ and $\varepsilon = 1$:



Convergence, Accuracy, and Stability in QSS III

In order to judge the *accuracy* of the QSS simulation, we need something a bit more concrete. We'll need a *quantitative upper bound of the oscillation*. Unfortunately, such a bound can only be given for *linear time-invariant systems*.

Convergence, Accuracy, and Stability in QSS III

In order to judge the *accuracy* of the QSS simulation, we need something a bit more concrete. We'll need a *quantitative upper bound of the oscillation*. Unfortunately, such a bound can only be given for *linear time-invariant systems*.

However for these systems, we can offer a quantitative upper bound of the *global numerical integration error*, a result that we were unable to get for classical solvers. Until now, we were able to control the *local numerical integration error* within a single integration step only.

Convergence, Accuracy, and Stability in QSS III

In order to judge the *accuracy* of the QSS simulation, we need something a bit more concrete. We'll need a *quantitative upper bound of the oscillation*. Unfortunately, such a bound can only be given for *linear time-invariant systems*.

However for these systems, we can offer a quantitative upper bound of the *global numerical integration error*, a result that we were unable to get for classical solvers. Until now, we were able to control the *local numerical integration error* within a single integration step only.

This is a very important result that has significant practical implications.

QSS Simulations of Linear Time-invariant Systems

Let us consider a *linear time-invariant system* and its QSS approximation:

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

QSS Simulations of Linear Time-invariant Systems

Let us consider a *linear time-invariant system* and its QSS approximation:

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

If the $\mathbf{A}$-matrix is Hurwitz, i.e., the system is analytically stable, and if $\mathbf{x}(t_0) = \mathbf{x}_\mathbf{a}(t_0)$, it can be shown that:

$$|\mathbf{x}(t) - \mathbf{x}_\mathbf{a}(t)| \leq |\mathbf{V}| \cdot |\mathbb{R}\mathrm{e}\{\mathbf{\Lambda}\}^{-1} \cdot \mathbf{\Lambda}| \cdot |\mathbf{V}^{-1}| \cdot \mathbf{\Delta Q} \quad \forall t \geq t_0$$

where:

QSS Simulations of Linear Time-invariant Systems

Let us consider a *linear time-invariant system* and its QSS approximation:

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

If the $\mathbf{A}$-matrix is Hurwitz, i.e., the system is analytically stable, and if $\mathbf{x}(t_0) = \mathbf{x}_\mathbf{a}(t_0)$, it can be shown that:

$$|\mathbf{x}(t) - \mathbf{x}_\mathbf{a}(t)| \leq |\mathbf{V}| \cdot |\mathbb{R}\mathrm{e}\{\mathbf{\Lambda}\}^{-1} \cdot \mathbf{\Lambda}| \cdot |\mathbf{V}^{-1}| \cdot \mathbf{\Delta Q} \quad \forall t \geq t_0$$

where:

- The operators "$\leq$", "$|\cdot|$", and "$\mathbb{R}\mathrm{e}\{\cdot\}$" are applied *componentwise*.

QSS Simulations of Linear Time-invariant Systems

Let us consider a *linear time-invariant system* and its QSS approximation:

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

If the $\mathbf{A}$-matrix is Hurwitz, i.e., the system is analytically stable, and if $\mathbf{x}(t_0) = \mathbf{x}_\mathbf{a}(t_0)$, it can be shown that:

$$|\mathbf{x}(t) - \mathbf{x}_\mathbf{a}(t)| \leq |\mathbf{V}| \cdot |\mathbb{R}\mathrm{e}\{\mathbf{\Lambda}\}^{-1} \cdot \mathbf{\Lambda}| \cdot |\mathbf{V}^{-1}| \cdot \mathbf{\Delta Q} \quad \forall t \geq t_0$$

where:

- The operators "$\leq$", "$|\cdot|$", and "$\mathbb{R}\mathrm{e}\{\cdot\}$" are applied *componentwise*.
- $\mathbf{A} = \mathbf{V} \cdot \mathbf{\Lambda} \cdot \mathbf{V}^{-1}$ is the spectral decomposition of $\mathbf{A}$.

QSS Simulations of Linear Time-invariant Systems

Let us consider a *linear time-invariant system* and its QSS approximation:

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

If the $\mathbf{A}$-matrix is Hurwitz, i.e., the system is analytically stable, and if $\mathbf{x}(t_0) = \mathbf{x}_\mathbf{a}(t_0)$, it can be shown that:

$$|\mathbf{x}(t) - \mathbf{x}_\mathbf{a}(t)| \leq |\mathbf{V}| \cdot |\mathbb{R}\mathrm{e}\{\mathbf{\Lambda}\}^{-1} \cdot \mathbf{\Lambda}| \cdot |\mathbf{V}^{-1}| \cdot \mathbf{\Delta Q} \quad \forall t \geq t_0$$

where:

- The operators "$\leq$", "$|\cdot|$", and "$\mathbb{R}\mathrm{e}\{\cdot\}$" are applied *componentwise*.
- $\mathbf{A} = \mathbf{V} \cdot \mathbf{\Lambda} \cdot \mathbf{V}^{-1}$ is the spectral decomposition of $\mathbf{A}$.
- $\mathbf{\Delta Q} = \begin{bmatrix} \max(\Delta Q_1, \varepsilon_1) \\ \vdots \\ \max(\Delta Q_n, \varepsilon_n) \end{bmatrix}$.

QSS Simulations of Linear Time-invariant Systems

Let us consider a *linear time-invariant system* and its QSS approximation:

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

If the $\mathbf{A}$-matrix is Hurwitz, i.e., the system is analytically stable, and if $\mathbf{x}(t_0) = \mathbf{x}_\mathbf{a}(t_0)$, it can be shown that:

$$|\mathbf{x}(t) - \mathbf{x}_\mathbf{a}(t)| \leq |\mathbf{V}| \cdot |\mathbb{R}\mathrm{e}\{\mathbf{\Lambda}\}^{-1} \cdot \mathbf{\Lambda}| \cdot |\mathbf{V}^{-1}| \cdot \mathbf{\Delta Q} \quad \forall t \geq t_0$$

where:

- The operators "$\leq$", "$|\cdot|$", and "$\mathbb{R}\mathrm{e}\{\cdot\}$" are applied *componentwise*.
- $\mathbf{A} = \mathbf{V} \cdot \mathbf{\Lambda} \cdot \mathbf{V}^{-1}$ is the spectral decomposition of $\mathbf{A}$.
- $\mathbf{\Delta Q} = \begin{bmatrix} \max(\Delta Q_1, \varepsilon_1) \\ \vdots \\ \max(\Delta Q_n, \varepsilon_n) \end{bmatrix}$.

It should be noted that:

- The QSS method offers a *computable global error bound*.

QSS Simulations of Linear Time-invariant Systems

Let us consider a *linear time-invariant system* and its QSS approximation:

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

If the $\mathbf{A}$-matrix is Hurwitz, i.e., the system is analytically stable, and if $\mathbf{x}(t_0) = \mathbf{x}_\mathbf{a}(t_0)$, it can be shown that:

$$|\mathbf{x}(t) - \mathbf{x}_\mathbf{a}(t)| \leq |\mathbf{V}| \cdot |\mathbb{R}\mathrm{e}\{\mathbf{\Lambda}\}^{-1} \cdot \mathbf{\Lambda}| \cdot |\mathbf{V}^{-1}| \cdot \mathbf{\Delta Q} \quad \forall t \geq t_0$$

where:

- The operators "$\leq$", "$|\cdot|$", and "$\mathbb{R}\mathrm{e}\{\cdot\}$" are applied *componentwise*.
- $\mathbf{A} = \mathbf{V} \cdot \mathbf{\Lambda} \cdot \mathbf{V}^{-1}$ is the spectral decomposition of $\mathbf{A}$.
- $\mathbf{\Delta Q} = \begin{bmatrix} \max(\Delta Q_1, \varepsilon_1) \\ \vdots \\ \max(\Delta Q_n, \varepsilon_n) \end{bmatrix}$.

It should be noted that:

- The QSS method offers a *computable global error bound*.
- The numerical solution remains *practically stable* for any value of the *quantum*.

Choosing Quantum and Hysteresis Width

The simulation results generated by the QSS solver usually exhibit steady-state *oscillations*. Their *frequency* depends on the *hysteresis*.

Choosing Quantum and Hysteresis Width

The simulation results generated by the QSS solver usually exhibit steady-state *oscillations*. Their *frequency* depends on the *hysteresis*. For example, the output trajectories of the scalar system:

$$\dot{x}_a(t) = -x_a(t) + 9.5 \quad ; \quad x_a(0) = 0$$

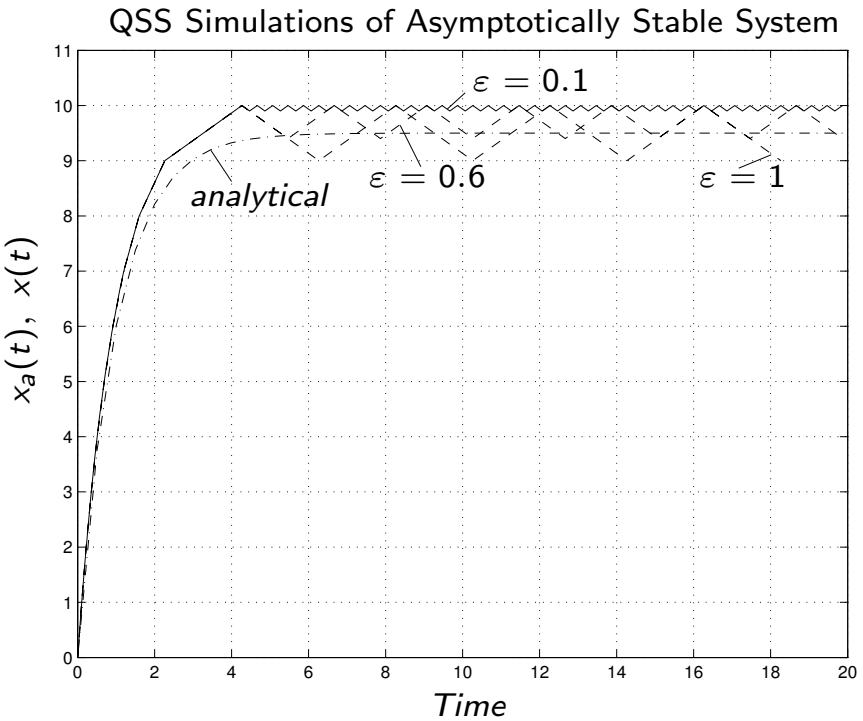
with $\Delta Q = 1$ and different values of ε are:

Choosing Quantum and Hysteresis Width

The simulation results generated by the QSS solver usually exhibit steady-state *oscillations*. Their *frequency* depends on the *hysteresis*. For example, the output trajectories of the scalar system:

$$\dot{x}_a(t) = -x_a(t) + 9.5 \quad ; \quad x_a(0) = 0$$

with $\Delta Q = 1$ and different values of ε are:

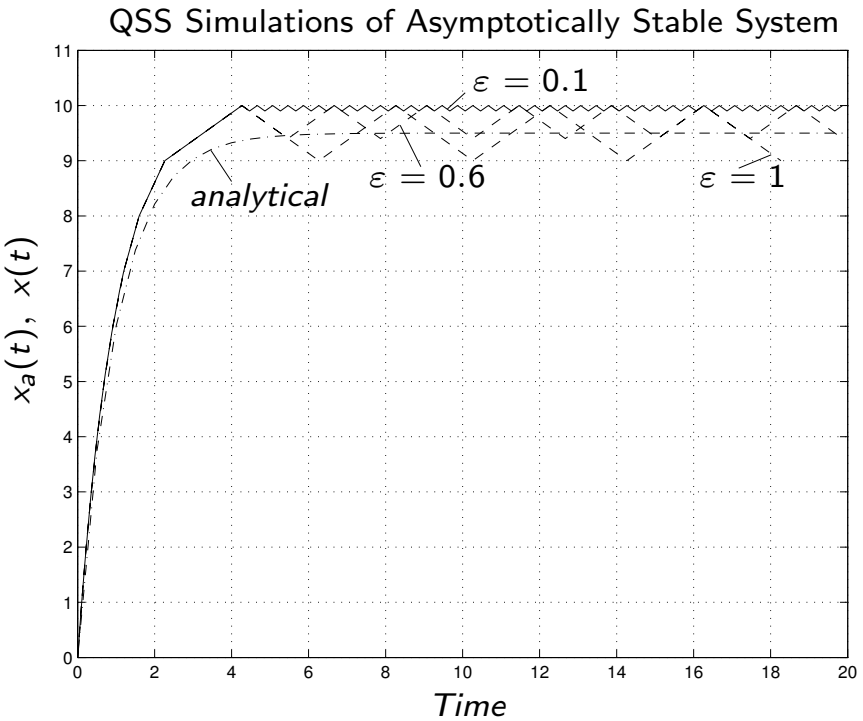


Choosing Quantum and Hysteresis Width

The simulation results generated by the QSS solver usually exhibit steady-state *oscillations*. Their *frequency* depends on the *hysteresis*. For example, the output trajectories of the scalar system:

$$\dot{x}_a(t) = -x_a(t) + 9.5 \quad ; \quad x_a(0) = 0$$

with $\Delta Q = 1$ and different values of ε are:



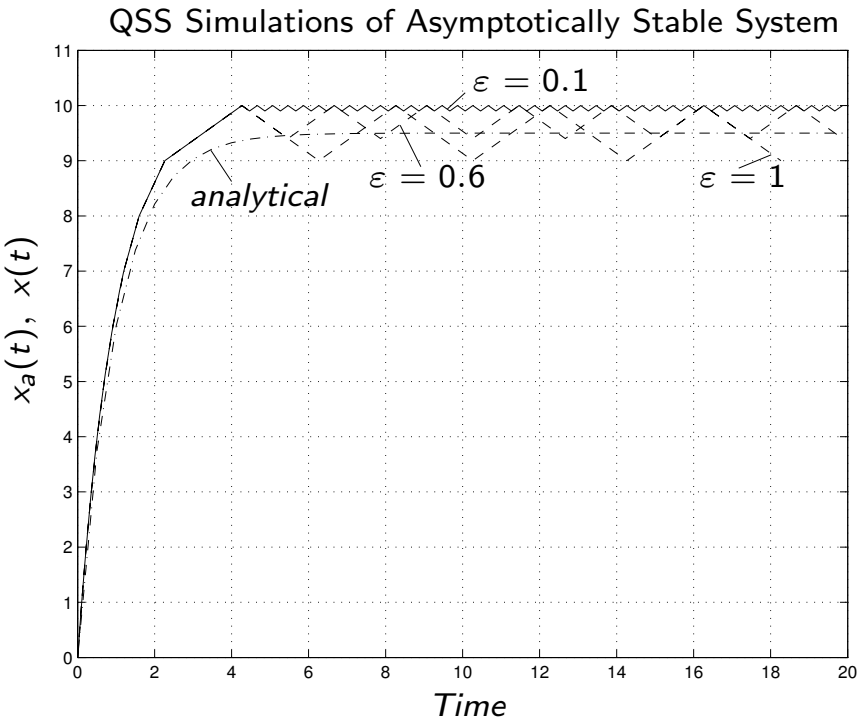
- To *reduce the frequency* (and with it the number of integration steps), it is convenient to choose a large value of ε.

Choosing Quantum and Hysteresis Width

The simulation results generated by the QSS solver usually exhibit steady-state *oscillations*. Their *frequency* depends on the *hysteresis*. For example, the output trajectories of the scalar system:

$$\dot{x}_a(t) = -x_a(t) + 9.5 \quad ; \quad x_a(0) = 0$$

with $\Delta Q = 1$ and different values of ε are:



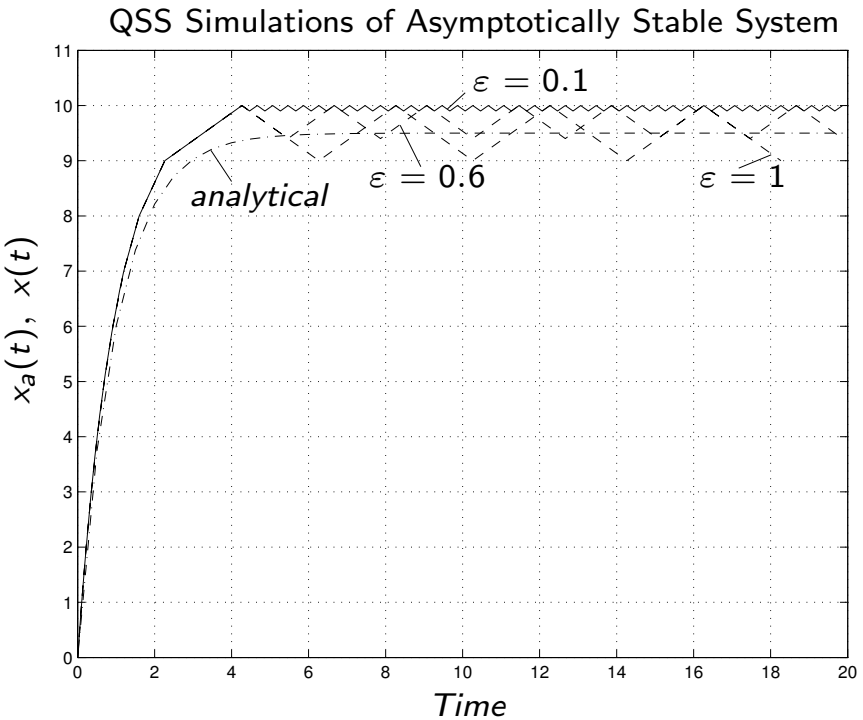
- To *reduce the frequency* (and with it the number of integration steps), it is convenient to choose a large value of ε.
- As the *error bound* depends on $\max(\Delta Q, \varepsilon)$, the best choice is to set $\Delta Q = \varepsilon$.

Choosing Quantum and Hysteresis Width

The simulation results generated by the QSS solver usually exhibit steady-state *oscillations*. Their *frequency* depends on the *hysteresis*. For example, the output trajectories of the scalar system:

$$\dot{x}_a(t) = -x_a(t) + 9.5 \quad ; \quad x_a(0) = 0$$

with $\Delta Q = 1$ and different values of ε are:



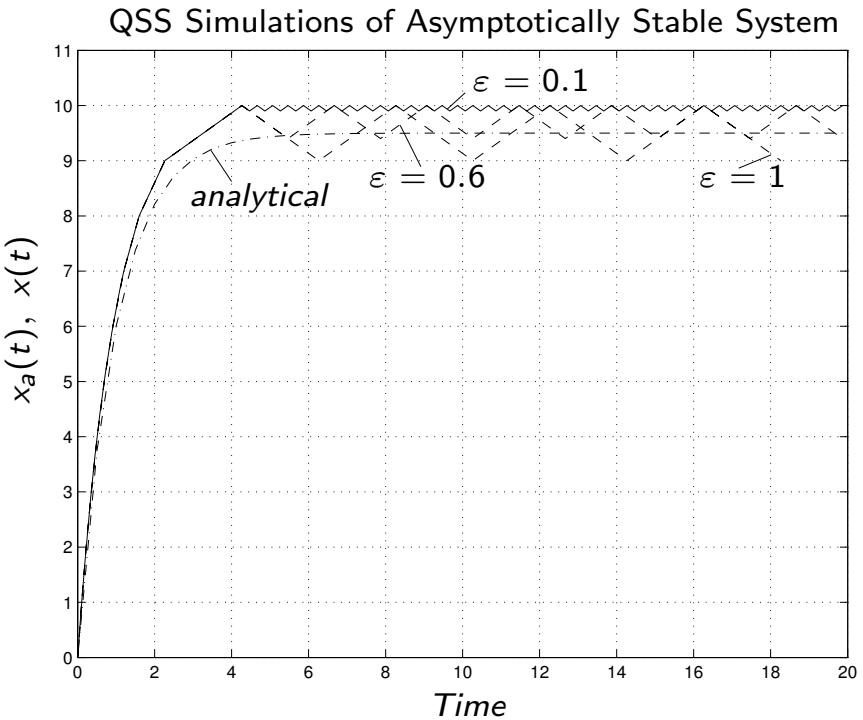
- To *reduce the frequency* (and with it the number of integration steps), it is convenient to choose a large value of ε.
- As the *error bound* depends on $\max(\Delta Q, \varepsilon)$, the best choice is to set $\Delta Q = \varepsilon$.
- The error bound depends *linearly* on ΔQ. Thus, the *quantum* should be chosen *proportional* to the tolerated error in every variable.

Choosing Quantum and Hysteresis Width

The simulation results generated by the QSS solver usually exhibit steady-state *oscillations*. Their *frequency* depends on the *hysteresis*. For example, the output trajectories of the scalar system:

$$\dot{x}_a(t) = -x_a(t) + 9.5 \quad ; \quad x_a(0) = 0$$

with $\Delta Q = 1$ and different values of ε are:



- To *reduce the frequency* (and with it the number of integration steps), it is convenient to choose a large value of ε.
- As the *error bound* depends on $\max(\Delta Q, \varepsilon)$, the best choice is to set $\Delta Q = \varepsilon$.
- The error bound depends *linearly* on ΔQ. Thus, the *quantum* should be chosen *proportional* to the tolerated error in every variable.
- A *practical rule* is to choose the quantum proportional to the *amplitude* of every state variable.

Input Signals in the QSS Method

In order to incorporate *input signals* $\mathbf{u}(t)$ in a QSS simulation, we need to construct *DEVS models* that generate and couple these models with the *quantized integrators* and the *static functions*.

Input Signals in the QSS Method

In order to incorporate *input signals* $\mathbf{u}(t)$ in a QSS simulation, we need to construct *DEVS models* that generate and couple these models with the *quantized integrators* and the *static functions*.

Programming a DEVS model that generates a *piecewise constant input signal* is trivial. The **PowerDEVS** library offers various *source signals* of this type: a step function, a square-wave function, a pulse function, etc.

Input Signals in the QSS Method

In order to incorporate *input signals* $\mathbf{u}(t)$ in a QSS simulation, we need to construct *DEVS models* that generate and couple these models with the *quantized integrators* and the *static functions*.

Programming a DEVS model that generates a *piecewise constant input signal* is trivial. The **PowerDEVS** library offers various *source signals* of this type: a step function, a square-wave function, a pulse function, etc.

Yet, also other types of input signals can be generated quite easily by *approximating* them through piecewise constant signals. **PowerDEVS** offers source signals approximating sinusoidal inputs, ramps, triangular inputs, etc.

Input Signals in the QSS Method

In order to incorporate *input signals* $\mathbf{u}(t)$ in a QSS simulation, we need to construct *DEVS models* that generate and couple these models with the *quantized integrators* and the *static functions*.

Programming a DEVS model that generates a *piecewise constant input signal* is trivial. The **PowerDEVS** library offers various *source signals* of this type: a step function, a square-wave function, a pulse function, etc.

Yet, also other types of input signals can be generated quite easily by *approximating* them through piecewise constant signals. **PowerDEVS** offers source signals approximating sinusoidal inputs, ramps, triangular inputs, etc.

However, when we approximate an input signal by a piecewise constant trajectory, we introduce a *new type of error*. In this case, the expression for the *global error bound* needs to be augmented with an additional term:

$$|\mathbf{x}(t) - \mathbf{x_a}(t)| \leq |\mathbf{V}| \cdot |\mathbb{R}\mathrm{e}\{\mathbf{\Lambda}\}^{-1} \cdot \mathbf{\Lambda}| \cdot |\mathbf{V}^{-1}| \cdot \mathbf{\Delta Q} + |\mathbf{V}| \cdot |\mathbb{R}\mathrm{e}\{\mathbf{\Lambda}\}^{-1} \cdot \mathbf{V}^{-1} \cdot \mathbf{B}| \cdot \mathbf{\Delta u}$$

Cost vs. Precision in QSS

Although the QSS method exhibits many attractive features, its *principal limitation* lies in the fact that it represents a *first-order accurate approximation* only.

Cost vs. Precision in QSS

Although the QSS method exhibits many attractive features, its *principal limitation* lies in the fact that it represents a *first-order accurate approximation* only.

The equation for the error bound demonstrates that the *error is proportional to the quantum*.

Cost vs. Precision in QSS

Although the QSS method exhibits many attractive features, its *principal limitation* lies in the fact that it represents a *first-order accurate approximation* only.

The equation for the error bound demonstrates that the *error is proportional to the quantum*.

This means that the number of integration steps grows *inversely proportional to the tolerated error*. For this reason, if we tighten our *accuracy requirements* hundredfold, we need to reduce the *quantum* by a factor of 100, which in turn increases the *number of integration steps* by a factor of 100.

Cost vs. Precision in QSS

Although the QSS method exhibits many attractive features, its *principal limitation* lies in the fact that it represents a *first-order accurate approximation* only.

The equation for the error bound demonstrates that the *error is proportional to the quantum*.

This means that the number of integration steps grows *inversely proportional to the tolerated error*. For this reason, if we tighten our *accuracy requirements* hundredfold, we need to reduce the *quantum* by a factor of 100, which in turn increases the *number of integration steps* by a factor of 100.

We can see this easily when looking at the approximation of an input ramp signal. The signal gets represented by a stair case. The more accurate we need to represent the ramp, the smaller we need to make the individual steps, and the more steps we shall need.

Cost vs. Precision in QSS

Although the QSS method exhibits many attractive features, its *principal limitation* lies in the fact that it represents a *first-order accurate approximation* only.

The equation for the error bound demonstrates that the *error is proportional to the quantum*.

This means that the number of integration steps grows *inversely proportional to the tolerated error*. For this reason, if we tighten our *accuracy requirements* hundredfold, we need to reduce the *quantum* by a factor of 100, which in turn increases the *number of integration steps* by a factor of 100.

We can see this easily when looking at the approximation of an input ramp signal. The signal gets represented by a stair case. The more accurate we need to represent the ramp, the smaller we need to make the individual steps, and the more steps we shall need.

This is unacceptable for more stringent accuracy requirements. Most engineering applications call for relative error tolerances of 10^{-4} or better. For such accuracy requirements, QSS turns out to be highly inefficient. We consequently need *QSS methods of higher orders of approximation accuracy*.

First-order Quantizer

The *quantizer* used in QSS, with $\Delta Q = \varepsilon$, produces a *piecewise constant* signal $q(t)$ that changes its value when $|q(t) - x(t)| = \Delta Q$.

First-order Quantizer

The *quantizer* used in QSS, with $\Delta Q = \varepsilon$, produces a *piecewise constant* signal $q(t)$ that changes its value when $|q(t) - x(t)| = \Delta Q$.

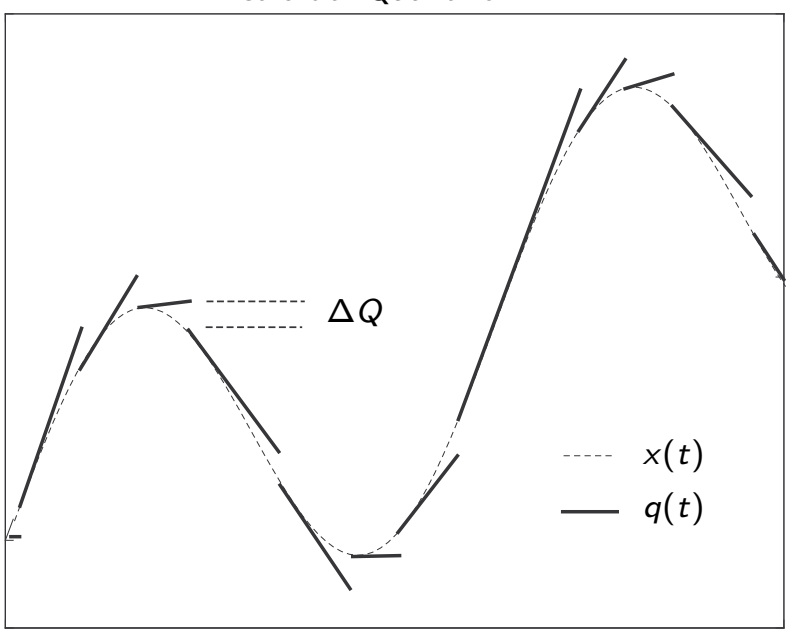
One way to *increment the order* of the approximation is to use a quantizer with an output that is *piecewise linear*, the gradient of which changes whenever $|q(t) - x(t)| = \Delta Q$.

First-order Quantizer

The *quantizer* used in QSS, with $\Delta Q = \varepsilon$, produces a *piecewise constant* signal $q(t)$ that changes its value when $|q(t) - x(t)| = \Delta Q$.

One way to *increment the order* of the approximation is to use a quantizer with an output that is *piecewise linear*, the gradient of which changes whenever $|q(t) - x(t)| = \Delta Q$.

First-order Quantizer

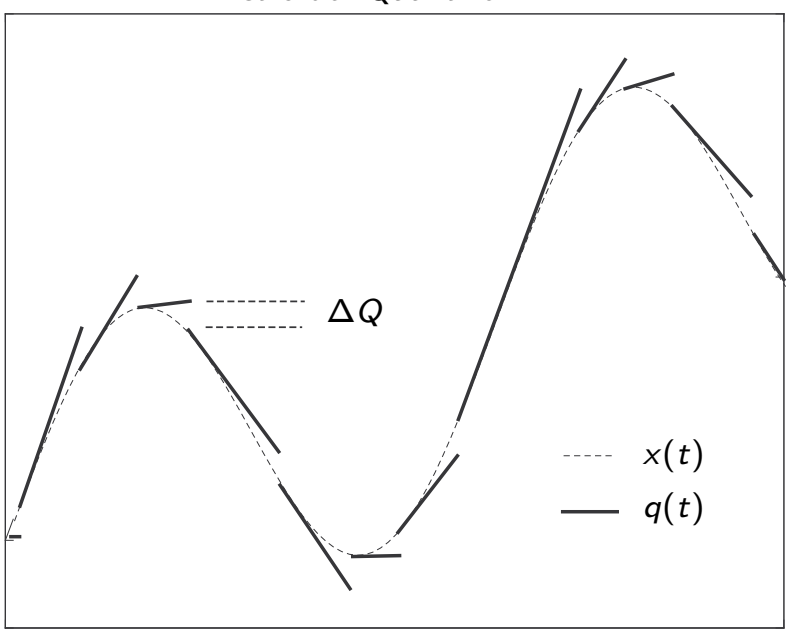


First-order Quantizer

The *quantizer* used in QSS, with $\Delta Q = \varepsilon$, produces a *piecewise constant* signal $q(t)$ that changes its value when $|q(t) - x(t)| = \Delta Q$.

One way to *increment the order* of the approximation is to use a quantizer with an output that is *piecewise linear*, the gradient of which changes whenever $|q(t) - x(t)| = \Delta Q$.

First-order Quantizer



We say that $x(t)$ and $q(t)$ are related by a *first-order quantization function* if:

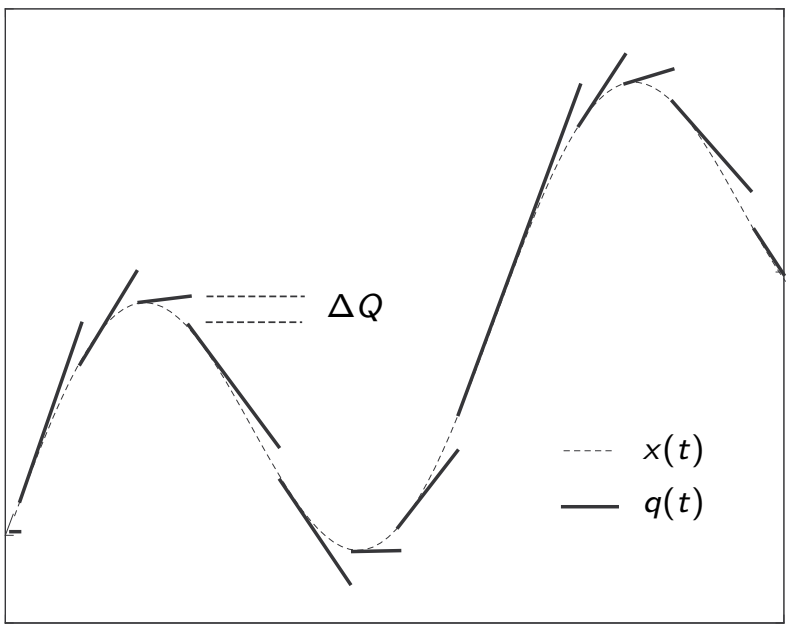
$$q(t) = \begin{cases} x(t), & \text{if } t = t_0 \vee |q(t^-) - x(t^-)| = \Delta Q \\ q(t_j) + m_j \cdot (t - t_j), & \text{otherwise} \end{cases}$$

First-order Quantizer

The *quantizer* used in QSS, with $\Delta Q = \varepsilon$, produces a *piecewise constant* signal $q(t)$ that changes its value when $|q(t) - x(t)| = \Delta Q$.

One way to *increment the order* of the approximation is to use a quantizer with an output that is *piecewise linear*, the gradient of which changes whenever $|q(t) - x(t)| = \Delta Q$.

First-order Quantizer



We say that $x(t)$ and $q(t)$ are related by a *first-order quantization function* if:

$$q(t) = \begin{cases} x(t), & \text{if } t = t_0 \vee |q(t^-) - x(t^-)| = \Delta Q \\ q(t_j) + m_j \cdot (t - t_j), & \text{otherwise} \end{cases}$$

The sequence $t_0, \ldots, t_j, \ldots$ is defined as:

$$t_{j+1} = \min(t > t_j), \quad \text{where } |x(t_j) + m_j \cdot (t - t_j) - x(t)| = \Delta Q$$

and the gradients are $m_0 = 0$, $m_j = \dot{x}(t_j^-)$ for $j = 1, \ldots, k, \ldots$.

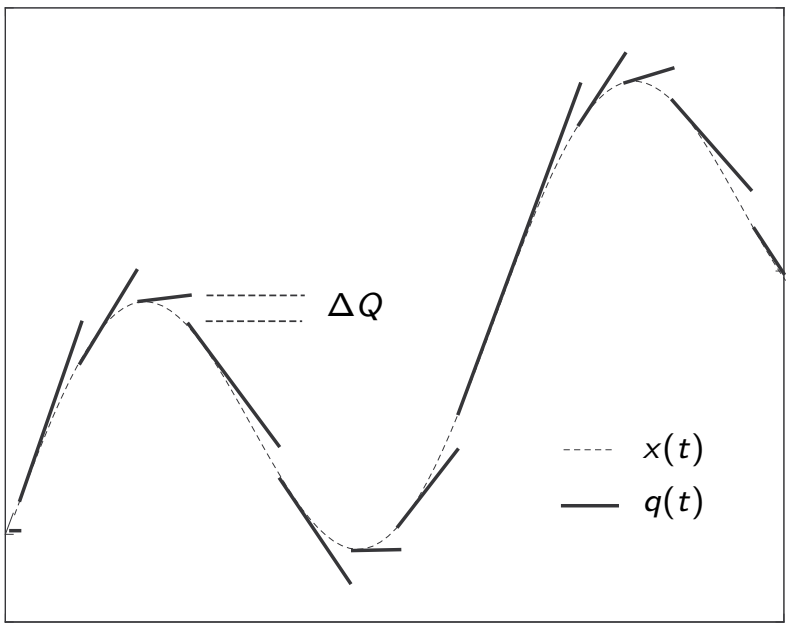
First-order Quantizer II

- The way, the *first-order quantizer function* has been defined, implies the presence of *hysteresis* with $\varepsilon = \Delta Q$.

First-order Quantizer II

- The way, the *first-order quantizer function* has been defined, implies the presence of *hysteresis* with $\varepsilon = \Delta Q$.

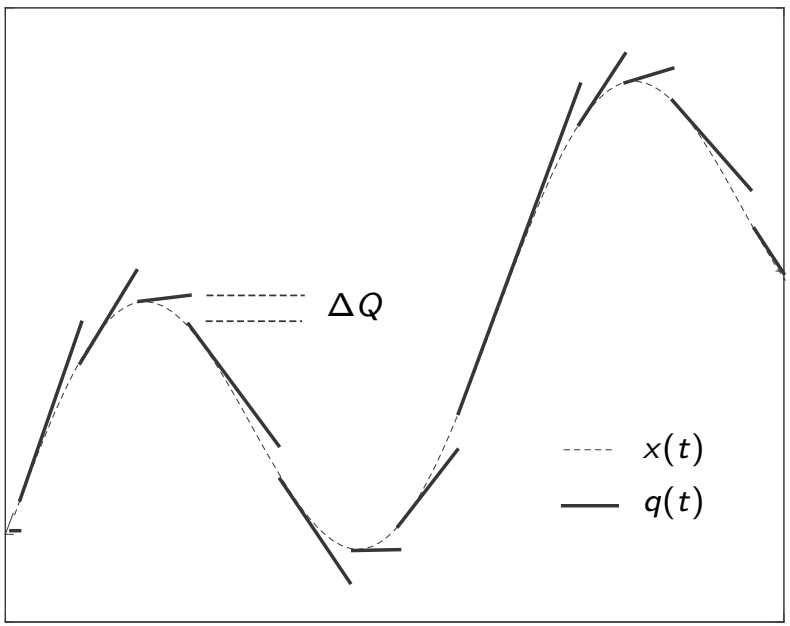
First-order Quantizer



First-order Quantizer II

- The way, the *first-order quantizer function* has been defined, implies the presence of *hysteresis* with $\varepsilon = \Delta Q$.

First-order Quantizer

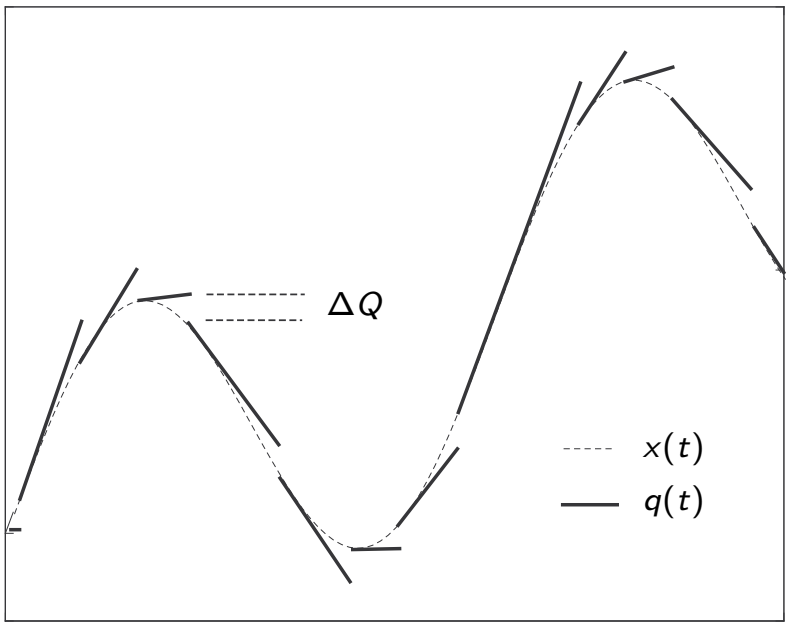


- This can be seen easily from the figure to the left, where the linear approximation $q(t)$ is sometimes *above* and sometimes *below* the continuous curve $x(t)$ depending on the sign of the second derivative of $x(t)$.

First-order Quantizer II

- The way, the *first-order quantizer function* has been defined, implies the presence of *hysteresis* with $\varepsilon = \Delta Q$.

First-order Quantizer

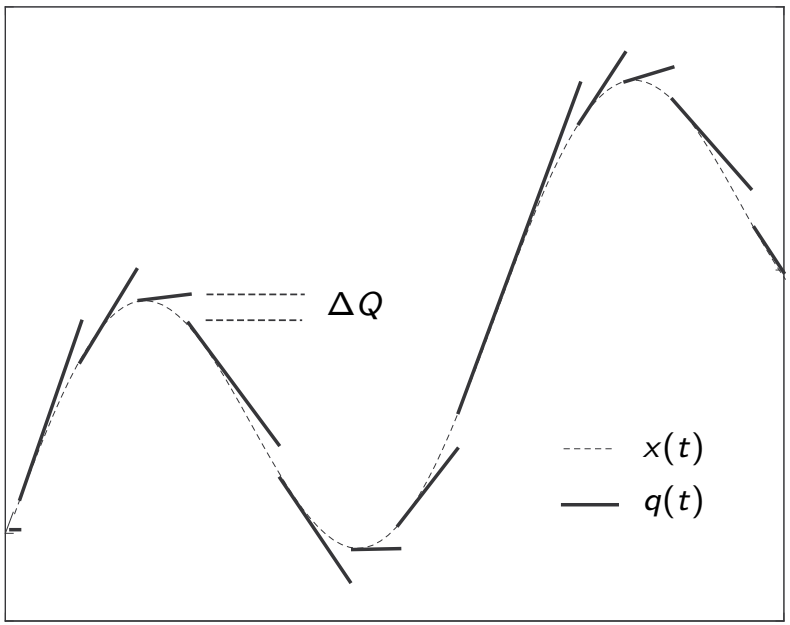


- This can be seen easily from the figure to the left, where the linear approximation $q(t)$ is sometimes *above* and sometimes *below* the continuous curve $x(t)$ depending on the sign of the second derivative of $x(t)$.
- $q(t)$ is above $x(t)$ when the second derivative of $x(t)$ is negative, and it is below $x(t)$ otherwise. $q(t)$ always starts at $x(t)$, and the linear approximation continues, until it deviates from $x(t)$ by $\pm\Delta Q$.

First-order Quantizer II

- The way, the *first-order quantizer function* has been defined, implies the presence of *hysteresis* with $\varepsilon = \Delta Q$.

First-order Quantizer



- This can be seen easily from the figure to the left, where the linear approximation $q(t)$ is sometimes *above* and sometimes *below* the continuous curve $x(t)$ depending on the sign of the second derivative of $x(t)$.
- $q(t)$ is above $x(t)$ when the second derivative of $x(t)$ is negative, and it is below $x(t)$ otherwise. $q(t)$ always starts at $x(t)$, and the linear approximation continues, until it deviates from $x(t)$ by $\pm\Delta Q$.
- This avoids the possibility of occurrence of *illegitimate models*.

Second-order accurate QSS Method (QSS2)

The *second-order accurate QSS method (QSS2)* is defined in an identical manner to the QSS method, with the exception that the state variables x_i are now related to the quantized states q_i through *first-order quantization functions*.

Second-order accurate QSS Method (QSS2)

The *second-order accurate QSS method (QSS2)* is defined in an identical manner to the QSS method, with the exception that the state variables x_i are now related to the quantized states q_i through *first-order quantization functions*.

Given the non-linear continuous system:

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

Its *QSS2 approximation* can be written as:

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

Second-order accurate QSS Method (QSS2)

The *second-order accurate QSS method (QSS2)* is defined in an identical manner to the QSS method, with the exception that the state variables x_i are now related to the quantized states q_i through *first-order quantization functions*.

Given the non-linear continuous system:

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

Its *QSS2 approximation* can be written as:

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

We now accept that the *input variables* $u_j(t)$ have or are approximated by signals that are *piecewise linear*.

Second-order accurate QSS Method (QSS2)

The *second-order accurate QSS method (QSS2)* is defined in an identical manner to the QSS method, with the exception that the state variables x_i are now related to the quantized states q_i through *first-order quantization functions*.

Given the non-linear continuous system:

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

Its *QSS2 approximation* can be written as:

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

We now accept that the *input variables* $u_j(t)$ have or are approximated by signals that are *piecewise linear*.

When using QSS2, a ramp input signal does not cause any events at all, as it can be represented accurately by a linear trajectory.

Trajectories in the QSS2 Method

In accordance with the definition of QSS2, the *trajectories of the approximated system* assume the following form:

Trajectories in the QSS2 Method

In accordance with the definition of QSS2, the *trajectories of the approximated system* assume the following form:

- The *quantized variables* $q_i(t)$ are now *piecewise linear*.

Trajectories in the QSS2 Method

In accordance with the definition of QSS2, the *trajectories of the approximated system* assume the following form:

- The *quantized variables* $q_i(t)$ are now *piecewise linear*.
- The *input signals* $u_j(t)$ are also *piecewise linear*.

Trajectories in the QSS2 Method

In accordance with the definition of QSS2, the *trajectories of the approximated system* assume the following form:

- The *quantized variables* $q_i(t)$ are now *piecewise linear*.
- The *input signals* $u_j(t)$ are also *piecewise linear*.
- In the case of a *linear time-invariant system* are also the *state derivatives* $\dot{x}_i(t)$ *piecewise linear*.

Trajectories in the QSS2 Method

In accordance with the definition of QSS2, the *trajectories of the approximated system* assume the following form:

- The *quantized variables* $q_i(t)$ are now *piecewise linear*.
- The *input signals* $u_j(t)$ are also *piecewise linear*.
- In the case of a *linear time-invariant system* are also the *state derivatives* $\dot{x}_i(t)$ *piecewise linear*.
- In that case are the *state variables* $x_i(t)$ *piecewise parabolic*.

Trajectories in the QSS2 Method

In accordance with the definition of QSS2, the *trajectories of the approximated system* assume the following form:

- The *quantized variables* $q_i(t)$ are now *piecewise linear*.
- The *input signals* $u_j(t)$ are also *piecewise linear*.
- In the case of a *linear time-invariant system* are also the *state derivatives* $\dot{x}_i(t)$ *piecewise linear*.
- In that case are the *state variables* $x_i(t)$ *piecewise parabolic*.
- In the general non-linear case do neither the state variables nor their derivatives assume any special exploitable form.

Trajectories in the QSS2 Method

In accordance with the definition of QSS2, the *trajectories of the approximated system* assume the following form:

- The *quantized variables* $q_i(t)$ are now *piecewise linear*.
- The *input signals* $u_j(t)$ are also *piecewise linear*.
- In the case of a *linear time-invariant system* are also the *state derivatives* $\dot{x}_i(t)$ *piecewise linear*.
- In that case are the *state variables* $x_i(t)$ *piecewise parabolic*.
- In the general non-linear case do neither the state variables nor their derivatives assume any special exploitable form.
- In that case shall the state derivatives be *approximated* by *piecewise linear trajectories*, and consequently will the approximated state variables turn out to be *piecewise parabolic*.

Trajectories in the QSS2 Method

In accordance with the definition of QSS2, the *trajectories of the approximated system* assume the following form:

- The *quantized variables* $q_i(t)$ are now *piecewise linear*.
- The *input signals* $u_j(t)$ are also *piecewise linear*.
- In the case of a *linear time-invariant system* are also the *state derivatives* $\dot{x}_i(t)$ *piecewise linear*.
- In that case are the *state variables* $x_i(t)$ *piecewise parabolic*.
- In the general non-linear case do neither the state variables nor their derivatives assume any special exploitable form.
- In that case shall the state derivatives be *approximated* by *piecewise linear trajectories*, and consequently will the approximated state variables turn out to be *piecewise parabolic*.

The simulation results do, in the non-linear case, not coincide exactly with the definition of the QSS2 methods, but rather represent an approximation only.

DEVS Representation of QSS2

The basic idea for obtaining an equivalent DEVS model of a QSS2 approximation is the same as in QSS, i.e., we divide the system to be modeled into *static functions* and *quantized integrators*. The following differences are to be noted:

DEVS Representation of QSS2

The basic idea for obtaining an equivalent DEVS model of a QSS2 approximation is the same as in QSS, i.e., we divide the system to be modeled into *static functions* and *quantized integrators*. The following differences are to be noted:

- As the trajectories are *piecewise linear*, every event must now carry two numbers: the *value* of the variable at the beginning of the segment and the *derivative* of the variable throughout the segment.

DEVS Representation of QSS2

The basic idea for obtaining an equivalent DEVS model of a QSS2 approximation is the same as in QSS, i.e., we divide the system to be modeled into *static functions* and *quantized integrators*. The following differences are to be noted:

- As the trajectories are *piecewise linear*, every event must now carry two numbers: the *value* of the variable at the beginning of the segment and the *derivative* of the variable throughout the segment.
- Consequently, the DEVS models of the *static functions* must now take into account the *values* and the *derivatives* of its inputs, and calculate from that information the *values* and the *derivatives* of the outputs.

DEVS Representation of QSS2

The basic idea for obtaining an equivalent DEVS model of a QSS2 approximation is the same as in QSS, i.e., we divide the system to be modeled into *static functions* and *quantized integrators*. The following differences are to be noted:

- As the trajectories are *piecewise linear*, every event must now carry two numbers: the *value* of the variable at the beginning of the segment and the *derivative* of the variable throughout the segment.
- Consequently, the DEVS models of the *static functions* must now take into account the *values* and the *derivatives* of its inputs, and calculate from that information the *values* and the *derivatives* of the outputs.
- The DEVS models of the *quantized integrators* must also take into account the derivatives. As the state trajectories are now *piecewise parabolic*, a quadratic equation must now be solved in every step to calculate the time instant at which $|x_i(t) - q_i(t)| = \Delta Q_i$.

DEVS Representation of QSS2

The basic idea for obtaining an equivalent DEVS model of a QSS2 approximation is the same as in QSS, i.e., we divide the system to be modeled into *static functions* and *quantized integrators*. The following differences are to be noted:

- As the trajectories are *piecewise linear*, every event must now carry two numbers: the *value* of the variable at the beginning of the segment and the *derivative* of the variable throughout the segment.
- Consequently, the DEVS models of the *static functions* must now take into account the *values* and the *derivatives* of its inputs, and calculate from that information the *values* and the *derivatives* of the outputs.
- The DEVS models of the *quantized integrators* must also take into account the derivatives. As the state trajectories are now *piecewise parabolic*, a quadratic equation must now be solved in every step to calculate the time instant at which $|x_i(t) - q_i(t)| = \Delta Q_i$.

Except for these differences, the resulting DEVS model will look exactly the same for **QSS2** as for **QSS**.

Simulation with QSS2 in PowerDEVS

PowerDEVS offers as part of its libraries a DEVS model of the *quantized integrator*, many *static function models*, and a good collection of *source input signals*.

Simulation with QSS2 in PowerDEVS

PowerDEVS offers as part of its libraries a DEVS model of the *quantized integrator*, many *static function models*, and a good collection of *source input signals*.

PowerDEVS offers only a single quantized integrator model. The order of approximation accuracy, i.e., whether the simulation should use QSS, QSS2, QSS3, or even QSS4 can be specified by the user for each integrator separately by setting a parameter value. QSS3 is currently the default value used.

Simulation with QSS2 in PowerDEVS

PowerDEVS offers as part of its libraries a DEVS model of the *quantized integrator*, many *static function models*, and a good collection of *source input signals*.

PowerDEVS offers only a single quantized integrator model. The order of approximation accuracy, i.e., whether the simulation should use QSS, QSS2, QSS3, or even QSS4 can be specified by the user for each integrator separately by setting a parameter value. QSS3 is currently the default value used.

Also the DEVS models of the *source inputs* allow the user to select the order of approximation accuracy of the source by setting a parameter value.

Simulation with QSS2 in PowerDEVS

PowerDEVS offers as part of its libraries a DEVS model of the *quantized integrator*, many *static function models*, and a good collection of *source input signals*.

PowerDEVS offers only a single quantized integrator model. The order of approximation accuracy, i.e., whether the simulation should use QSS, QSS2, QSS3, or even QSS4 can be specified by the user for each integrator separately by setting a parameter value. QSS3 is currently the default value used.

Also the DEVS models of the *source inputs* allow the user to select the order of approximation accuracy of the source by setting a parameter value.

The models of the *static functions* have all at least one input and are therefore capable to determine the appropriate order of approximation accuracy on their own.

Simulation with QSS2 in PowerDEVS

PowerDEVS offers as part of its libraries a DEVS model of the *quantized integrator*, many *static function models*, and a good collection of *source input signals*.

PowerDEVS offers only a single quantized integrator model. The order of approximation accuracy, i.e., whether the simulation should use QSS, QSS2, QSS3, or even QSS4 can be specified by the user for each integrator separately by setting a parameter value. QSS3 is currently the default value used.

Also the DEVS models of the *source inputs* allow the user to select the order of approximation accuracy of the source by setting a parameter value.

The models of the *static functions* have all at least one input and are therefore capable to determine the appropriate order of approximation accuracy on their own.

Consequently, the *block diagram* of a model specified in **PowerDEVS** looks exactly the same for all integration methods. The user can select the integration method by setting parameters at all integrators and sources.

Simulation with QSS2 in PowerDEVS

PowerDEVS offers as part of its libraries a DEVS model of the *quantized integrator*, many *static function models*, and a good collection of *source input signals*.

PowerDEVS offers only a single quantized integrator model. The order of approximation accuracy, i.e., whether the simulation should use QSS, QSS2, QSS3, or even QSS4 can be specified by the user for each integrator separately by setting a parameter value. QSS3 is currently the default value used.

Also the DEVS models of the *source inputs* allow the user to select the order of approximation accuracy of the source by setting a parameter value.

The models of the *static functions* have all at least one input and are therefore capable to determine the appropriate order of approximation accuracy on their own.

Consequently, the *block diagram* of a model specified in **PowerDEVS** looks exactly the same for all integration methods. The user can select the integration method by setting parameters at all integrators and sources.

Since **PowerDEVS** supports *mixed-mode integration*, these parameters need to be set for each integrator and source separately.

Stability and Accuracy of QSS2

The QSS2 approximation of a *linear time-invariant system*:

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

takes the same form as the QSS approximation:

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

Stability and Accuracy of QSS2

The QSS2 approximation of a *linear time-invariant system*:

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

takes the same form as the QSS approximation:

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

Also in QSS2, it is true that:

$$|q_i(t) - x_i(t)| \leq \Delta Q_i \quad \forall t \geq t_0$$

Stability and Accuracy of QSS2

The QSS2 approximation of a *linear time-invariant system*:

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

takes the same form as the QSS approximation:

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

Also in QSS2, it is true that:

$$|q_i(t) - x_i(t)| \leq \Delta Q_i \quad \forall t \geq t_0$$

For this reason, QSS and QSS2 share the same *perturbed representation* in the case of linear time-invariant systems.

Stability and Accuracy of QSS2

The QSS2 approximation of a *linear time-invariant system*:

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

takes the same form as the QSS approximation:

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

Also in QSS2, it is true that:

$$|q_i(t) - x_i(t)| \leq \Delta Q_i \quad \forall t \geq t_0$$

For this reason, QSS and QSS2 share the same *perturbed representation* in the case of linear time-invariant systems.

As the formula for the *global error bound* of QSS has been deduced directly from that representation, we can conclude that the QSS2 approximation is subject to the same error bound.

Stability and Accuracy of QSS2

The QSS2 approximation of a *linear time-invariant system*:

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

takes the same form as the QSS approximation:

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

Also in QSS2, it is true that:

$$|q_i(t) - x_i(t)| \leq \Delta Q_i \quad \forall t \geq t_0$$

For this reason, QSS and QSS2 share the same *perturbed representation* in the case of linear time-invariant systems.

As the formula for the *global error bound* of QSS has been deduced directly from that representation, we can conclude that the QSS2 approximation is subject to the same error bound.

In the case of linear time-invariant systems, QSS and QSS2 share the *identical stability and accuracy properties*.

Cost vs. Accuracy in QSS2

From the previous analysis, we conclude that, by using either QSS or QSS2 with the *same quantum*, we obtain simulation results of *similar accuracy*.

Cost vs. Accuracy in QSS2

From the previous analysis, we conclude that, by using either QSS or QSS2 with the *same quantum*, we obtain simulation results of *similar accuracy*.

What is then the advantage of using QSS2?

Cost vs. Accuracy in QSS2

From the previous analysis, we conclude that, by using either QSS or QSS2 with the *same quantum*, we obtain simulation results of *similar accuracy*.

What is then the advantage of using QSS2?

- In QSS, we saw that, by reducing the *quantum* 100 times, the *number of integration steps* increased by a factor of 100.

Cost vs. Accuracy in QSS2

From the previous analysis, we conclude that, by using either QSS or QSS2 with the *same quantum*, we obtain simulation results of *similar accuracy*.

What is then the advantage of using QSS2?

- In QSS, we saw that, by reducing the *quantum* 100 times, the *number of integration steps* increased by a factor of 100.
- In QSS2, it can be seen easily that, by reducing the *quantum* 100 times, the *number of integration steps* increases by a factor of 10 only.

Cost vs. Accuracy in QSS2

From the previous analysis, we conclude that, by using either QSS or QSS2 with the *same quantum*, we obtain simulation results of *similar accuracy*.

What is then the advantage of using QSS2?

- In QSS, we saw that, by reducing the *quantum* 100 times, the *number of integration steps* increased by a factor of 100.
- In QSS2, it can be seen easily that, by reducing the *quantum* 100 times, the *number of integration steps* increases by a factor of 10 only.
- More precisely, the cost grows *inversely proportional* to the *square root* of the tolerated error in QSS2.

Cost vs. Accuracy in QSS2

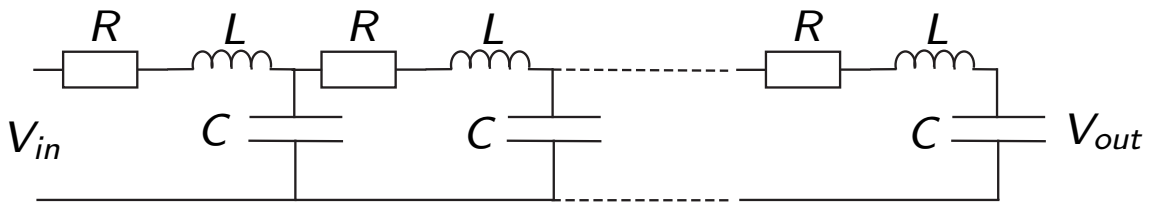
From the previous analysis, we conclude that, by using either QSS or QSS2 with the *same quantum*, we obtain simulation results of *similar accuracy*.

What is then the advantage of using QSS2?

- In QSS, we saw that, by reducing the *quantum* 100 times, the *number of integration steps* increased by a factor of 100.
- In QSS2, it can be seen easily that, by reducing the *quantum* 100 times, the *number of integration steps* increases by a factor of 10 only.
- More precisely, the cost grows *inversely proportional* to the *square root* of the tolerated error in QSS2.
- For example, if we wish to improve the accuracy of a simulation by a factor of $10,000$, the cost increases by a factor of $10,000$ when using QSS, whereas it only increases by a factor of 100 when using QSS2.

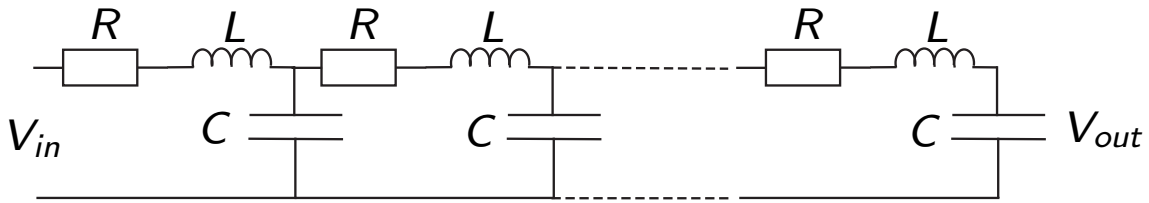
An Illustrative Example

The circuit below shows a *model with concentrated parameters* of a *transmission line*. In our case, the circuit represents a section of the path of an *integrated circuit* that transmits data at a *very high frequency*.



An Illustrative Example

The circuit below shows a *model with concentrated parameters* of a *transmission line*. In our case, the circuit represents a section of the path of an *integrated circuit* that transmits data at a *very high frequency*.



Let us consider 5 segments. There results a *linear time-invariant system* of order 10 with a *band-structured sparse system matrix*:

$$\mathbf{A} = \begin{pmatrix} -R/L & -1/L & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1/C & 0 & -1/C & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1/L & -R/L & -1/L & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1/C & 0 & -1/C & 0 & 0 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1/C & 0 \end{pmatrix}$$

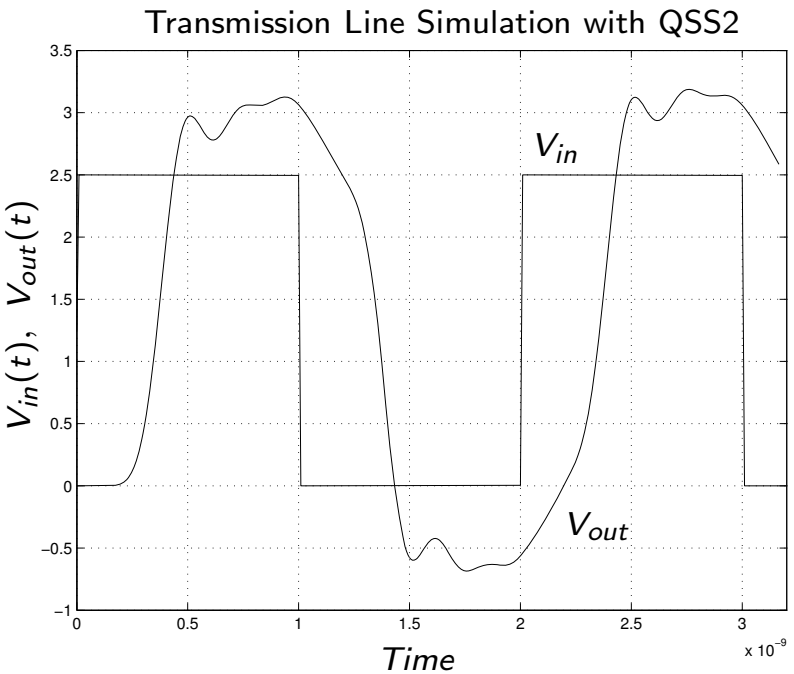
An Illustrative Example II

Using **PowerDEVS**, we constructed the *block diagram* and simulated the circuit using QSS2 with a *trapezoidal input* (the typical form of signals in integrated digital circuits). We used a quantum of $4mV$ for the potentials and $10\mu A$ for the currents.

An Illustrative Example II

Using **PowerDEVS**, we constructed the *block diagram* and simulated the circuit using QSS2 with a *trapezoidal input* (the typical form of signals in integrated digital circuits). We used a quantum of $4mV$ for the potentials and $10\mu A$ for the currents.

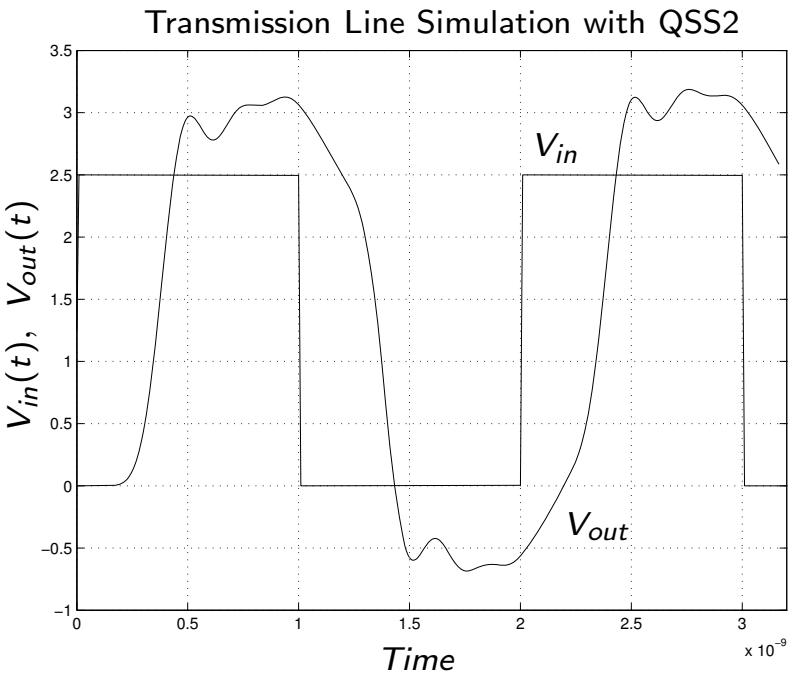
The simulation results were:



An Illustrative Example II

Using **PowerDEVS**, we constructed the *block diagram* and simulated the circuit using QSS2 with a *trapezoidal input* (the typical form of signals in integrated digital circuits). We used a quantum of $4mV$ for the potentials and $10\mu A$ for the currents.

The simulation results were:

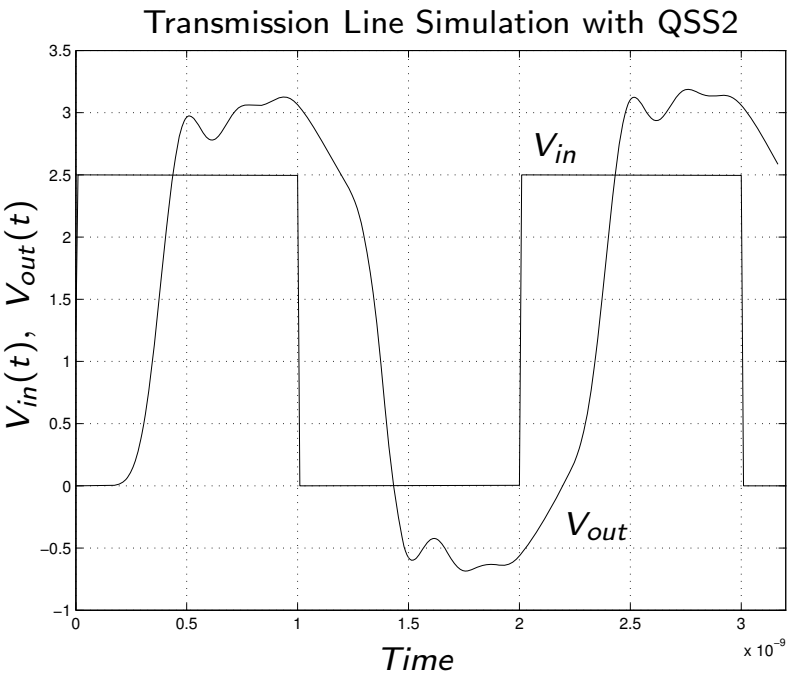


- The simulation took 2536 steps (between 198 and 319 steps for *each integrator*).

An Illustrative Example II

Using **PowerDEVS**, we constructed the *block diagram* and simulated the circuit using QSS2 with a *trapezoidal input* (the typical form of signals in integrated digital circuits). We used a quantum of $4mV$ for the potentials and $10\mu A$ for the currents.

The simulation results were:

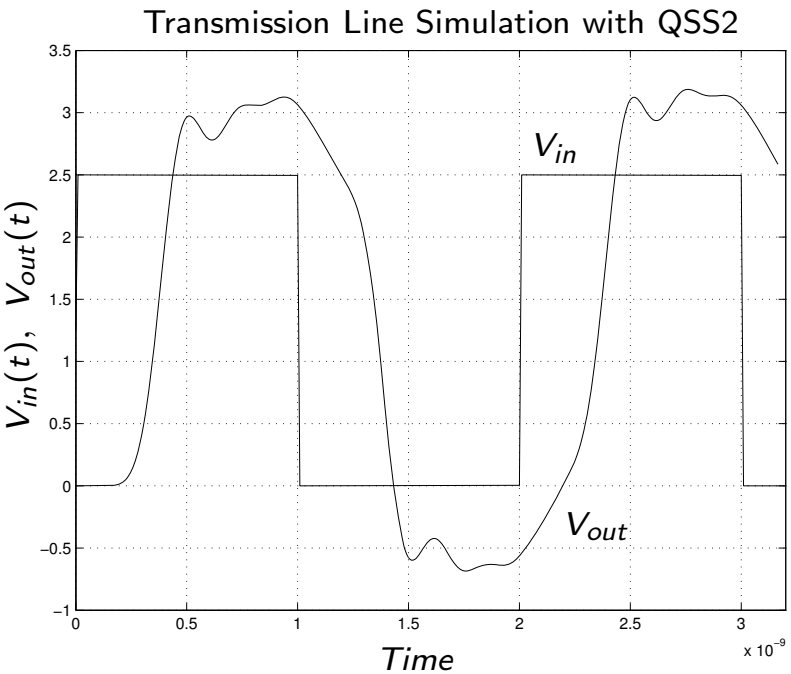


- The simulation took 2536 steps (between 198 and 319 steps for *each integrator*).
- Although the number of steps seems to be quite large, every single step only leads to *scalar calculations* in two or three integrators due to the *sparsity* of the **A**-matrix.

An Illustrative Example II

Using **PowerDEVS**, we constructed the *block diagram* and simulated the circuit using QSS2 with a *trapezoidal input* (the typical form of signals in integrated digital circuits). We used a quantum of $4mV$ for the potentials and $10\mu A$ for the currents.

The simulation results were:

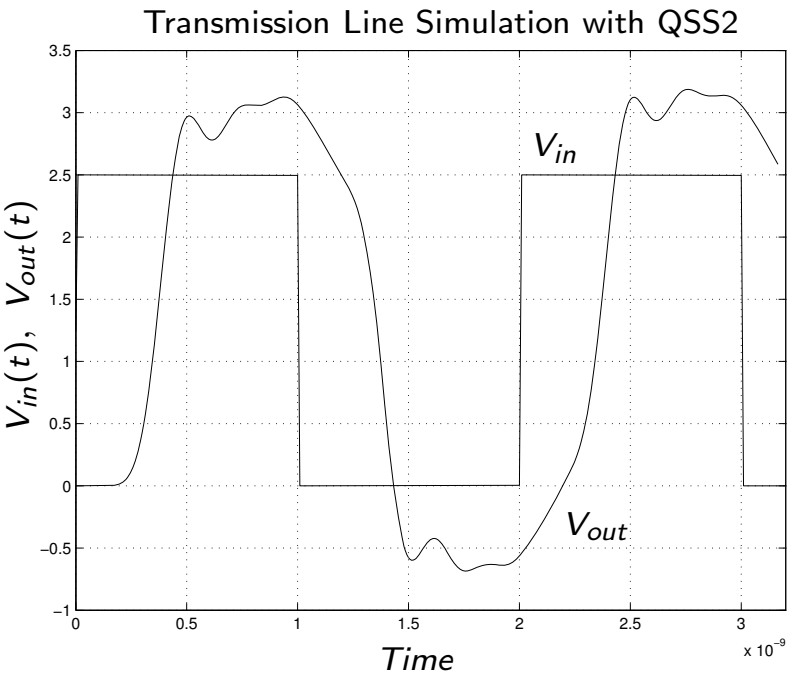


- The simulation took 2536 steps (between 198 and 319 steps for *each integrator*).
- Although the number of steps seems to be quite large, every single step only leads to *scalar calculations* in two or three integrators due to the *sparsity* of the **A**-matrix.
- As the input signal is *piecewise linear*, we can obtain a theoretical upper bound of the *global error* for the entire simulation. For the variable V_{out}, the error bound is $250mV$. That bound is *conservative*.

An Illustrative Example II

Using **PowerDEVS**, we constructed the *block diagram* and simulated the circuit using QSS2 with a *trapezoidal input* (the typical form of signals in integrated digital circuits). We used a quantum of $4mV$ for the potentials and $10\mu A$ for the currents.

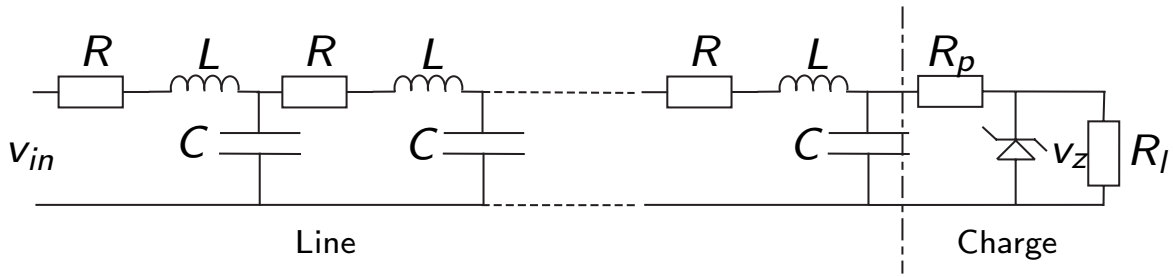
The simulation results were:



- The simulation took 2536 steps (between 198 and 319 steps for *each integrator*).
- Although the number of steps seems to be quite large, every single step only leads to *scalar calculations* in two or three integrators due to the *sparsity* of the **A**-matrix.
- As the input signal is *piecewise linear*, we can obtain a theoretical upper bound of the *global error* for the entire simulation. For the variable V_{out}, the error bound is $250mV$. That bound is *conservative*.
- Reducing the quantum 100 times, we estimate that the *number of steps* will be augmented approximately 10 times. The new error bound for V_{out} is then $2.5mV$.

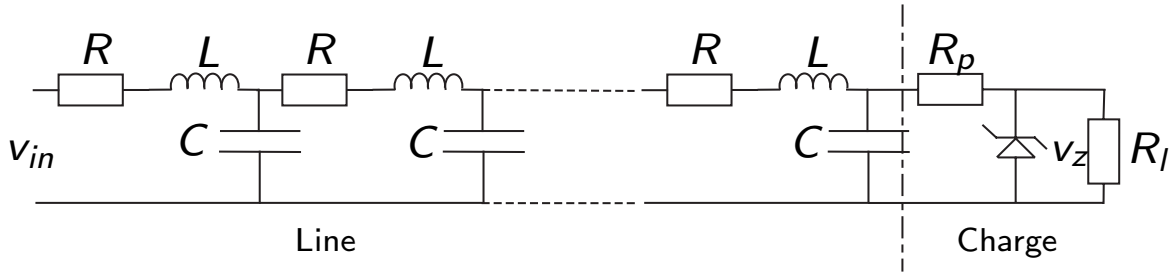
DAE Simulation with QSS Methods

The figure below shows the same circuit as the previous example with added *surge voltage protection* formed by a resistor and a *Zener diode*.



DAE Simulation with QSS Methods

The figure below shows the same circuit as the previous example with added *surge voltage protection* formed by a resistor and a *Zener diode*.

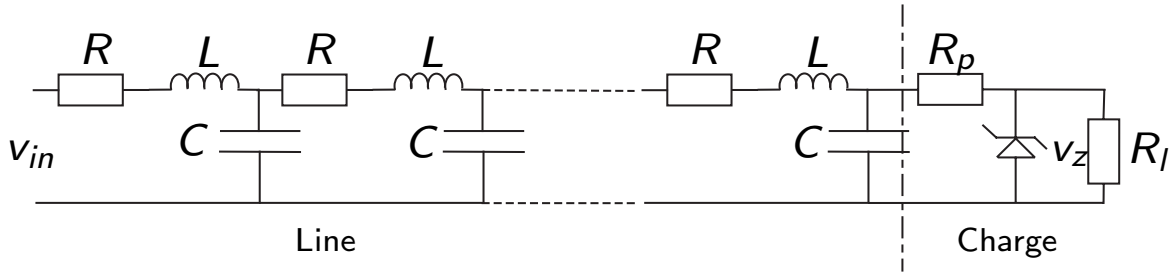


The new system of equations is a *DAE*:

$$\frac{di_1}{dt} = \frac{1}{L} \cdot v_{in} - \frac{R}{L} \cdot i_1 - \frac{1}{L} \cdot u_1$$
$$\frac{du_1}{dt} = \frac{1}{C} \cdot i_1 - \frac{1}{C} \cdot i_2$$
$$\vdots$$
$$\frac{di_5}{dt} = \frac{1}{L} \cdot u_4 - \frac{R}{L} \cdot i_5 - \frac{1}{L} \cdot u_5$$
$$\frac{du_5}{dt} = \frac{1}{C} \cdot i_5 - \frac{1}{R_p C} \cdot (u_5 - v_z)$$

DAE Simulation with QSS Methods

The figure below shows the same circuit as the previous example with added *surge voltage protection* formed by a resistor and a *Zener diode*.



The new system of equations is a *DAE*:

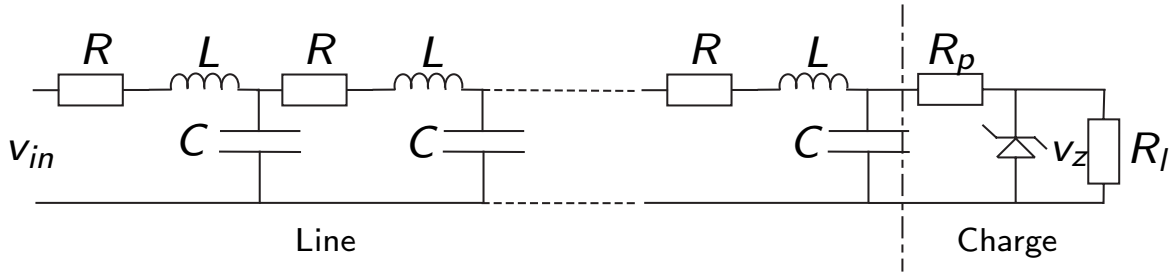
$$\frac{di_1}{dt} = \frac{1}{L} \cdot v_{in} - \frac{R}{L} \cdot i_1 - \frac{1}{L} \cdot u_1$$
$$\frac{du_1}{dt} = \frac{1}{C} \cdot i_1 - \frac{1}{C} \cdot i_2$$
$$\vdots$$
$$\frac{di_5}{dt} = \frac{1}{L} \cdot u_4 - \frac{R}{L} \cdot i_5 - \frac{1}{L} \cdot u_5$$
$$\frac{du_5}{dt} = \frac{1}{C} \cdot i_5 - \frac{1}{R_p C} \cdot (u_5 - v_z)$$

where v_z is an *algebraic variable* that satisfies the equation:

$$\frac{1}{R_p} \cdot u_5 - \left(\frac{1}{R_p} + \frac{1}{R_l} \right) \cdot v_z - \frac{I_0}{1 - (v_z / v_{br})^m} = 0$$

DAE Simulation with QSS Methods

The figure below shows the same circuit as the previous example with added *surge voltage protection* formed by a resistor and a *Zener diode*.



The new system of equations is a *DAE*:

$$\frac{di_1}{dt} = \frac{1}{L} \cdot v_{in} - \frac{R}{L} \cdot i_1 - \frac{1}{L} \cdot u_1$$
$$\frac{du_1}{dt} = \frac{1}{C} \cdot i_1 - \frac{1}{C} \cdot i_2$$
$$\vdots$$
$$\frac{di_5}{dt} = \frac{1}{L} \cdot u_4 - \frac{R}{L} \cdot i_5 - \frac{1}{L} \cdot u_5$$
$$\frac{du_5}{dt} = \frac{1}{C} \cdot i_5 - \frac{1}{R_p C} \cdot (u_5 - v_z)$$

where v_z is an *algebraic variable* that satisfies the equation:

$$\frac{1}{R_p} \cdot u_5 - \left(\frac{1}{R_p} + \frac{1}{R_l} \right) \cdot v_z - \frac{I_0}{1 - (v_z / v_{br})^m} = 0$$

and m, v_{br}, and I_0 are parameters, the values of which depend on the physical characteristics of the Zener diode.

DAE Simulation with QSS Methods II

- To use QSS or QSS2 in the simulation of the circuit, we just replace i_j and u_j by q_{i_j} and q_{u_j}. The problem is the calculation of v_z.

DAE Simulation with QSS Methods II

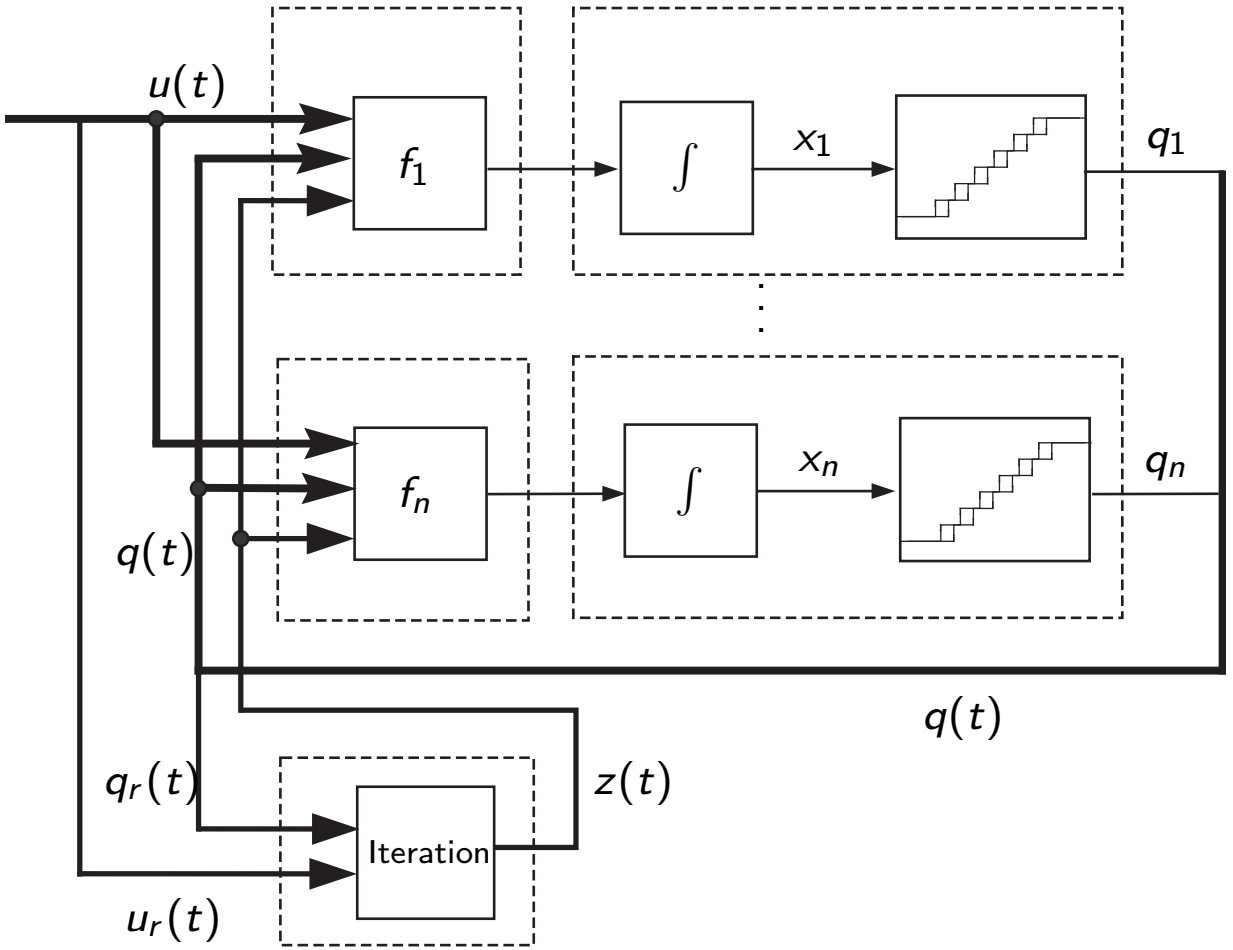
- To use QSS or QSS2 in the simulation of the circuit, we just replace i_j and u_j by q_{i_j} and q_{u_j}. The problem is the calculation of v_z.
- We can use a new DEVS models that computes v_z from q_{u_5}, *iterating* on the algebraic restriction each time this *quantized variable* changes its value.

DAE Simulation with QSS Methods III

The *general scheme* to simulate DAEs with QSS methods is the following:

DAE Simulation with QSS Methods III

The *general scheme* to simulate DAEs with QSS methods is the following:



$$\begin{aligned} \dot{\mathbf{x}} &= \mathbf{f}(\mathbf{q}, \mathbf{u}, \mathbf{z}) \\ 0 &= \mathbf{g}(\mathbf{q_r}, \mathbf{u_r}, \mathbf{z}) \end{aligned}$$

DAE Simulation with QSS Methods IV

- In *index 1 DAEs*, QSS methods only call for *iterations* in those steps involving changes in the quantized variables that form part of the algebraic loops.

DAE Simulation with QSS Methods IV

- In *index 1 DAEs*, QSS methods only call for *iterations* in those steps involving changes in the quantized variables that form part of the algebraic loops.
- This can be an important advantage. In the *transmission line* example, the simulation using the same settings as before took 2640 steps (between 200 and 316 at each integrator). However, there were only 200 steps (corresponding to q_{u_5}) that provoked *iterations*. This added an *almost negligible computational cost* with respect to the previous example.

DAE Simulation with QSS Methods IV

- In *index 1 DAEs*, QSS methods only call for *iterations* in those steps involving changes in the quantized variables that form part of the algebraic loops.
- This can be an important advantage. In the *transmission line* example, the simulation using the same settings as before took 2640 steps (between 200 and 316 at each integrator). However, there were only 200 steps (corresponding to q_{u_5}) that provoked *iterations*. This added an *almost negligible computational cost* with respect to the previous example.
- **PowerDEVS** offers a block that solves a generic *algebraic restriction* $g(\mathbf{q_r}, \mathbf{u_r}, z) = 0$, with z being a scalar variable and $g()$ being a scalar function.

DAE Simulation with QSS Methods IV

- In *index 1 DAEs*, QSS methods only call for *iterations* in those steps involving changes in the quantized variables that form part of the algebraic loops.
- This can be an important advantage. In the *transmission line* example, the simulation using the same settings as before took 2640 steps (between 200 and 316 at each integrator). However, there were only 200 steps (corresponding to q_{u_5}) that provoked *iterations*. This added an *almost negligible computational cost* with respect to the previous example.
- **PowerDEVS** offers a block that solves a generic *algebraic restriction* $g(\mathbf{q_r}, \mathbf{u_r}, z) = 0$, with z being a scalar variable and $g()$ being a scalar function.
- The iteration uses the *secant method*, a variant of the *regula falsi*.

Discontinuity Handling

We saw that the *simulation of discontinuous systems* requires performing a step at the exact moment of the discontinuity. Thus:

Discontinuity Handling

We saw that the *simulation of discontinuous systems* requires performing a step at the exact moment of the discontinuity. Thus:

- in the presence of *time events*, we needed to adjust the step size so that it coincides with those event instants, and,

Discontinuity Handling

We saw that the *simulation of discontinuous systems* requires performing a step at the exact moment of the discontinuity. Thus:

- in the presence of *time events*, we needed to adjust the step size so that it coincides with those event instants, and,
- in the presence of *state events*, we had to *iterate* to detect *when* discontinuities took place.

Discontinuity Handling

We saw that the *simulation of discontinuous systems* requires performing a step at the exact moment of the discontinuity. Thus:

- in the presence of *time events*, we needed to adjust the step size so that it coincides with those event instants, and,
- in the presence of *state events*, we had to *iterate* to detect *when* discontinuities took place.

With QSS methods these problems will disappear for the following reasons.

Discontinuity Handling

We saw that the *simulation of discontinuous systems* requires performing a step at the exact moment of the discontinuity. Thus:

- in the presence of *time events*, we needed to adjust the step size so that it coincides with those event instants, and,
- in the presence of *state events*, we had to *iterate* to detect *when* discontinuities took place.

With QSS methods these problems will disappear for the following reasons.

- All blocks are prepared to receive external events in an *asynchronous* way. Thus, when a *time event* occurs, the input signal that undergoes a discontinuity simply propagates the event to all blocks it is connected to.

Discontinuity Handling

We saw that the *simulation of discontinuous systems* requires performing a step at the exact moment of the discontinuity. Thus:

- in the presence of *time events*, we needed to adjust the step size so that it coincides with those event instants, and,
- in the presence of *state events*, we had to *iterate* to detect *when* discontinuities took place.

With QSS methods these problems will disappear for the following reasons.

- All blocks are prepared to receive external events in an *asynchronous* way. Thus, when a *time event* occurs, the input signal that undergoes a discontinuity simply propagates the event to all blocks it is connected to.
- In the case of a *state event*, its detection is trivial, since the trajectories are piecewise constant (QSS), linear (QSS2), or parabolic (QSS3). Thus, the event time can be computed explicitly, and *iterations* are not needed to detect the *zero crossings*.

Discontinuity Handling

We saw that the *simulation of discontinuous systems* requires performing a step at the exact moment of the discontinuity. Thus:

- in the presence of *time events*, we needed to adjust the step size so that it coincides with those event instants, and,
- in the presence of *state events*, we had to *iterate* to detect *when* discontinuities took place.

With QSS methods these problems will disappear for the following reasons.

- All blocks are prepared to receive external events in an *asynchronous* way. Thus, when a *time event* occurs, the input signal that undergoes a discontinuity simply propagates the event to all blocks it is connected to.
- In the case of a *state event*, its detection is trivial, since the trajectories are piecewise constant (QSS), linear (QSS2), or parabolic (QSS3). Thus, the event time can be computed explicitly, and *iterations* are not needed to detect the *zero crossings*.

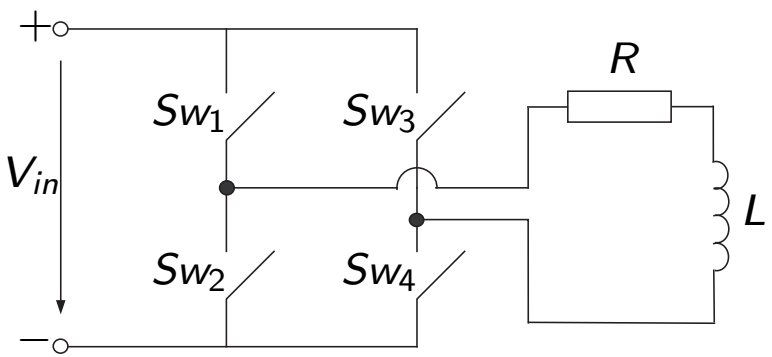
For these reasons, the *main advantage* of the QSS methods lies in *simulations of systems with discontinuities*.

An Introductory Example

The figure below shows an *inverter circuit*:

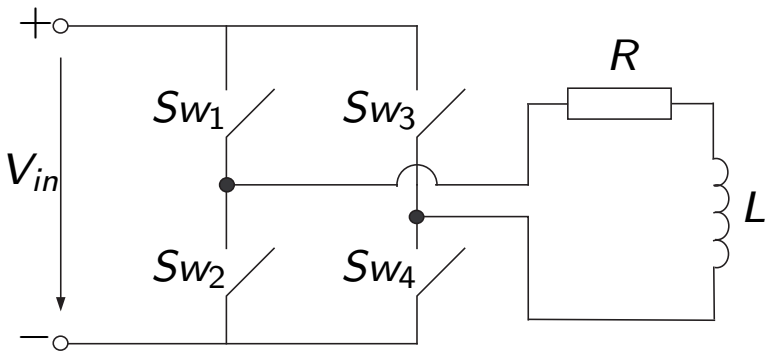
An Introductory Example

The figure below shows an *inverter circuit*:



An Introductory Example

The figure below shows an *inverter circuit*:

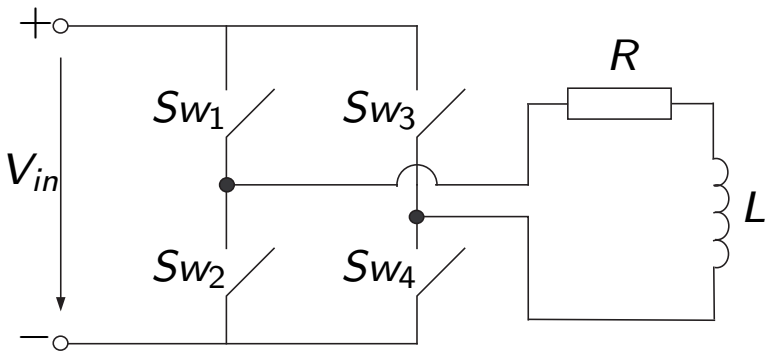


which can be modeled with the ODE:

$$\frac{di_L}{dt} = -\frac{R}{L} \cdot i_L + \frac{V_{in}}{L} \cdot s_w(t)$$

An Introductory Example

The figure below shows an *inverter circuit*:



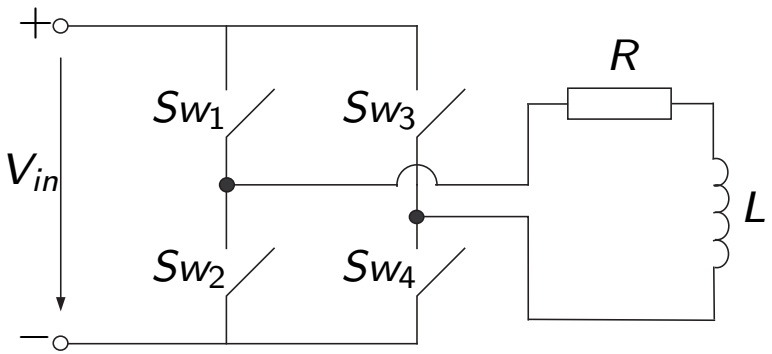
which can be modeled with the ODE:

$$\frac{di_L}{dt} = -\frac{R}{L} \cdot i_L + \frac{V_{in}}{L} \cdot s_w(t)$$

where $s_w(t)$ takes the values ± 1 depending on the switch positions.

An Introductory Example

The figure below shows an *inverter circuit*:



which can be modeled with the ODE:

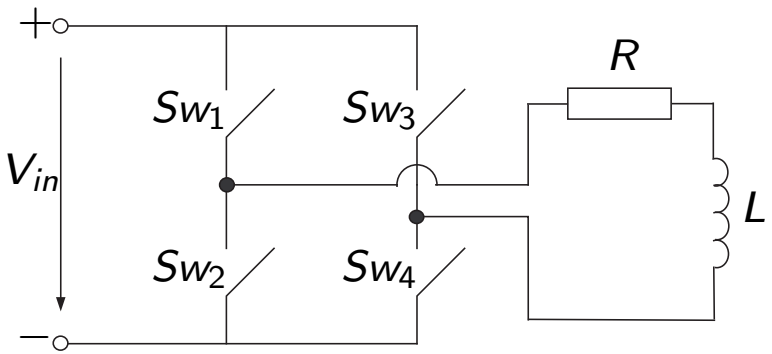
$$\frac{di_L}{dt} = -\frac{R}{L} \cdot i_L + \frac{V_{in}}{L} \cdot s_w(t)$$

where $s_w(t)$ takes the values ± 1 depending on the switch positions.

A typical way of *controlling* the switches in order to obtain an approximately *sinusoidal* current at the load is by using a *pulse width modulation* (PWM) strategy:

An Introductory Example

The figure below shows an *inverter circuit*:

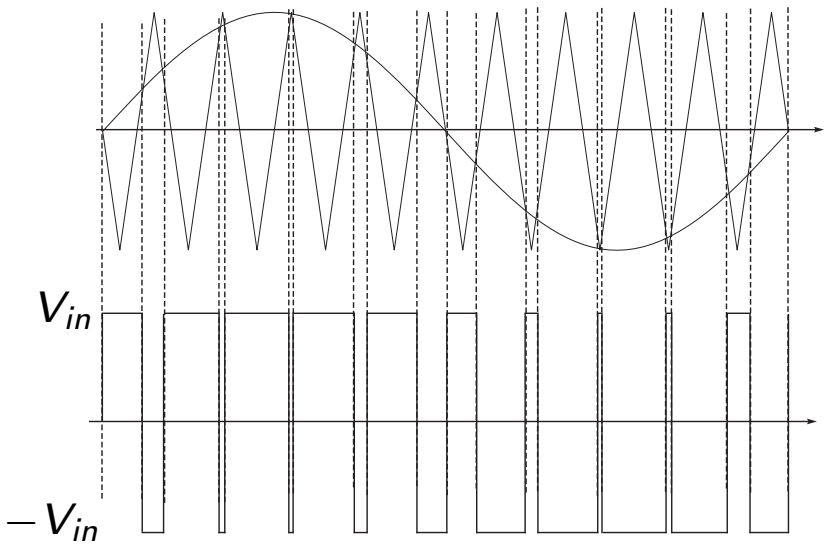


which can be modeled with the ODE:

$$\frac{di_L}{dt} = -\frac{R}{L} \cdot i_L + \frac{V_{in}}{L} \cdot s_w(t)$$

where $s_w(t)$ takes the values ± 1 depending on the switch positions.

A typical way of *controlling* the switches in order to obtain an approximately *sinusoidal* current at the load is by using a *pulse width modulation* (PWM) strategy:



An Introductory Example: Time Events

In order to simulate the previous system using any of the QSS$_i$ algorithms, we need a DEVS model that *generates the sequence of events* corresponding to $s_w(t)$. This DEVS model can be easily obtained.

An Introductory Example: Time Events

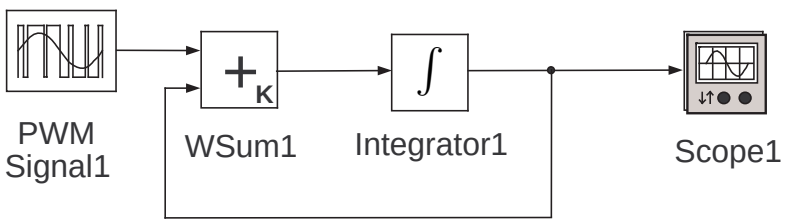
In order to simulate the previous system using any of the QSS$_i$ algorithms, we need a DEVS model that *generates the sequence of events* corresponding to $s_w(t)$. This DEVS model can be easily obtained.

The model coded in **PowerDEVS** looks as follows:

An Introductory Example: Time Events

In order to simulate the previous system using any of the QSS$_i$ algorithms, we need a DEVS model that *generates the sequence of events* corresponding to $s_w(t)$. This DEVS model can be easily obtained.

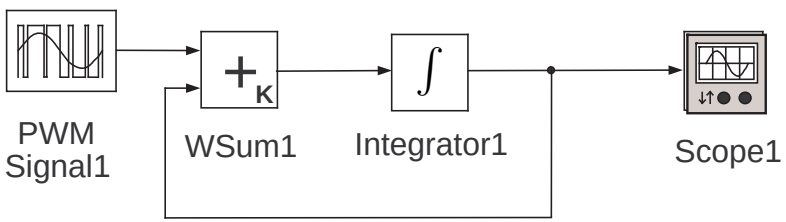
The model coded in **PowerDEVS** looks as follows:



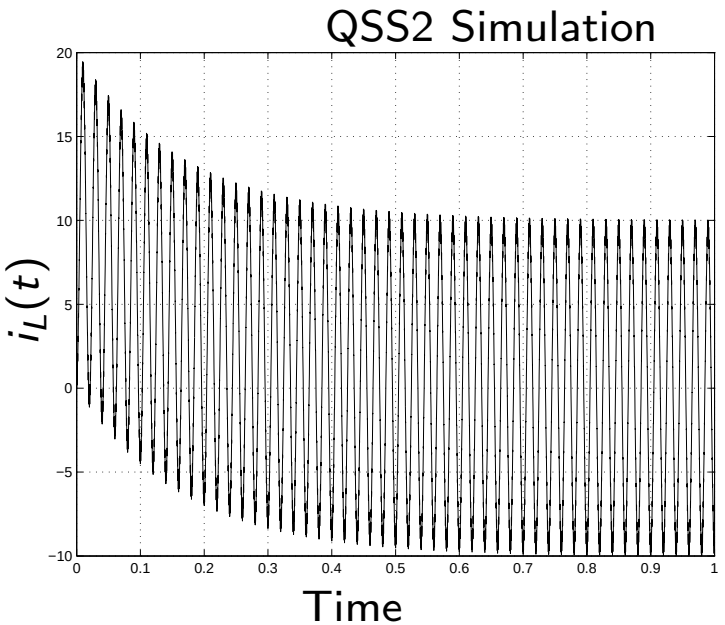
An Introductory Example: Time Events

In order to simulate the previous system using any of the QSS$_i$ algorithms, we need a DEVS model that *generates the sequence of events* corresponding to $s_w(t)$. This DEVS model can be easily obtained.

The model coded in **PowerDEVS** looks as follows:



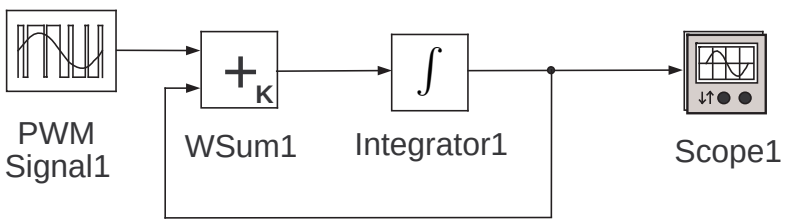
and the simulation results are:



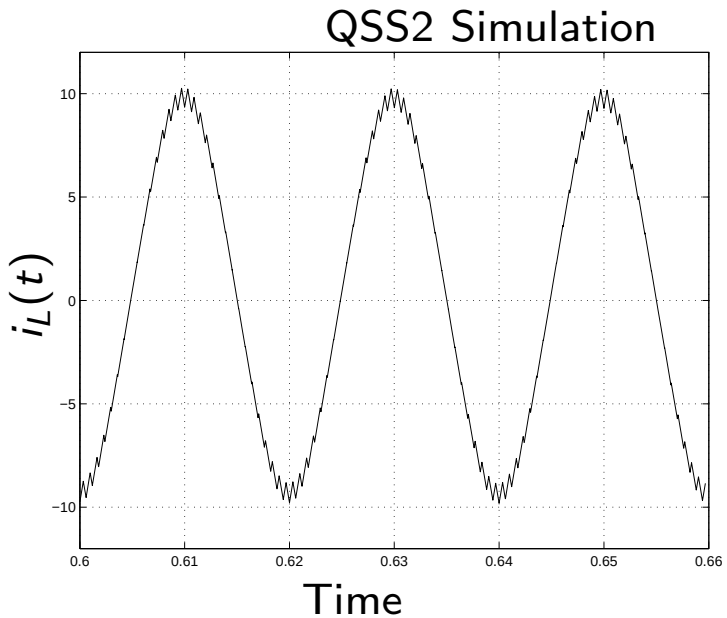
An Introductory Example: Time Events

In order to simulate the previous system using QSS, QSS2, or QSS3, we need a DEVS model that *generates the sequence of events* corresponding to $s_w(t)$. This DEVS model can be easily obtained.

The model coded in **PowerDEVS** looks as follows:



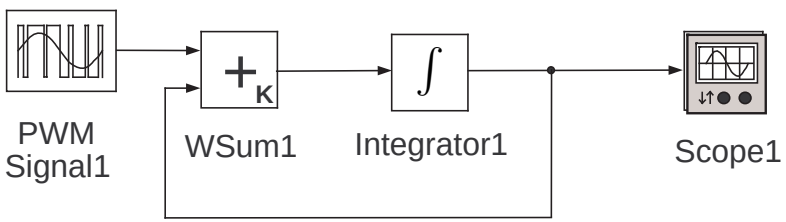
and the simulation results are:



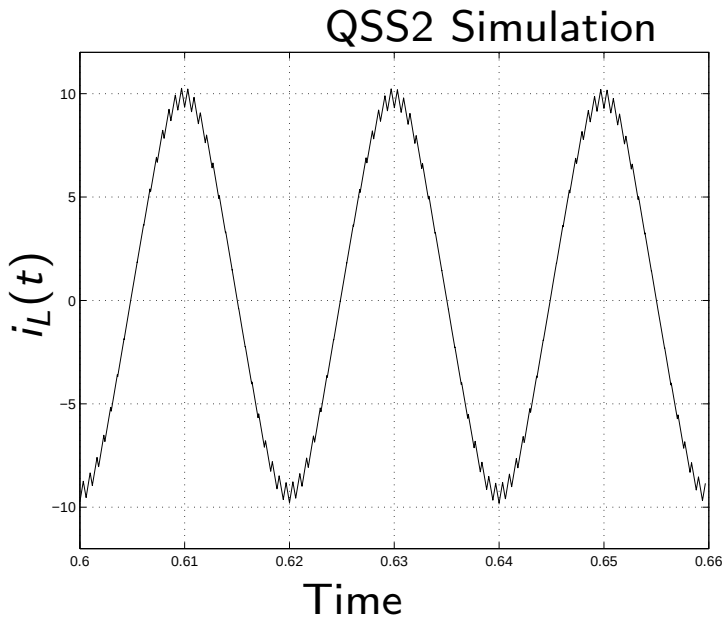
An Introductory Example: Time Events

In order to simulate the previous system using QSS, QSS2, or QSS3, we need a DEVS model that *generates the sequence of events* corresponding to $s_w(t)$. This DEVS model can be easily obtained.

The model coded in **PowerDEVS** looks as follows:



and the simulation results are:



The *time events* are treated as a conventional *inputs*. In this case, the *global error bound* holds. The error is bounded by the quantum $\Delta i_L = 0.01\ A$.

An Introductory Example: State Events

A simple way to protect the previous circuit against voltage surges during transients or faults is by adding a device that *measures* the load current and closes all four switches, enforcing $s_w = 0$, when the current surpasses its allowed maximum value.

An Introductory Example: State Events

A simple way to protect the previous circuit against voltage surges during transients or faults is by adding a device that *measures* the load current and closes all four switches, enforcing $s_w = 0$, when the current surpasses its allowed maximum value.

With the addition of this device, the system equations becomes:

$$\frac{di_L}{dt} = -\frac{R}{L} \cdot i_L + V_{in} \cdot \tilde{s}_w(t)$$

An Introductory Example: State Events

A simple way to protect the previous circuit against voltage surges during transients or faults is by adding a device that *measures* the load current and closes all four switches, enforcing $s_w = 0$, when the current surpasses its allowed maximum value.

With the addition of this device, the system equations becomes:

$$\frac{di_L}{dt} = -\frac{R}{L} \cdot i_L + V_{in} \cdot \tilde{s}_w(t)$$

where:

$$\tilde{s}_w(t) = \begin{cases} s_w(t) & \text{if } i_L(t) < i_M \\ 0 & \text{otherwise} \end{cases}$$

and i_M is the *largest allowed current*.

An Introductory Example: State Events

A simple way to protect the previous circuit against voltage surges during transients or faults is by adding a device that *measures* the load current and closes all four switches, enforcing $s_w = 0$, when the current surpasses its allowed maximum value.

With the addition of this device, the system equations becomes:

$$\frac{di_L}{dt} = -\frac{R}{L} \cdot i_L + V_{in} \cdot \tilde{s}_w(t)$$

where:

$$\tilde{s}_w(t) = \begin{cases} s_w(t) & \text{if } i_L(t) < i_M \\ 0 & \text{otherwise} \end{cases}$$

and i_M is the *largest allowed current*.

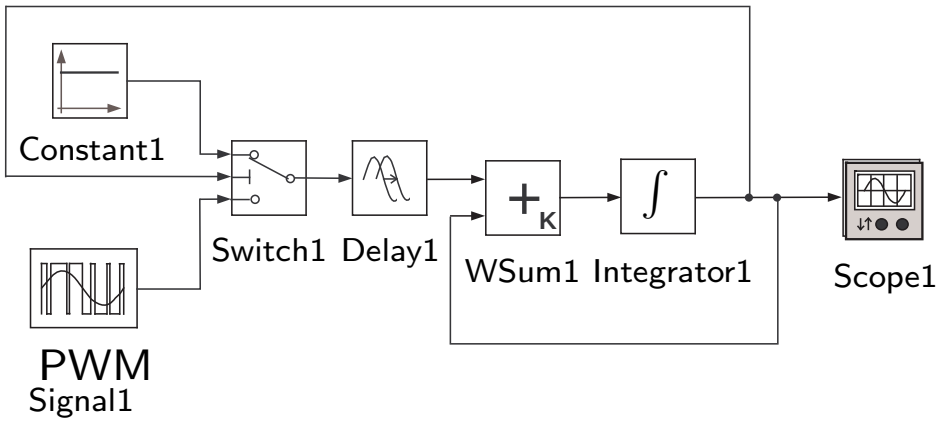
Now we are dealing with a *state event*.

An Introductory Example: State Events II

The corresponding **PowerDEVS** model can be coded as follows:

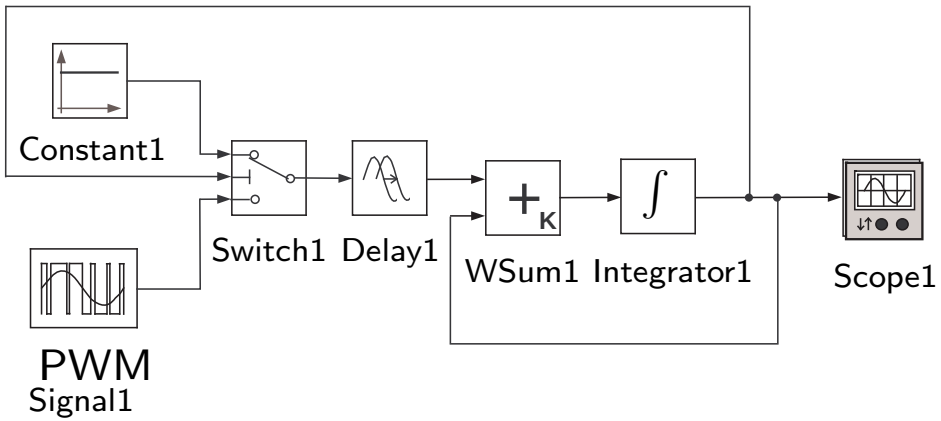
An Introductory Example: State Events II

The corresponding **PowerDEVS** model can be coded as follows:



An Introductory Example: State Events II

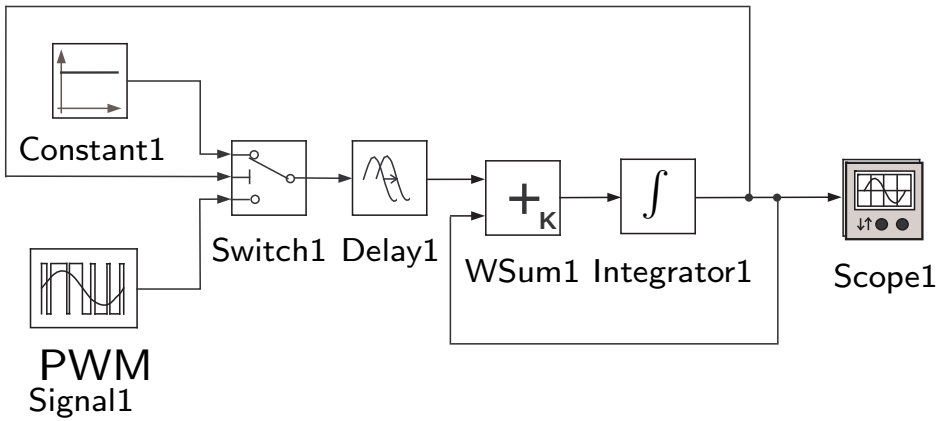
The corresponding **PowerDEVS** model can be coded as follows:



The *Delay* block represents the time required for the commutation. Had we not included the delay, the resulting model would have been *illegitimate*.

An Introductory Example: State Events II

The corresponding **PowerDEVS** model can be coded as follows:

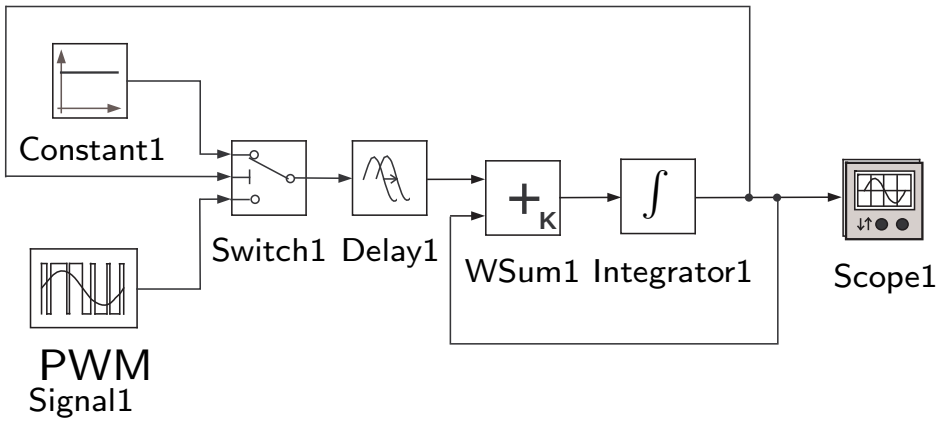


The *Delay* block represents the time required for the commutation. Had we not included the delay, the resulting model would have been *illegitimate*.

The *Switch* block represents the equation with the *if* statement. It works as follows:

An Introductory Example: State Events II

The corresponding **PowerDEVS** model can be coded as follows:



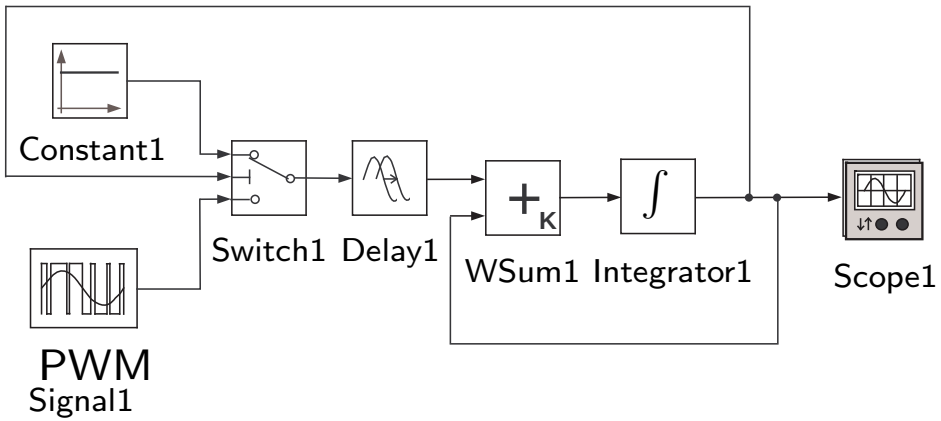
The *Delay* block represents the time required for the commutation. Had we not included the delay, the resulting model would have been *illegitimate*.

The *Switch* block represents the equation with the *if* statement. It works as follows:

- When the block receives an *event* at its center port, it computes when the signal crosses a given threshold. At that time, it *changes* the switch position.

An Introductory Example: State Events II

The corresponding **PowerDEVS** model can be coded as follows:



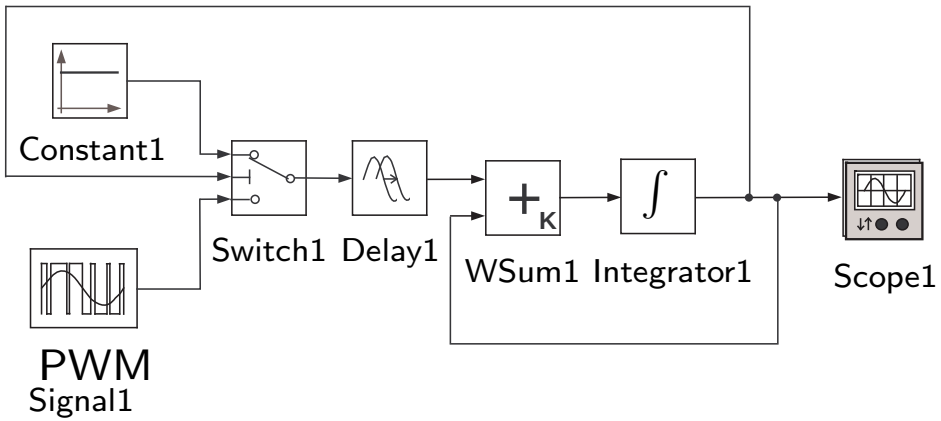
The *Delay* block represents the time required for the commutation. Had we not included the delay, the resulting model would have been *illegitimate*.

The *Switch* block represents the equation with the *if* statement. It works as follows:

- When the block receives an *event* at its center port, it computes when the signal crosses a given threshold. At that time, it *changes* the switch position.
- When the block receives events on either of the other two ports, it may pass them on to the output port depending on the switch position.

An Introductory Example: State Events II

The corresponding **PowerDEVS** model can be coded as follows:



The *Delay* block represents the time required for the commutation. Had we not included the delay, the resulting model would have been *illegitimate*.

The *Switch* block represents the equation with the *if* statement. It works as follows:

- When the block receives an *event* at its center port, it computes when the signal crosses a given threshold. At that time, it *changes* the switch position.
- When the block receives events on either of the other two ports, it may pass them on to the output port depending on the switch position.

The calculation of the *commutation time* is trivial, because the signal is either piecewise linear (QSS2) or piecewise parabolic (QSS3).

An Introductory Example: Simulation Results

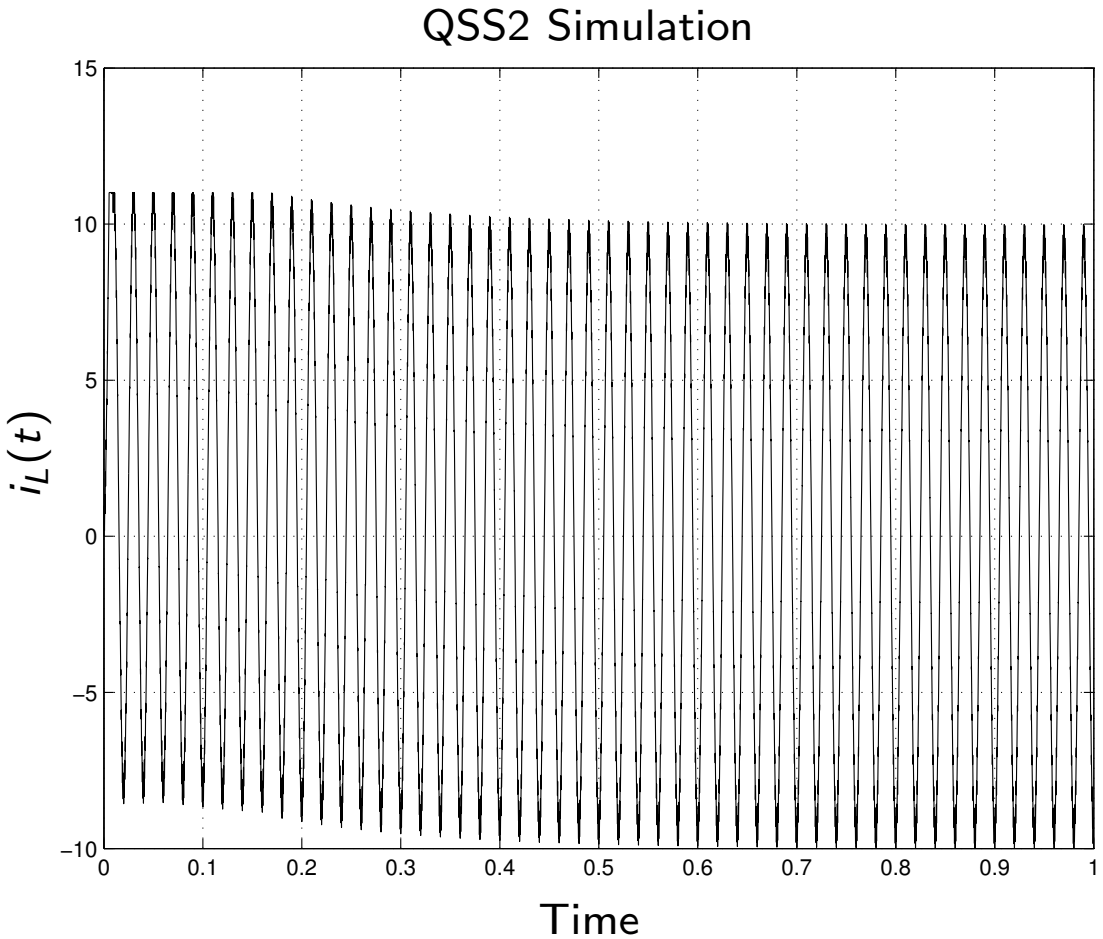


Figure: Load Current with Surge Protection.

An Introductory Example: Simulation Results

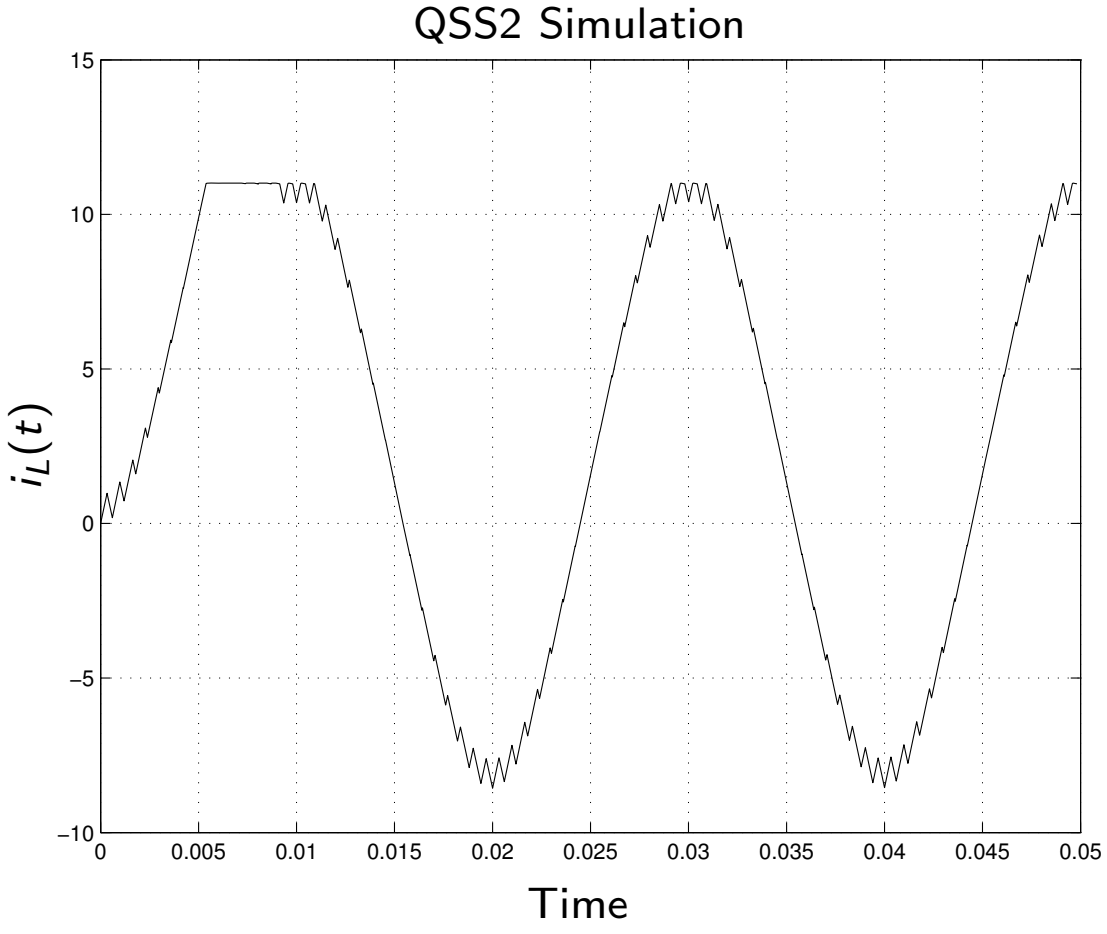


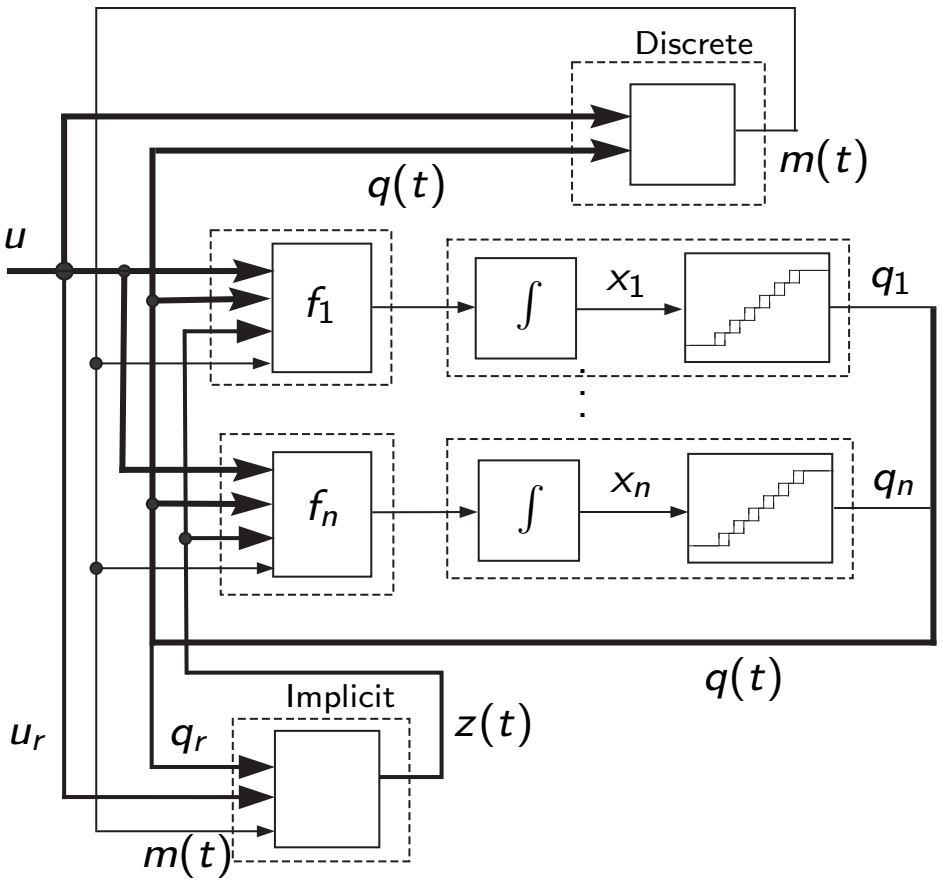
Figure: Load Current with Surge Protection.

Discontinuous Systems: General Scheme

In general, the following scheme can be used to model discontinuous systems:

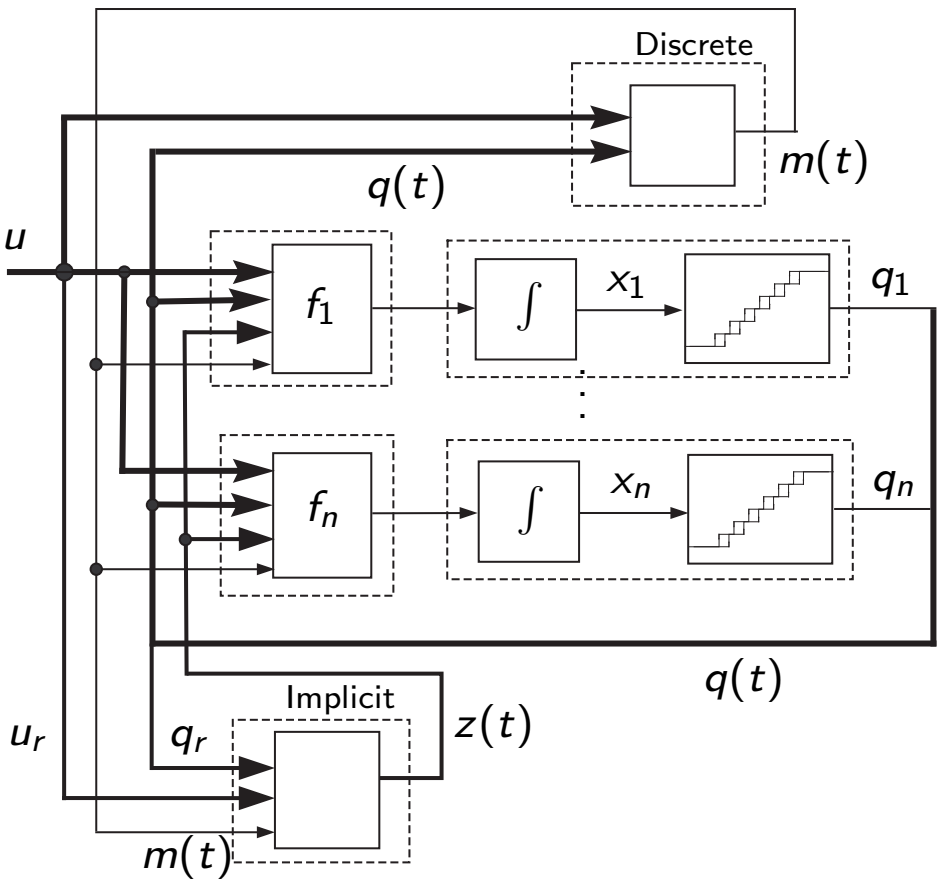
Discontinuous Systems: General Scheme

In general, the following scheme can be used to model discontinuous systems:



Discontinuous Systems: General Scheme

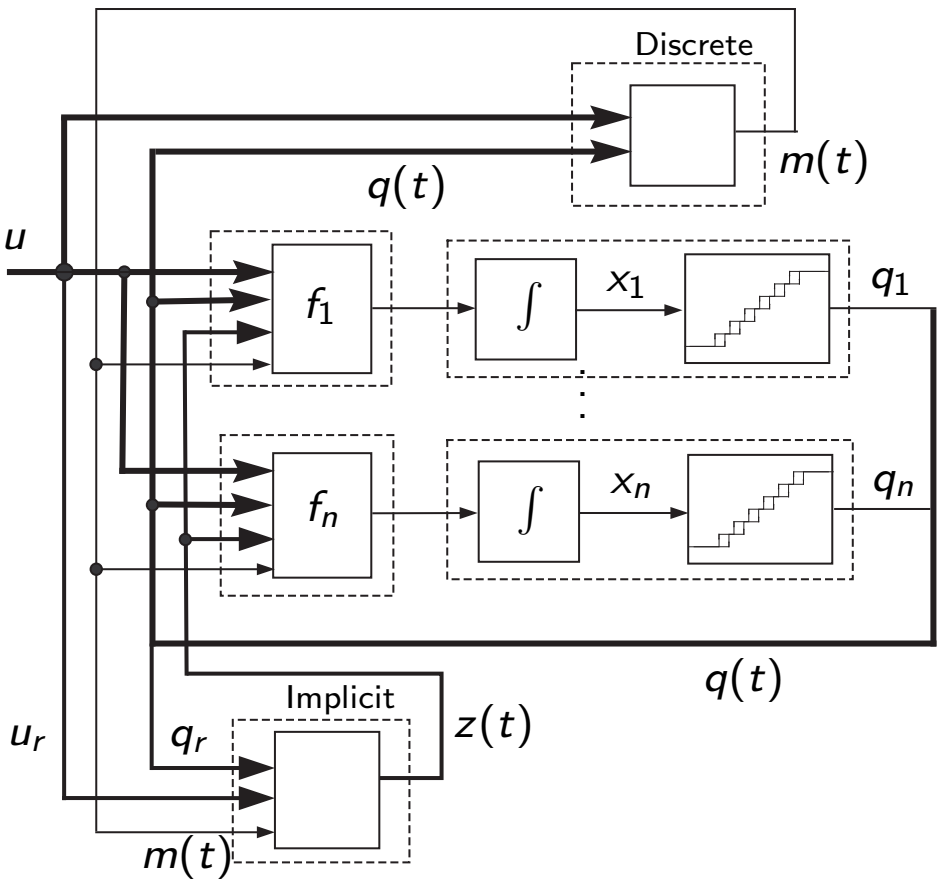
In general, the following scheme can be used to model discontinuous systems:



- The *discrete* subsystem is in charge of *detecting* and *handling* discontinuities, computing a *piecewise constant* signal $m(t)$.

Discontinuous Systems: General Scheme

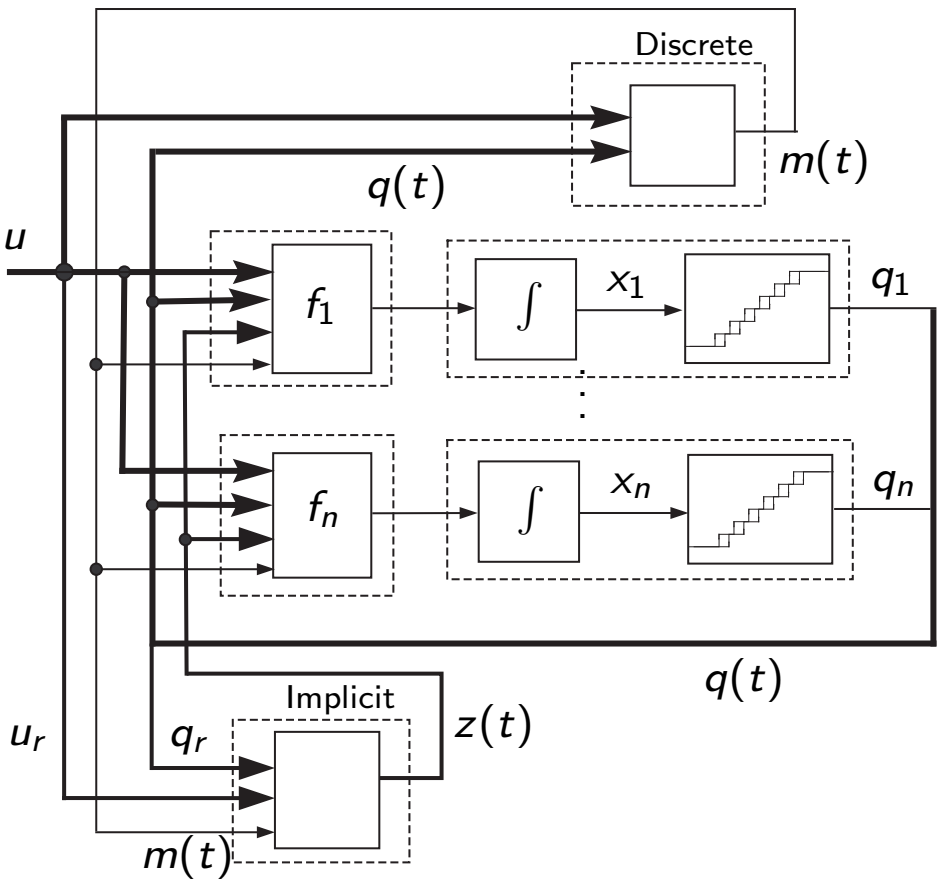
In general, the following scheme can be used to model discontinuous systems:



- The *discrete* subsystem is in charge of *detecting* and *handling* discontinuities, computing a *piecewise constant* signal $m(t)$.
- The remaining blocks *ignore* the presence of the *discontinuities*. The signal $m(t)$ is used as a regular *input signal* by those blocks that need it.

Discontinuous Systems: General Scheme

In general, the following scheme can be used to model discontinuous systems:



- The *discrete* subsystem is in charge of *detecting* and *handling* discontinuities, computing a *piecewise constant* signal $m(t)$.
- The remaining blocks *ignore* the presence of the *discontinuities*. The signal $m(t)$ is used as a regular *input signal* by those blocks that need it.
- To support the computation of the signal $m(t)$, **PowerDEVS** offers a set of different blocks including *switches*, *quantizers*, *comparators*, etc.

Conclusions

- We continued with the discussion of *quantization-based integration methods*. We introduced *second-order accurate methods*. Third- and fourth-order accurate methods can be defined accordingly and are already offered in **PowerDEVS**.

Conclusions

- We continued with the discussion of *quantization-based integration methods*. We introduced *second-order accurate methods*. Third- and fourth-order accurate methods can be defined accordingly and are already offered in **PowerDEVS**.
- We showed that the problem of stability and accuracy can be studied based on *perturbation analysis*.

Conclusions

- We continued with the discussion of *quantization-based integration methods*. We introduced *second-order accurate methods*. Third- and fourth-order accurate methods can be defined accordingly and are already offered in **PowerDEVS**.
- We showed that the problem of stability and accuracy can be studied based on *perturbation analysis*.
- We showed that the **QSS**$_i$ methods remain *practically stable* and that the *global integration error* can be estimated. The latter issue is of big significance, as classical ODE solvers that are based on time slicing only estimate the *local integration error* of a single step.

Conclusions

- We continued with the discussion of *quantization-based integration methods*. We introduced *second-order accurate methods*. Third- and fourth-order accurate methods can be defined accordingly and are already offered in **PowerDEVS**.
- We showed that the problem of stability and accuracy can be studied based on *perturbation analysis*.
- We showed that the **QSS**$_i$ methods remain *practically stable* and that the *global integration error* can be estimated. The latter issue is of big significance, as classical ODE solvers that are based on time slicing only estimate the *local integration error* of a single step.
- **QSS**$_i$ methods are *naturally asynchronous* and can be easily implemented on *parallel architectures*.

Conclusions

- We continued with the discussion of *quantization-based integration methods*. We introduced *second-order accurate methods*. Third- and fourth-order accurate methods can be defined accordingly and are already offered in **PowerDEVS**.
- We showed that the problem of stability and accuracy can be studied based on *perturbation analysis*.
- We showed that the **QSS**$_i$ methods remain *practically stable* and that the *global integration error* can be estimated. The latter issue is of big significance, as classical ODE solvers that are based on time slicing only estimate the *local integration error* of a single step.
- **QSS**$_i$ methods are *naturally asynchronous* and can be easily implemented on *parallel architectures*.
- **QSS**$_i$ methods are particularly well suited for the simulation of *hybrid systems*.

References

1. Cellier, F.E., E. Kofman, G. Migoni, and M. Bortolotto (2008), "Quantized State System Simulation," *Proc. GCMS'08, Grand Challenges in Modeling and Simulation*, part of *SCSC'08, Summer Computer Simulation Conference*, Edinburgh, Scotland, pp. 504-510.

2. Sanz, V., A. Urquía, S. Dormido, and F.E. Cellier (2010), "System Modeling Using the Parallel DEVS Formalism and the Modelica Language," *Simulation Modelling Practice and Theory*, **18**(7), pp.998-1018.

3. Sanz, V., F.E. Cellier, A. Urquía, and S. Dormido (2009), "Modeling of the ARGESIM 'Crane and Embedded Controller' System Using the DEVSLib Modelica Library," *Proc. ADHS'09: 3rd IFAC Conference on Analysis and Design of Hybrid Systems*, Zaragoza, Spain.

4. Beltrame, T. and F.E. Cellier (2006), "Quantized State System Simulation in Dymola/Modelica Using the DEVS Formalism," *Simulation News Europe*, **16**(3), pp.3-12.

5. Beltrame, Tamara (2006), *Design and Development of a Dymola/Modelica Library for Discrete Event-oriented Systems Using DEVS Methodology*, MS Thesis, Dept. of Computer Science, ETH Zurich, Switzerland.

References II

1. Floros, X., F.E. Cellier, and E. Kofman (2010), "Discretizing Time or States? A Comparative Study between DASSL and QSS," *Proc. 3rd International Workshop on Equation-based Object-oriented Modeling Languages and Tools*, Oslo, Norway, pp.107-115.

2. Floros, X., F. Bergero, F.E. Cellier, and E. Kofman (2011), "Automated Simulation of Modelica Models with QSS Methods – The Discontinuous Case," *Proc. 8th International Modelica Conference*, Dresden, Germany.