12

Quantization–based Integration

Preview

This chapter focuses on the *Quantized State Systems (QSS)* method and its extensions. After a brief explanation concerning the connections between this discrete event method and perturbation theory, the main theoretical properties of the method, i.e., convergence, stability, and error control properties, are presented.

The reader is then introduced to some practical aspects of the method related to the choice of quantum and hysteresis, the incorporation of input signals, as well as output interpolation.

In spite of the theoretical and practical advantages that the QSS method offers, the method has a serious drawback, as it is only first–order accurate. For this reason, a second–order accurate quantization–based method is subsequently presented that conserves the main theoretical properties that characterize the QSS method.

Further, we shall focus on the use of both quantization–based methods in the simulation of DAEs and discontinuous systems, where we shall observe some interesting advantages that these methods have over the classical discrete–time methods.

Finally, and following the discussion of a real–time implementation of these methods, some drawbacks and open problems of the proposed methodology shall be discussed with particular emphasis given to the simulation of stiff system.

12.1 Introduction

In Chapter 2, we introduced two basic properties of numerical methods: the *approximation accuracy* and the *numerical stability*. If we want to rely on the simulation results generated by a method, we must know something about these properties in the context of the application at hand.

The conventional tools for the analysis of numerical stability are based on the discrete–time systems theory. The basic idea is to obtain the difference equations corresponding to a given method applied to a linear time–invariant autonomous system, and then to relate the eigenvalues of the **F**–matrix of the transformed discrete–time system to those of the **A**–matrix of the original continuous–time system.

This technique, that we have applied throughout this book to the analysis

of discrete–time methods, cannot be extended to the QSS method, because the resulting simulation model is a discrete event system that does not possess an **F**–matrix.

Since linear stability theory is such a convenient tool, we might be inclined to attempt tackling this problem by looking for a discrete event systems theory that would support this kind of stability analysis. In fact, there exists a nice mathematical theory based on the use of *max–plus algebra* that permits expressing some discrete event systems through difference equations in the context of that algebra [12.1]. This theory also arrives at stability results based on the study of eigenvalues and is completely analogous to the discrete–time systems theory. However, it can only be applied to systems described by Petri nets, and unfortunately, the QSS method produces DEVS models that do not have Petri net equivalents.

A different approach to studying the QSS dynamics might be to compare directly the results obtained when simulating the original continuous–time model of Eq.(11.8) with those obtained when simulating its QSS approximation of Eq.(11.9).

Let us rewrite these two representations using vector notation. The original continuous system may be written as follows:

$$\dot{\mathbf{x}}_{\mathbf{a}}(t) = \mathbf{f}(\mathbf{x}_{\mathbf{a}}(t), \mathbf{u}(t)) \tag{12.1}$$

and the resulting quantized state system can be written as:

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{q}(t), \mathbf{u}(t)) \tag{12.2}$$

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

Let us define $\mathbf{\Delta x}(t) = \mathbf{q}(t) - \mathbf{x}(t)$. Then, Eq.(12.2) can be rewritten as:

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t) + \mathbf{\Delta x}(t), \mathbf{u}(t)) \tag{12.3}$$

and now, the simulation model of Eq.(12.2) can be interpreted as a *perturbed representation* of the original system of Eq.(12.1).

Hysteretic quantization functions have a fundamental property. If two variables $q_i(t)$ and $x_i(t)$ are related by a hysteretic quantization function, such as that of Eq.(11.10), then:

$$Q_0 < x_i(t) < Q_r \Rightarrow |q_i(t) - x_i(t)| \leq \max(\Delta Q, \varepsilon) \tag{12.4}$$

where $\Delta Q = \max(Q_{j+1} - Q_j),\ 0 \leq j \leq r-1$, is the largest quantum.

The property given by Eq.(12.4) implies that each component of the perturbation $\mathbf{\Delta x}$ is bounded by the corresponding hysteresis width and quantum size. Thus, the accuracy and stability analysis can be based on the effects of a bounded perturbation.

In Chapter 2, we mentioned that there exists a theory that allows studying the numerical stability of nonlinear systems. We shunned away from

pursuing that theory any further, because the mathematical apparatus required to do so is quite formidable.

Unfortunately, we now have no choice but to go down that route, because the QSS representation even of a linear system is in fact nonlinear. Luckily, we shall see that many of the problems associated with general contractivity theory disappear in the special case of a QSS, because we can reduce the nonlinearity of the quantization to the special case of a linear perturbation, and perturbation analysis can be applied even to nonlinear systems quite easily. Furthermore, when the QSS method is applied to a linear system, it will also be possible to establish a global error bound, and the problem of approximation accuracy can then be dealt with not only locally, but even globally.

Despite these advantages, a new problem appears in the QSS method. Let us illustrate this problem by means of the following example. Consider the first–order system:

$$\dot{x}_a(t) = -x_a(t) + 9.5 \tag{12.5}$$

with the initial condition $x_a(0) = 0$.

The results of a simulation using the QSS method with a quantum of $\Delta Q = 1$ and a hysteresis width of $\varepsilon = 1$ are shown in Fig.12.1

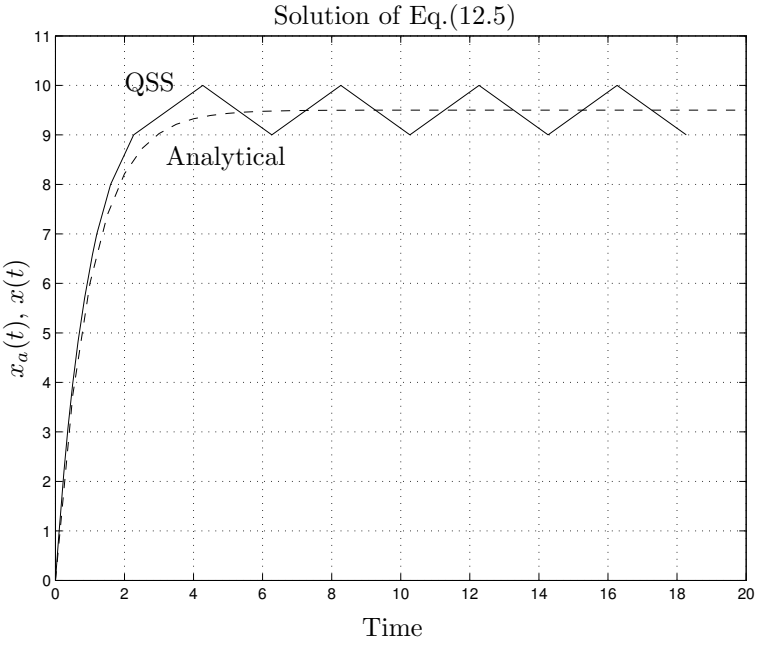

FIGURE 12.1. QSS simulation of Eq.(12.5).

Although the system of Eq.(12.5) is asymptotically stable, the QSS sim-

ulation ends in a limit cycle. The equilibrium point $\bar{x} = 9.5$ is no longer a stable equilibrium point in the resulting QSS, and we cannot claim that stability, in the sense of Lyapunov, is conserved by the method.

However, the QSS solution never deviates far from the exact solution, and it finishes with an oscillation around the equilibrium point. Taking into account that our goal was to just simulate the system and obtain some meaningful trajectories, this result is not bad.

The trajectory found by the QSS method is called *ultimately bounded* [12.8]. In general, the quantization based methods cannot ensure stability in accordance with the classical definition. Hence we shall talk, in the context of QSS simulations, about stability in terms of ultimate boundedness of the solutions obtained.

12.2 Convergence, Accuracy, and Stability in QSS

When a time–invariant system, such as that of Eq.(12.1), is simulated using the QSS method, we obtain an exact simulation, ignoring the roundoff problems, of the perturbed system of Eq.(12.3). Then, as it was mentioned above, the theoretical properties can be studied based on the effects of perturbation.

The first property proven in the context of the QSS method was that of *convergence* [12.10]. The analysis shows that the solutions of Eq.(12.3) approach those of Eq.(12.1) when the largest quantum, ΔQ, and the hysteresis width, ε, are chosen sufficiently small. The importance of this property lies in the fact that an arbitrarily small simulation error can be achieved, when a sufficiently small quantization is being used.

A sufficient condition that ensures that the trajectories of the system of Eq.(12.3) converge to the trajectories of Eq.(12.1) is that the function $\mathbf{f}(\mathbf{x}(t), \mathbf{u}(t))$ is locally Lipschitz[1]. Hence the convergence of the QSS method is a property satisfied by nonlinear systems in general.

Although convergence constitutes an important theoretical property, it does not offer any quantitative information about the relationship between the quantum and the error, and it does not establish any condition for the stability domain.

The stability properties of the QSS method were studied in [12.10] by finding a Lyapunov function for the perturbed system. The analysis shows that, when the system of Eq.(12.1) has an asymptotically stable equilibrium point, for any arbitrarily small region around that equilibrium point, a quantization can be found, so that the solutions of Eq.(12.2) finish inside that region. Moreover, an algorithm can be derived from this analysis that

[1] In a nutshell, this means that the function $\mathbf{f}$ must not escape to infinity within the range of interest.

allows calculating the appropriate quantum.

A sufficient condition for ensuring stability is that the function $\mathbf{f}$ be continuous and continuously differentiable. Hence the stability condition is a bit stronger than the convergence condition.

Thus, the QSS method offers tools, that can be applied to nonlinear systems, for choosing a quantum that ensures that the steady–state simulation error is smaller than a desired bound. Although this result represents an important advantage over the classical discrete–time methods, where stability is usually studied in the context of linear time–invariant (LTI) systems only, the algorithm is quite involved and requires the use of a Lyapunov function of Eq.(12.1) that cannot be easily derived in general cases. Thus, the importance of this stability analysis is, as before, more of a theoretical than a practical nature, and we shall refrain from delving into details about it.

Like in discrete–time methods, the most interesting qualities of QSS come from the analysis of its application to LTI systems. The main result of that analysis, performed in [12.13], states that the error in the QSS simulation of an asymptotically stable LTI system is always bounded. The error bound, which can be calculated from the quantum and some geometrical properties of the system, does not depend either on the initial condition or on the input trajectory and remains constant during the simulation.

Before explaining this fundamental property in more detail, we shall need to introduce some new notation, in order to be able to express the relationships between quanta and error bounds in terms of compact formulae.

We shall use the symbol $|\cdot|$ to denote the componentwise module of a vector or matrix. For instance, if $\mathbf{G}$ is a $j \times k$ matrix with complex components $g_{1,1}, \ldots, g_{j,k}$, then $|\mathbf{G}|$ is also a $j \times k$ matrix with the real positive components $|g_{1,1}|, \ldots, |g_{j,k}|$.

Similarly, we shall use the symbol "$\leq$" to perform a componentwise comparison between real–valued vectors of equal length. Thus, the expression $\mathbf{x} \leq \mathbf{y}$ states that $x_1 \leq y_1, \ldots, x_n \leq y_n$.

With these definitions, let $\mathbf{x_a}(t)$ be a solution of the LTI system:

$$\dot{\mathbf{x}}_\mathbf{a}(t) = \mathbf{A} \cdot \mathbf{x_a}(t) + \mathbf{B} \cdot \mathbf{u}(t) \tag{12.6}$$

Let $\mathbf{x}(t)$ be the solution, starting from the same initial condition, of its associated QSS:

$$\dot{\mathbf{x}}(t) = \mathbf{A} \cdot \mathbf{q}(t) + \mathbf{B} \cdot \mathbf{u}(t) \tag{12.7}$$

which can be written as:

$$\dot{\mathbf{x}}(t) = \mathbf{A} \cdot [\mathbf{x}(t) + \mathbf{\Delta x}(t)] + \mathbf{B} \cdot \mathbf{u}(t) \tag{12.8}$$

Let us define the error $\mathbf{e}(t) \triangleq \mathbf{x}(t) - \mathbf{x_a}(t)$. By subtracting Eq.(12.6) evaluated at $\mathbf{x_a}(t)$ from Eq.(12.8) evaluated at $\mathbf{x}(t)$, we find that $\mathbf{e}(t)$ satisfies the equation:

$$\dot{\mathbf{e}}(t) = \mathbf{A} \cdot [\mathbf{e}(t) + \mathbf{\Delta x}(t)] \tag{12.9}$$

with $\mathbf{e}(t_0) = 0$ since both trajectories, $\mathbf{x_a}(t)$ and $\mathbf{x}(t)$, start out from identical initial conditions.

Let us start analyzing the simple scalar case:

$$\dot{e}(t) = a \cdot [e(t) + \Delta x(t)] \tag{12.10}$$

For reasons that the reader will soon understand, we shall assume that a, e, and Δx belong to $\mathbb{C}$. We shall furthermore request $\mathbb{R}\mathrm{e}\{a\}$ to be negative.

We shall also ask that $|\Delta x| \leq w$, with w being some positive constant.

$e(t)$ can be written in polar notation as:

$$e(t) = \rho(t) \cdot e^{j\theta(t)} \tag{12.11}$$

where $\rho(t) = |e(t)|$.

Then, Eq.(12.10) becomes:

$$\dot{\rho}(t) \cdot e^{j\theta(t)} + \rho(t) \cdot e^{j\theta(t)} \cdot j \cdot \dot{\theta}(t) = a \cdot [\rho(t) \cdot e^{j\theta(t)} + \Delta x(t)] \tag{12.12}$$

or:

$$\dot{\rho}(t) + \rho(t) \cdot j \cdot \dot{\theta}(t) = a \cdot [\rho(t) + \Delta x(t) \cdot e^{-j\theta(t)}] \tag{12.13}$$

Let us take now only the real part of the last equation:

$$\begin{aligned} \dot{\rho}(t) &= \mathbb{R}\mathrm{e}\{a\} \cdot \rho(t) + \mathbb{R}\mathrm{e}\{a \cdot \Delta x(t) \cdot e^{-j\theta(t)}\} \\ &\leq \mathbb{R}\mathrm{e}\{a\} \cdot \rho(t) + |a| \cdot |\Delta x(t)| \\ &\leq \mathbb{R}\mathrm{e}\{a\} \cdot \left[\rho(t) - \left|\frac{a}{\mathbb{R}\mathrm{e}\{a\}}\right| \cdot w\right] \end{aligned} \tag{12.14}$$

Then, as $\mathbb{R}\mathrm{e}\{a\}$ is negative and $\rho(0) = 0$, it will always be true that:

$$|e(t)| = \rho(t) \leq \left|\frac{a}{\mathbb{R}\mathrm{e}\{a\}}\right| \cdot w \tag{12.15}$$

since, when ρ reaches the upper limit, its derivative becomes negative (or zero).

Before proceeding to the most general situation, we shall apply this last result to the diagonal case. Let the $\mathbf{A}$–matrix in Eq.(12.9) be a diagonal matrix with complex diagonal elements $a_{i,i}$ with negative real parts. We shall also assume that:

$$|\mathbf{\Delta x}(t)| \leq \mathbf{w} \tag{12.16}$$

Repeating the scalar analysis for each component of $\mathbf{e}(t)$, we find that:

$$|\mathbf{e}| \leq |\mathbb{R}\mathrm{e}\{\mathbf{A}\}^{-1} \cdot \mathbf{A}| \cdot \mathbf{w} \tag{12.17}$$

Now, let us return once more to Eq.(12.9), this time assuming that the **A**–matrix be Hurwitz and diagonalizable, i.e., that the original system is asymptotically stable and can be decoupled.

Let us assume that the quantum and hysteresis were adjusted such that:

$$|\mathbf{\Delta x}| = |\mathbf{q} - \mathbf{x}| \leq \mathbf{\Delta Q} \tag{12.18}$$

where each element of vector $\mathbf{\Delta Q}$ contains the larger of the corresponding quantum and hysteresis width.

Let $\mathbf{\Lambda}$ be a diagonal eigenvalue matrix of **A**, and let **V** be a corresponding right eigenvector matrix. The matrix **V** is sometimes also referred to as a right modal matrix. Then:

$$\mathbf{A} = \mathbf{V} \cdot \mathbf{\Lambda} \cdot \mathbf{V}^{-1} \tag{12.19}$$

is the spectral decomposition of the **A**–matrix.

We introduce the variable transformation:

$$\mathbf{z}(t) = \mathbf{V}^{-1} \cdot \mathbf{e}(t) \tag{12.20}$$

Using the new variable **z**, Eq.(12.9) can be rewritten as follows:

$$\mathbf{V} \cdot \dot{\mathbf{z}}(t) = \mathbf{A} \cdot [\mathbf{V} \cdot \mathbf{z}(t) + \mathbf{\Delta x}(t)] \tag{12.21}$$

and then:

$$\dot{\mathbf{z}}(t) = \mathbf{V}^{-1} \cdot \mathbf{A} \cdot [\mathbf{V} \cdot \mathbf{z}(t) + \mathbf{\Delta x}(t)] \tag{12.22}$$

$$= \mathbf{\Lambda} \cdot [\mathbf{z}(t) + \mathbf{V}^{-1} \cdot \mathbf{\Delta x}(t)] \tag{12.23}$$

¿From Eq.(12.18), it results that:

$$|\mathbf{V}^{-1} \cdot \mathbf{\Delta x}| \leq |\mathbf{V}^{-1}| \cdot \mathbf{\Delta Q} \tag{12.24}$$

Taking into account that the $\mathbf{\Lambda}$–matrix is diagonal, it turns out that Eq.(12.23) is the diagonal case that we analyzed before, and consequently from Eq.(12.17), it results that:

$$|\mathbf{z}(t)| \leq |\mathbb{R}\mathrm{e}\{\mathbf{\Lambda}\}^{-1} \cdot \mathbf{\Lambda}| \cdot |\mathbf{V}^{-1}| \cdot \mathbf{\Delta Q} \tag{12.25}$$

and therefore:

$$|\mathbf{e}(t)| \leq |\mathbf{V}| \cdot |\mathbf{z}(t)| \leq |\mathbf{V}| \cdot |\mathbb{R}\mathrm{e}\{\mathbf{\Lambda}\}^{-1} \cdot \mathbf{\Lambda}| \cdot |\mathbf{V}^{-1}| \cdot \mathbf{\Delta Q} \tag{12.26}$$

Thus, we can conclude that:

$$|\mathbf{x}(t) - \mathbf{x_a}(t)| \leq |\mathbf{V}| \cdot |\mathbb{R}\mathrm{e}\{\mathbf{\Lambda}\}^{-1} \cdot \mathbf{\Lambda}| \cdot |\mathbf{V}^{-1}| \cdot \mathbf{\Delta Q} \tag{12.27}$$

Inequality Eq.(12.27) has strong theoretical and practical implications.

It can be easily seen that the error bound is proportional to the quantum and, for any quantum adopted, the error is always bounded.

It is also important to notice that the inequality of Eq.(12.27) is an analytical expression for the *global error bound*. Discrete–time methods lack similar formulae. The fact that Eq.(12.27) is independent of initial conditions and input trajectories promises additional important theoretical and practical advantages.

In some way, the QSS method offers an intrinsic error control without requiring the use of adaptation rules. Indeed:

> The QSS method is always stable without using implicit formulae at all.

The importance of this statement cannot be stressed enough. It revolutionizes the field of numerical ODE (and DAE) solution.

12.3 Choosing Quantum and Hysteresis Width

The QSS method requires the choice of an adequate quantum and hysteresis width. Although we mentioned that the error is, at least for LTI systems, always bounded, an appropriate quantization must be chosen in order to obtain decent simulation results.

The inequality of Eq.(12.27) can be used for quantization design. Given a desired error bound, it is not difficult to find an appropriate value of $\mathbf{\Delta Q}$ that satisfies that inequality. Let us illustrate this design in a simple example. Consider the system:

$$\begin{array}{rcl} \dot{x}_{a_1} & = & x_{a_2} \\ \dot{x}_{a_2} & = & -x_{a_1} - x_{a_2} + u(t) \end{array} \tag{12.28}$$

A set of matrices of eigenvalues and eigenvectors (calculated with MATLAB) are:

$$\mathbf{\Lambda} = \begin{pmatrix} -0.5 + 0.866j & 0 \\ 0 & -0.5 - 0.866j \end{pmatrix}$$

and:

$$\mathbf{V} = \begin{pmatrix} 0.6124 - 0.3536j & 0.6124 + 0.3536j \\ 0.7071j & -0.7071j \end{pmatrix}$$

Then:

$$\mathbf{T} \triangleq |\mathbf{V}| \cdot |\mathbb{R}\mathrm{e}\{\mathbf{\Lambda}\}^{-1} \cdot \mathbf{\Lambda}| \cdot |\mathbf{V}^{-1}| = \begin{pmatrix} 2.3094 & 2.3094 \\ 2.3094 & 2.3094 \end{pmatrix} \tag{12.29}$$

Let us consider that the goal is to simulate Eq.(12.28) for an arbitrary initial condition and input trajectory with an error less than or equal to 0.1 in each variable. Then, a quantum:

$$\mathbf{\Delta Q} = \begin{pmatrix} 0.05/2.3094 \\ 0.05/2.3094 \end{pmatrix} = \begin{pmatrix} 0.0217 \\ 0.0217 \end{pmatrix} \tag{12.30}$$

is sufficiently small to ensure that the error cannot exceed the given bound.

Although the inequality of Eq.(12.27) can be used to compute an upper bound for the error as a function of the quantum and the hysteresis width, the measure will often turn out to be quite conservative.

In fact, using quantum and hysteresis width equal to 0.05 in each variable of the system of Eq.(12.28) and applying $u(t) = 1$, we arrive at the simulation results shown in Fig.11.15. The predicted error bound is 0.23094 in each variable. However, the maximum error obtained in that simulation is considerably smaller than this bound.

Except in specific applications, where we would need to ensure a certain error bound, we do not want to calculate eigenvectors and eigenvalues before performing the simulation. A practical rule to avoid this is to use a quantum proportional to the estimated amplitude of each variable trajectory (assuming that we know in advance the order of magnitude of the values reached by each state variable).

The reader may have already noticed that, in all of the examples discussed so far, the hysteresis width was chosen to be equal to the quantum size. However, we did not provide any rationale for that choice.

The problem of hysteresis width selection is discussed in [12.11]. The conclusion is that it should be chosen equal to the quantum. The reason is that, in this way, the presence of hysteresis does not modify the error bound (cf. Eq.(12.4)), while the final oscillation frequency is being minimized.

The reduction of the oscillation frequency is due to the fact that the minimum time between successive changes in a quantized variable q_i is proportional to the inverse of the hysteresis (assuming that the quantum is greater or equal than the hysteresis width), as has been shown in [12.10].

Let us illustrate this idea with the simulation of the first–order system of Eq.(12.5), using a quantum equal to 1 and different hysteresis values.

Figure 12.2 shows the simulation results with a hysteresis width of $\varepsilon = 1$, $\varepsilon = 0.6$, and $\varepsilon = 0.1$, respectively. In all three cases, the maximum error is bounded by the same value. In theory, the bound is equal to 1, but in the simulations, we can observe that the maximum error is always 0.5.

However, the steady–state oscillation frequency increases as the hysteresis width becomes smaller. In fact, that frequency can be calculated as:

$$f = \frac{1}{2\varepsilon}$$

Then, it is clear that by choosing the hysteresis width equal to the quantum, the frequency is minimized without increasing the error. Reducing the

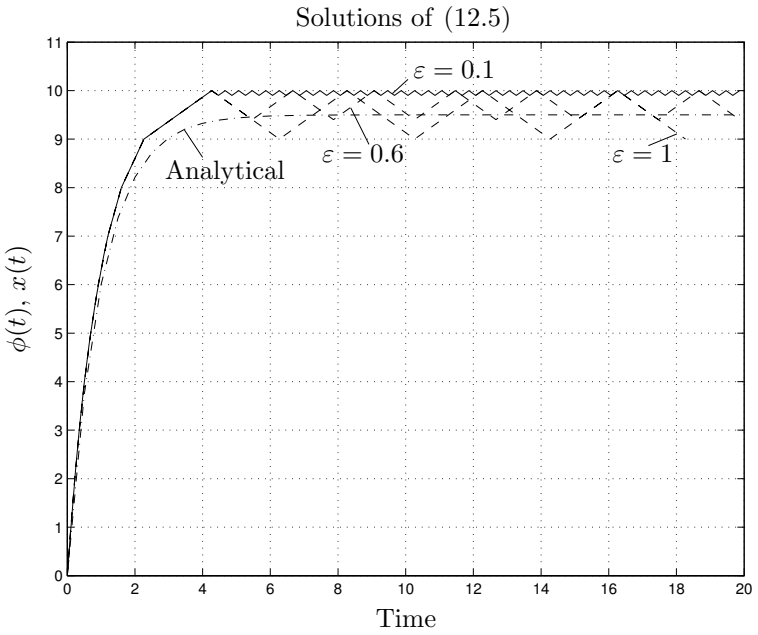

FIGURE 12.2. Simulation of Eq.(12.5) with different hysteresis values.

steady–state oscillation frequency reduces in the number of steps performed by the algorithm, and consequently reduces the computational cost.

12.4 Input Signals in the QSS Method

In the previous chapter, we mentioned that the QSS method allows the simulation of time–invariant systems with piecewise constant input signals. However, we did not say how these signals can be incorporated into the simulation model.

In the DEVS simulation model, each event represents a change in a piecewise constant trajectory. Consequently, input trajectories can be incorporated as sequences of events.

Looking at the block diagram of Fig.11.12, the input signals $\mathbf{u}(t)$ seem to come from the external world, and it is not clear, where the corresponding sequences of events should be generated.

In the context of a DEVS simulation, all events must emanate from an atomic DEVS model. Hence a new DEVS model class must be created that generates those sequences of events. The input function models must then be coupled with the rest of the system for the purpose of simulation.

Suppose that we have a piecewise constant input signal $u(t)$ that assumes the values v_1, v_2, ..., v_j, ... at times t_1, t_2, ..., t_j, ..., respectively. A

DEVS model that produces events in accordance with this input signal can be specified as follows:

$$
\begin{aligned}
&M_6 = (X, Y, S, \delta_{\text{int}}, \delta_{\text{ext}}, \lambda, ta), \text{ where} \\
&X = \emptyset \\
&Y = \mathbb{R} \times \mathbb{N} \\
&S = \mathbb{N} \times \mathbb{R}_0^+ \\
&\delta_{\text{int}}(s) = \delta_{\text{int}}(j, \sigma) = (j + 1, t_{j+1} - t_j) \\
&\lambda(s) = \lambda(j, \sigma) = (v_j, 0) \\
&ta(s) = ta(j, \sigma) = \sigma
\end{aligned}
$$

Notice that, in this model, the external transition function δ_{ext} is not defined, as it it will never be called, since the model is not designed to ever receive input events.

A particular case of model M_6 in PowerDEVS is the step function model that we invoked from within the models of Fig.11.9 and Fig.11.14.

ATOMIC MODEL STEP1
State Variables and Parameters:
float *sigma*;
int *j*; //states
float *y*; //output
float *T*[3], *v*[3], *inf*; //parameters

Init Function:
va_list *parameters*;
va_start(*parameters*, *t*);
inf = 1*e*10;
T[0] = 0;
T[1] = va_arg(*parameters*, double);
T[2] = *inf*;
v[0] = va_arg(*parameters*, double);
v[1] = va_arg(*parameters*, double);
sigma = 0;
j = 0;

Time Advance Function:
return *sigma*;

Internal Transition Function:
sigma = *T*[*j* + 1] − *T*[*j*];
j = *j* + 1;

Output Function:
y = *v*[*j*];
return Event(&*y*,0);

The parameters defined in the graphical block of this DEVS model are

the step time, the initial value of the output trajectory, and the final value after the step.

An interesting advantage of the QSS method is that it deals with input trajectory changes in an asynchronous way. The event indicating a change in the signal is always processed at the correct instant of time, producing instantaneous changes in the slopes of the state variable trajectories that are directly affected.

This is an intrinsic characteristic of the method, and it is obtained without modifying the DEVS models corresponding to the quantized integrators and the static functions. In contrast, discrete–time methods require a special treatment in order to perform a step at the exact moment when an input change is supposed to occur. We shall revisit to this issue once more later in this chapter, demonstrating the advantages of the QSS method in the context of simulating discontinuous systems.

Up to this point, only piecewise constant input trajectories have been considered. In most applications, the input signals take on more general forms. However, these can be approximated by piecewise constant trajectories with the addition of quantization functions, and thus, they can be represented by DEVS models.

For example, the DEVS model M_7, shown below, generates an event trajectory that approximates a sine function with angular frequency ω and amplitude A using a quantum Δu.

$$
\begin{aligned}
&M_7 = (X, Y, S, \delta_{\text{int}}, \delta_{\text{ext}}, \lambda, ta), \text{ where} \\
&X = \emptyset \\
&Y = \mathbb{R} \times \mathbb{N} \\
&S = \mathbb{R} \times \mathbb{R}_0^+ \\
&\delta_{\text{int}}(s) = \delta_{\text{int}}(\tau, \sigma) = (\tilde{\tau}, \tilde{\sigma}) \\
&\lambda(s) = \lambda(\tau, \sigma) = (A \cdot \sin(\omega\tau), 1) \\
&ta(s) = ta(\tau, \sigma) = \sigma
\end{aligned}
$$

with:

$$
\tilde{\sigma} = \begin{cases} \dfrac{\arcsin[\sin(\omega\tau) + \Delta u/A]}{\omega} - \tau & \text{if } (\sin(\omega\tau) + \Delta u/A \leq 1 \wedge \cos(\omega\tau) > 0) \\ & \quad \vee \sin(\omega\tau) - \Delta u < -1 \\ \dfrac{\pi \cdot \text{sign}(\tau) - \arcsin[\sin(\omega\tau) - \Delta u/A]}{\omega} - \tau & \text{otherwise} \end{cases}
$$

and:

$$
\tilde{\tau} = \begin{cases} \tau + \tilde{\sigma} & \text{if } \omega(\tau + \tilde{\sigma}) < \pi \\ \tau + \tilde{\sigma} - \dfrac{2\pi}{\omega} & \text{otherwise} \end{cases}
$$

The DEVS model M_7 can be encoded in PowerDEVS as follows:

ATOMIC MODEL SINUS
State Variables and Parameters:
float $tau, sigma$; //states
float y; //output
float A, w, phi, du, pi; //parameters

Init Function:
va_list $parameters$;
va_start($parameters$, t);
$pi = 2 * asin(1)$;
A = va_arg($parameters$, double);
w = va_arg($parameters$, double)$*2 * pi$;
phi = va_arg($parameters$, double);
du = va_arg($parameters$, double);
$sigma = 0$;
$tau = phi/w$;

Time Advance Function:
return $sigma$;

Internal Transition Function:
if $(((\sin(w * tau) + du/A <= 1)$ && $(\cos(w * tau) > 0))$ || $(\sin(w * tau) - du/A < -1))$ {
$sigma = \text{asin}(\sin(w * tau) + du/A)/w - tau$;
}
else {
if $(tau > 0)$ {
$sigma = (pi - \text{asin}(\sin(w * tau) - du/A))/w - tau$;
}
else {
$sigma = (-pi - \text{asin}(\sin(w * tau) - du/A))/w - tau$;
};
};
$tau = tau + sigma$;
if $(tau * w >= pi)\{tau = tau - 2 * pi/w;\}$;

Output Function:
$y = A * \sin(w * tau)$;;
return Event(&y,0);

The trajectory generated by this model with parameters $A = 2.001$, $\omega = 0.5$, and $\Delta u = 0.2$ is shown in Fig.12.3.

A piecewise constant trajectory could also be obtained using a constant time step. However, the previously advocated approximation is better in QSS, since the quantization in the values ensures that the distance between the continuous signal an the piecewise constant signal is always less than the quantum. This fact can be easily noticed in Fig.12.3. In contrast, the maximum error would have depended on the relationship between the time step and the signal frequency, had a constant time step been used.

The input signal quantization introduces a new error to the simulation.

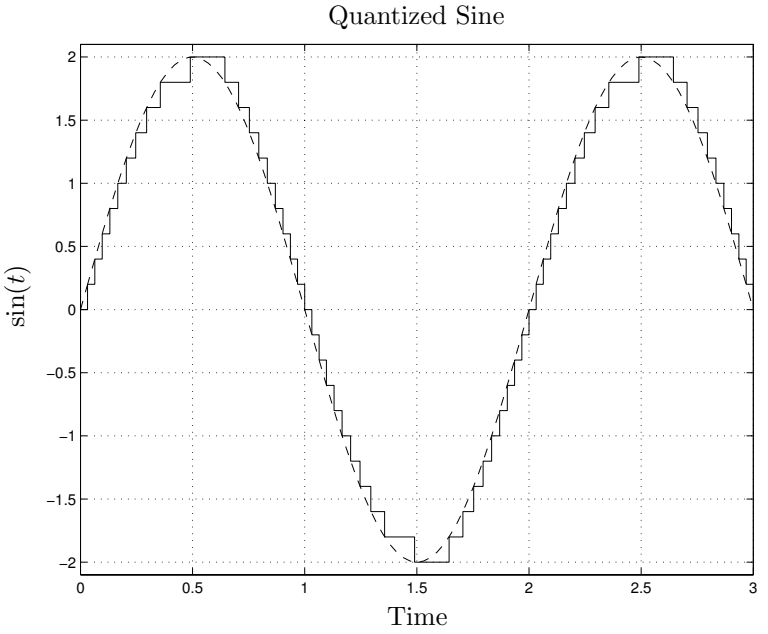

FIGURE 12.3. Piecewise constant sine trajectory.

In the particular case of LTI systems, the presence of the input quantization error transforms Eq.(12.27) into:

$$\begin{aligned} |\mathbf{x}(t) - \mathbf{x_a}(t)| \quad \leq \quad & |\mathbf{V}| \cdot |\mathbb{R}\mathrm{e}\{\mathbf{\Lambda}\}^{-1} \cdot \mathbf{\Lambda}| \cdot |\mathbf{V}^{-1}| \cdot \mathbf{\Delta Q} \\ & + |\mathbf{V}| \cdot |\mathbb{R}\mathrm{e}\{\mathbf{\Lambda}\}^{-1} \cdot \mathbf{V}^{-1} \cdot \mathbf{B}| \cdot \mathbf{\Delta u} \end{aligned} \tag{12.31}$$

where $\mathbf{\Delta u}$ is the vector with the quanta adopted in the input signals. This formula can be derived following the approach developed in [12.18] (cf. Hw.[H12.2]).

12.5 Startup and Output Interpolation

The QSS method startup consists in assigning appropriate initial conditions to the atomic DEVS models that perform the simulation.

The quantized integrator state can be written as $s = (x, d_x, k, \sigma)$ (cf. model M_5 on page 543), where x is the state variable, d_x is its derivative, k is the index corresponding to the quantized variable q_k, and σ is the time advance.

It is clear that we must choose $x = x_i(t_0)$ and k such that $Q_k \leq x_i(t_0) \leq Q_{k+1}$. Finally, appropriate values of d_x and σ can be found from the corresponding state equation $f_i(\mathbf{q}, \mathbf{u})$.

Yet, there is a much simpler solution. If we choose $\sigma = 0$, all quantized integrators will perform internal transitions at the beginning of the simulation and send their initial values to the static functions.

Then, the models associated with the static functions, f_i, will calculate their output values, producing output events instantaneously. These output events will carry the values of the state derivatives. After that, the quantized integrators will receive the correct values of d_x, and they will calculate the corresponding σ in the external transition.

However, this behavior will only be observed if the quantized integrators receive the input events, arriving from the static functions, *after* they performed their own internal transitions. The problem is that, after the first quantized integrator performs its transition, not only the other quantized integrators but also some static models will have their σ set equal to 0, because they undergo an external transition due to the quantized integrator output event.

Thus, it is necessary to establish priorities between the components, in order to ensure that, when some models schedule their next transition for the same instant of time, the quantized integrators are those that perform it first. This can be easily accomplished using the tie–breaking function *Select* mentioned in Section 11.4.

These considerations also solve the problem of the initial conditions of the static model. They can be arbitrarily chosen, since the static models will receive input events arriving from the quantized integrators at the beginning of the simulation, and then their external transition functions will set the appropriate states.

Finally, the input signal generator models must start with their σ set equal to 0, and the rest of the states must be chosen such that the first output event corresponds to the initial value of the input signal.

In PowerDEVS, we treat these problems in the init function. The priorities between subsystems can be easily chosen from the *Edit* menu in the model editor.

Yet, there is another solution that avoids the need of priorities: We can check the external transition function of the quantized integrators for the condition $\sigma = 0$. If that condition is true, this means that an internal transition is going to occur. Thus, we can just leave $\sigma = 0$ in that case, and this solves the initialization problems without using priorities.

When it comes to output interpolation, we already know that the state trajectories in QSS assume particular forms: They are piecewise linear and continuous.

Hence if we know the values adopted by variable x_i at all event times, we can interpolate using straight–line segments, in order to obtain the exact solution of Eq.(12.2). In fact, to do that, we only require the values after external transitions, because the slope does not change during internal transitions.

Consequently, the problem of output interpolation has a straightforward

solution in the QSS method.

It is important to remember that Eq.(12.27) and Eq.(12.31) are valid for all values of time t. Thus, we can ensure that the interpolated values remain inside their theoretical error bound, which is an interesting and unusual characteristic for a simulation method.

12.6 Second–order QSS

As we saw along this chapter, the QSS method exhibits strong theoretical and practical properties that make the method attractive for use in the simulation of continuous systems. Unfortunately, the method is only first–order accurate, and therefore, the simulation results obtained with this method cannot be very accurate.

The inequality of Eq.(12.27) states that the error bound grows linearly with the quantum. Thus, if we want to reduce the error bound by a certain amount, we have to reduce the quantum in the same proportion. The problem is that also the time interval between successive events, i.e., the time advance σ in the DEVS model, is proportional to the quantum (cf. model M_5 on page 543). Consequently, the error reduction results in a proportional increment in the number of computations.

To improve the situation, a second–order accurate QSS method was proposed in [12.13]. This new approximation, called *second–order quantized state systems method*, or QSS2 method in short, exhibits similar stability, convergence, and accuracy properties as the previously introduced QSS method.

The basic idea behind this second–order accurate method is the use of *first–order quantizers*, replacing the simple hysteretic quantizers that were used in the design of the QSS method.

A hysteretic quantizer with equal quantum and hysteresis width can be viewed as a system producing a piecewise constant output trajectory that only changes when the difference between output and input reaches a certain threshold level, i.e., the quantum.

Following this idea, a first–order quantizer has been defined as a system that produces a piecewise linear output trajectory having discontinuities only, when its difference with the input reaches the quantum. This behavior is illustrated in Fig.12.4.

Formally, we say that the trajectories $x_i(t)$ and $q_i(t)$ are related by a first–order quantization function, if they satisfy:

$$q_i(t) = \begin{cases} x_i(t) & \text{if} \quad t = t_0 \vee |q_i(t^-) - x_i(t^-)| = \Delta Q \\ q_i(t_j) + m_j \cdot (t - t_j) & \text{otherwise} \end{cases} \tag{12.32}$$

with the sequence $t_0, \ldots, t_j, \ldots$ defined as:

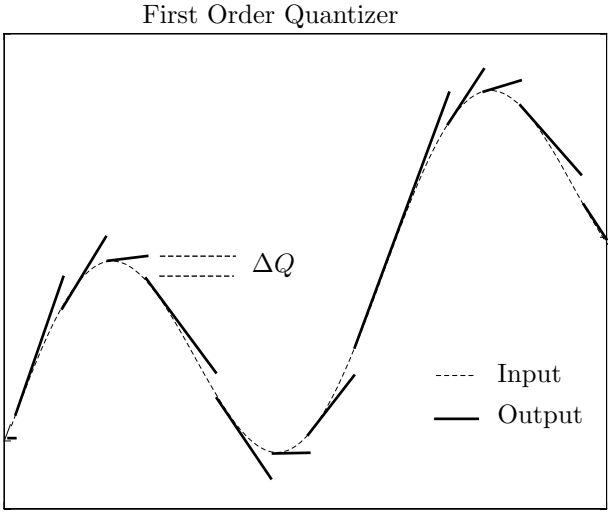

FIGURE 12.4. Input and output trajectories of a first–order quantizer.

$$t_{j+1} = \min(t), \quad \forall t > t_j \wedge |x_i(t_j) + m_j \cdot (t - t_j) - x_i(t)| = \Delta Q$$

and the slopes:

$$m_0 = 0; \quad m_j = \dot{x}_i(t_j^-), \quad j = 1, \ldots, k, \ldots$$

The QSS2 method then simulates a system like that of Eq.(12.2), where the components of $\mathbf{q}(t)$ and $\mathbf{x}(t)$ are related componentwise by first–order quantization functions. As a consequence, the quantized variable trajectories $q_i(t)$ are piecewise linear.

In QSS, we had to add hysteresis in order to ensure that the trajectories become piecewise constant avoiding illegitimacy. The reader may wonder why we did not need to add hysteresis here. The reason is that hysteresis is implicitly present in the definition of the first–order quantization function. Indeed, the absolute value in Eq.(12.32) expresses that hysteretic behavior.

Remember that, in QSS, not only the quantized variables, but also the state derivatives are piecewise constant. In this way, we were able to affirm that the state variables have piecewise linear trajectories, and we exploited this features in building the DEVS model.

Unfortunately, we cannot find this kind of particular trajectories in QSS2. Although the quantized variables are piecewise linear, this does not mean that the state derivatives are piecewise linear as well, even if all inputs possess piecewise linear trajectories. The reason is that a nonlinear function, f_i, applied to a set of piecewise linear trajectories does not necessarily result in a trajectory that is piecewise linear.

Thus, we shall be able to simulate QSS2 approximations to LTI systems only. Of course,, the QSS2 method can be applied to general nonlinear systems as well, but in this case, the simulation results will not coincide exactly with the solutions of Eq.(12.2).

In the linear case, however, provided that the input trajectories are piecewise linear, the state derivatives turn out to be piecewise linear as well, and then, the state variables assume continuous piecewise parabolic trajectories.

Using these facts, we can proceed following the same lines of thought that we used in QSS in order to build the DEVS model, i.e., we can split the model into quantized integrators and static functions. But now, the atomic models are quite different from before, since they must calculate and take into account not only the values but also the slopes of the trajectories. Moreover, the events will have to carry both value and slope information.

The quantized integrators in QSS2 will be formed by an integrator and a first–order quantizer. We shall call them *second–order quantized integrators*. The reason for this name is that they calculate the state trajectories using their first and second derivatives (i.e., state derivative values and their slopes).

In order to obtain a DEVS model of a second–order quantized integrator, we shall suppose that a state derivative is described, in a certain interval $[t_k, t_{k+1}]$, by:

$$\dot{x}(t) = d_x(t_k) + m_{d_x}(t_k) \cdot (t - t_k) \tag{12.33}$$

where $d_x(t_k)$ is the state derivative at time t_k, and $m_{d_x}(t_k)$ is the corresponding linear slope. The slope of the state derivative , m_{d_x}, will, in general, be different from the slope of the quantized state variable, m_q.

Then, the state variable trajectory can be written as:

$$x(t) = x(t_k) + d_x(t_k) \cdot (t - t_k) + \frac{m_{d_x}(t_k)}{2} \cdot (t - t_k)^2 \tag{12.34}$$

If t_k is an instant, at which a change occurs in the quantized variable, i.e., t_k is the time instant of an internal transition, then $q(t_k)$ will have the same value and slope as $x(t_k)$:

$$q(t) = x(t_k) + d_x(t_k) \cdot (t - t_k) \tag{12.35}$$

and then, the time instant at which $x(t)$ and $q(t)$ differ from each other by ΔQ can be calculated as:

$$t = t_k + \sqrt{\frac{2 \cdot \Delta Q}{|m_{d_x}(t_k)|}} \tag{12.36}$$

If t_k is another instant in time, i.e., the time of an external transition, the time instant, at which $|q(t) - x(t)| = \Delta Q$ must be recalculated. Now, the quantized trajectory can be written as:

$$q(t) = q(t_k) + m_q(t_k) \cdot (t - t_k) \tag{12.37}$$

and then, we have to calculate the value of t, at which $|q(t) - x(t)| = \Delta Q$, by finding the roots of the corresponding quadratic polynomial.

A DEVS model that can represent the behavior of a second–order quantized integrator is presented below:

$M_8 = (X, S, Y, \delta_{\text{int}}, \delta_{\text{ext}}, \lambda, ta)$, where:
$X = \mathbb{R}^2 \times \mathbb{N}$
$S = \mathbb{R}^5 \times \mathbb{R}_0^+$
$Y = \mathbb{R}^2 \times \mathbb{N}$
$\delta_{\text{int}}(d_x, m_{d_x}, x, q, m_q, \sigma) = (d_x + m_{d_x} \cdot \sigma, m_{d_x}, \tilde{q}, \tilde{q}, d_x + m_{d_x} \cdot \sigma, \sigma_1)$
$\delta_{\text{ext}}(d_x, m_{d_x}, x, q, m_q, \sigma, e, x_v, m_{x_v}, p) = (x_v, m_{x_v}, \tilde{x}, q + m_q \cdot e, m_q, \sigma_2)$
$\lambda(d_x, m_{d_x}, x, q, m_q, \sigma) = (\tilde{q}, d_x + m_{d_x} \cdot \sigma, 0)$
$ta(d_x, m_{d_x}, x, q, m_q, \sigma) = \sigma$

where:

$$\tilde{q} = x + d_x \cdot \sigma + \frac{m_{d_x}}{2} \cdot \sigma^2; \quad \tilde{x} = x + d_x \cdot e + \frac{m_{d_x}}{2} \cdot e^2$$

$$\sigma_1 = \begin{cases} \sqrt{\dfrac{2 \cdot \Delta Q}{|m_{d_x}|}} & \text{if} \quad m_{d_x} \neq 0 \\ \infty & \text{otherwise} \end{cases} \tag{12.38}$$

and σ_2 can be calculated as the smallest positive solution of:

$$|\tilde{x} + x_v \cdot \sigma_2 + \frac{m_{x_v}}{2} \cdot \sigma_2^2 - (q + m_q \cdot e + m_q \cdot \sigma_2)| = \Delta Q \tag{12.39}$$

The model M_8 represents a second–order quantized integrator with piecewise linear input trajectories exactly.

Equation (12.38) and Eq.(12.39) calculate the time advance, that is, the time instant at which the distance between the piecewise parabolic state trajectory $x(t)$ and the piecewise linear quantized trajectory $q(t)$ reaches the quantum ΔQ.

The corresponding PowerDEVS atomic model can be coded as follows:

ATOMIC MODEL QSS2INT
State Variables and Parameters:
float $dx, mdx, X, q, mq, sigma$; //states
float $y[2]$; //output
float inf, dq; //parameters

Init Function:

```
va_list parameters;
va_start(parameters, t);
dq = va_arg(parameters, double);
X = va_arg(parameters, double);
inf = 1e10;
q = X;
dx = 0;
mdx = 0;
mq = 0;
sigma = 0;
```

Time Advance Function:

```
return sigma;
```

Internal Transition Function:

```
X = X + dx * sigma + mdx/2 * sigma * sigma;
q = X;
dx = dx + mdx * sigma;
mq = dx;
if (mdx == 0) {
    sigma = inf;
}
else
    sigma = sqrt(2 * dq/fabs(mdx));
};
```

External Transition Function:

```
float *xv;
float a, b, c, s;
xv = (float*)(x.value);
X = X + dx * e + mdx/2 * e * e;
dx = xv[0]; //input value
mdx = xv[1]; //input slope
if (sigma != 0) {
    q = q + mq * e;
    a = mdx/2;
    b = dx - mq;
    c = X - q + dq;
    sigma = inf;
    if (a == 0) {
        if (b != 0) {
            s = -c/b;
            if (s > 0) {sigma = s;};
            c = X - q - dq;
            s = -c/b;
            if ((s > 0) && (s < sigma)) {sigma = s;};
        };
    }
    else {
        s = (-b + sqrt(b * b - 4 * a * c))/2/a;
        if (s > 0) {sigma = s;};
        s = (-b - sqrt(b * b - 4 * a * c))/2/a;
        if ((s > 0) && (s < sigma)) {sigma = s;};
        c = X - q - dq;
```

```
        s = (−b + sqrt(b ∗ b − 4 ∗ a ∗ c))/2/a;
        if ((s > 0) && (s < sigma)) {sigma = s;};
        s = (−b − sqrt(b ∗ b − 4 ∗ a ∗ c))/2/a;
        if ((s > 0) && (s < sigma)) {sigma = s;};
    };
};
```

Output Function:

```
y[0] = X + dx ∗ sigma + mdx/2 ∗ sigma ∗ sigma;
y[1] = u + mdx ∗ sigma;
return Event(&y[0], 0);
```

In a QSS2 simulation, we represent the integrators with models of the M_8 class, instead of using those of the M_5 class.

For representing static functions, we used models of the M_3 class in the QSS method. However, the M_3 model does not take into account the slopes. Thus, the representation of static functions in QSS2 requires using a different DEVS model as well.

Each component f_j of a static vector function $\mathbf{f}(\mathbf{q}, \mathbf{u})$ receives the piecewise linear trajectories of the quantized states and input variables.

Let us define $\mathbf{v} \triangleq [\mathbf{q}; \mathbf{u}]$. Each component of $\mathbf{v}$ has a piecewise linear trajectory:

$$v_j(t) = v_j(t_k) + m_{v_j}(t_k) \cdot (t - t_k)$$

Then, the output of the static function can be written as:

$$\dot{x}_i(t) = f_i(\mathbf{v}(t)) = f_i(v_1(t_k) + m_{v_1}(t_k) \cdot (t - t_k), \ldots, v_l(t_k) + m_{v_l}(t_k) \cdot (t - t_k))$$

where $l \triangleq n + m$ is the number of components of $\mathbf{v}(t)$.

Defining $\mathbf{m_v} \triangleq [m_{v_1}, \ldots, m_{v_l}]^T$, the last equation can be rewritten as:

$$\dot{x}_i(t) = f_i(\mathbf{v}(t)) = f_i(\mathbf{v}(t_k) + \mathbf{m_v}(t_k) \cdot (t - t_k))$$

which can be developed into a Taylor series as follows:

$$\dot{x}_i(t) = f_i(\mathbf{v}(t)) = f_i(\mathbf{v}(t_k)) + \left(\frac{\partial f_j}{\partial \mathbf{v}}(\mathbf{v}(t_k)) \right)^T \cdot \mathbf{m_v}(t_k) \cdot (t - t_k) + \ldots \quad (12.40)$$

Then, a piecewise linear approximation of the output can be obtained by truncating the Taylor series after the first two terms of Eq.(12.40).

In the linear time–invariant case, we have $\mathbf{f}(\mathbf{v}(t)) = \mathbf{A} \cdot \mathbf{v}(t)$, and therefore: $f_i(\mathbf{v}(t)) = \mathbf{a_i}^T \cdot \mathbf{v}(t)$ where $\mathbf{a_i} \in \mathbb{R}^l$. Then:

$$\dot{x}_i(t) = \mathbf{a_i}^T \cdot \mathbf{v}(t_k) + \mathbf{a_i}^T \cdot \mathbf{m_v}(t_k) \cdot (t - t_k)$$

which means that the output value and its slope are obtained as linear combinations of the input values and their slopes, respectively.

The construction of the corresponding DEVS model is left to the reader (cf. Hw.[H12.4]).

The nonlinear case is a bit more complicated. The expression of Eq.(12.40) requires the knowledge of the partial derivatives of function f_i evaluated at the successive values of the quantized state and input variables. In a general case, we may not have a closed–form expression for these derivatives, in which case we shall need to approximate them numerically.

A DEVS model that follows this idea, representing a static nonlinear function $f_i(\mathbf{v}) = f(v_1, \ldots, v_l)$ and taking into account input and output values and their slopes can be coded as follows :

$$
\begin{aligned}
&M_9 = (X, S, Y, \delta_{\text{int}}, \delta_{\text{ext}}, \lambda, ta), \text{ where:}\\
&X = \mathbb{R}^2 \times \mathbb{N}\\
&S = \mathbb{R}^{3l} \times \mathbb{R}_0^+\\
&Y = \mathbb{R}^2 \times \mathbb{N}\\
&\delta_{\text{int}}(\mathbf{v}, \mathbf{m_v}, \mathbf{c}, \sigma) = (\mathbf{v}, \mathbf{m_v}, \mathbf{c}, \infty)\\
&\delta_{\text{ext}}(\mathbf{v}, \mathbf{m_v}, \mathbf{c}, \sigma, e, x_v, m_{x_v}, p) = (\tilde{\mathbf{v}}, \tilde{\mathbf{m}}_\mathbf{v}, \tilde{\mathbf{c}}, 0)\\
&\lambda(\mathbf{v}, \mathbf{m_v}, \mathbf{c}, \sigma) = (f_i(\mathbf{v}), m_f, 0)\\
&ta(\mathbf{v}, \mathbf{m_v}, \mathbf{c}, \sigma) = \sigma
\end{aligned}
$$

where $\mathbf{v} = (v_1, \ldots, v_l)^T$ and $\tilde{\mathbf{v}} = (\tilde{v}_1, \ldots, \tilde{v}_l)^T$ are input values. Similarly, $\mathbf{m_v} = (m_{v_1}, \ldots, m_{v_l})^T$ and $\tilde{\mathbf{m}}_\mathbf{v} = (\tilde{m}_{v_1}, \ldots, \tilde{m}_{v_l})^T$ represent the corresponding input slopes.

The coefficients $\mathbf{c} = (c_1, \ldots, c_l)^T$ and $\tilde{\mathbf{c}} = (\tilde{c}_1, \ldots, \tilde{c}_l)^T$ estimate the partial derivatives $\frac{\partial f_i}{\partial v_j}$ that are used to calculate the output slope in accordance with:

$$m_f = \sum_{j=1}^{n} c_j \cdot m_{v_j}$$

When the system undergoes an external transition, the components of $\tilde{\mathbf{v}}$, $\tilde{\mathbf{m}}_\mathbf{v}$, and $\tilde{\mathbf{c}}$ are calculated using the equations:

$$\tilde{v}_j = \begin{cases} x_v & \text{if} \quad p+1 = j \\ v_j + m_{v_j} \cdot e & \text{otherwise} \end{cases}$$

$$\tilde{m}_{v_j} = \begin{cases} m_{x_v} & \text{if} \quad p+1 = j \\ m_{v_j} & \text{otherwise} \end{cases}$$

$$\tilde{c}_j = \begin{cases} \dfrac{f_i(\mathbf{v} + \mathbf{m_v} \cdot e) - f_i(\tilde{\mathbf{v}})}{v_j + m_{v_j} \cdot e - \tilde{v}_j} & \text{if} \quad p+1 = j \quad \wedge \quad v_j + m_{v_j} \cdot e - \tilde{v}_j \neq 0 \\ c_j & \text{otherwise} \end{cases} \tag{12.41}$$

where p denotes the port number, i.e., determines, which of the inputs is currently undergoing an external transition.

If function $f_i(\mathbf{v})$ is linear, this DEVS model represents the behavior of the system exactly, assuming that the components of $\mathbf{v}$ are piecewise linear. However, as we already mentioned, there exists a much simpler and more efficient solution in that case, since the coefficients c_j are constant and coincide with the entries $a_{i,j}$ of matrix $\mathbf{A}$ (cf. Hw.[H12.4]).

The PowerDEVS model for this general nonlinear static function can then be specified as follows:

ATOMIC MODEL STFUNCTION2
State Variables and Parameters:
float $sigma, v[10], mv[10], c[10]$; //states
float $y[2]$; //output
float inf;
int l;

Init Function:
va_list $parameters$;
va_start($parameters$, t);
l = va_arg($parameters$, double);
$inf = 1e10$;
$sigma = inf$;
for (int $i = 0; i < l; i++$) {
$v[i] = 0$;
$mv[i] = 0$;
};
Time Advance Function:
return $sigma$;

Internal Transition Function:
$sigma = inf$;

External Transition Function:
float $*xv$;
float $fv, vaux$;
xv = (float$*$)($x.value$);
for (int $i = 0; i < l; i++$) {
$v[i] = v[i] + mv[i] * e$;
};
$fv = f_j(v)$;//put your function here
$vaux = v[x.port]$;
$v[x.port] = xv[0]$;
$mv[x.port] = xv[1]$;
$y[0] = f_j(v)$; //put your function here
if ($vaux$!= $v[x.port]$) {
$c[x.port] = (fv - y[0])/(vaux - v[x.port])$;
};
$y[1] = 0$;
for (int $i = 0; i < l; i++$) {
$y[1] = y[1] + mv[i] * c[i]$;
};

$sigma = 0;$

Output Function:
return Event($\&y[0], 0$);

The only problem with the PowerDEVS model proposed above is that a new atomic model must be introduced for each distinct function f_j. However, this problem has already been solved in PowerDEVS by introducing a function that parses algebraic expressions. Thus, the corresponding nonlinear function block included in the *Continuous* library of PowerDEVS, offers a parameter consisting in a string that contains the algebraic expression describing the function. That expression, just like any other parameter, can be modified by double clicking on the block.

There are also some particular nonlinear functions, for which the partial derivatives can be calculated analytically and the coefficients c_j do not have to be computed using Eq.(12.41). Some examples of such functions are the sin() function, the multiplier, and the $(\cdot)^2$ block included in the *Continuous* library (cf. Hw.[H12.5]).

By coupling DEVS models of the M_8 and M_9 classes in the same way as we did in Fig.11.12, we can use the QSS2 method to simulate any time–invariant ODE system. Using PowerDEVS, we can simulate any time–invariant ODE system using the QSS2 method by building the block diagram from QSS2INT models, used in place of the quantized hysteretic integrators, and from STFUNCTION2 blocks, used instead of the static functions introduced earlier.

We already mentioned that the QSS2 method shares the main properties of QSS. The reasons behind this assertion can be easily explained. Two variables, $x_i(t)$ and $q_i(t)$, that are related by a first–order quantization function satisfy:

$$|q_i(t) - x_i(t)| \leq \Delta Q_i \quad \forall t \tag{12.42}$$

This inequality is just a particular case of Eq.(12.4) with a constant quantum equal to the hysteresis width. The QSS properties were derived using this inequality, which implies that the method only introduces a bounded perturbation in a system that can be represented in the form of Eq.(12.3). Taking into account that this representation is also valid for QSS2, we conclude that the QSS2 method satisfies the same convergence, stability, and accuracy properties as QSS.

However in nonlinear systems, the QSS2 definition does not coincide exactly with the DEVS simulation. Thus, our analysis only ensures in a strict sense that those properties hold true in the simulation of LTI systems. Although there are many good reasons that allow us to conjecture that the convergence and stability properties would hold true in the simulation of nonlinear systems as well, there has not yet been found a formal proof of this conjecture.

As far as the error bound is concerned, the inequalities of Eq.(12.27) and Eq.(12.31) hold true for the QSS2 method, since they were derived for LTI systems. Thus, if we use the same quantum in QSS and QSS2, we obtain the same error bound in both cases.

This last remark gives rise to a question: Where is the advantage of using the QSS2 method, if it offers the same error bound as the QSS method?

This question can be answered by Eq.(12.38) and Fig.12.4.

On the one hand in QSS, the time advance is proportional to the quantum and to the error. In QSS2 however, it is proportional to the square root of the quantum, as Eq.(12.38) shows. For this reason, we can reduce the quantum without obtaining a proportional increment in the number of calculations, when using the QSS2 method.

This fact can be clearly observed in Fig.12.4, where the use of a simple hysteretic quantizer instead of a first–order quantizer with the same quantum would have resulted in a much larger number of events.

On the other hand, each transition in the QSS2 method involves more computations than in QSS. Thus, if we are not interested in obtaining simulation results that are highly accurate, the QSS method may turn out to be more efficient.

All these facts are discussed in more detail in [12.13], where an experimental comparison between the execution times of both methods was presented, illustrating the characteristics of the two methods as outlined in the above paragraph.

Beside from the theoretical properties that were derived from perturbation analysis, we saw that the QSS method also exhibits practical advantages related to the incorporation of input signals and the exploitation of sparsity. Let us discuss then what happens with these practical issues in the QSS2 method.

The sparsity exploitation is straightforward. We conserve the same simulation structure as in QSS, and each transition only involves calculation at the integrators and static functions that are directly connected to the integrator that undergoes the transition.

When it comes to input signals, we have now further advantages. We not only ensure that changes are being processed as soon as they occur, but we are furthermore able to correctly represent piecewise linear instead of just piecewise constant input trajectories.

Let us illustrate these advantages in the following example, taken from [12.13].

The circuit of Fig.12.5 represents an RLC transmission line. A similar model had already been introduced in Chapter 10 of this book.

This model can be used to study the performance of integrated circuits transmitting data at a very fast rate. Although the wires are only a few centimeters long, the high frequency of the transmitted signal requires that the delays introduced by the wires must not be ignored, and transmission line theory must be applied.

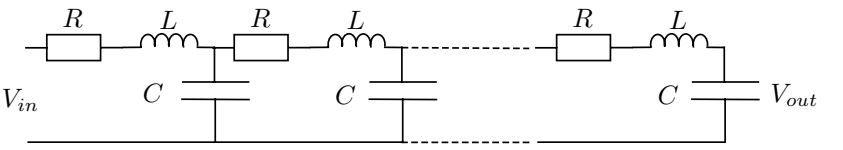

FIGURE 12.5. RLC transmission line.

Transmission lines are described as systems of partial differential equations. However, they can be approximated by lumped models, where the distributed effects of capacity, inductance, and resistance are approximated by a cascade of single capacitors, inductors, and resistors, as Fig.12.5 shows. In order to constitute a good approximation, the RLC model must be formed by several sections. As a consequence of this, the resulting model is a linear time–invariant system of ordinary differential equations with a sparse system matrix.

In [12.6], an example composed by five sections of an RLC circuit is introduced. The resistance, inductance, and capacitance values used in [12.6] can be considered realistic parameter values. The model obtained is a 10^{th}–order linear time–invariant system with the following system matrix:

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

A typical input trajectory for these digital systems is a trapezoidal wave, representing the "0" and "1" levels, as well as the rising and falling edges. Since a trapezoidal wave is a piecewise linear trajectory, we can generate it exactly using the following DEVS model:

$$\begin{aligned} &M_{10} = (X, S, Y, \delta_{\text{int}}, \delta_{\text{ext}}, \lambda, ta), \text{ where:} \\ &X = \emptyset \\ &S = \mathbb{N} \times \mathbb{R}_0^+ \\ &Y = \mathbb{R}^2 \times \mathbb{N} \\ &\delta_{\text{int}}(k, \sigma) = (\tilde{k}, T_{\tilde{k}}) \\ &\lambda(k, \sigma) = (u_{\tilde{k}}, m_{u_{\tilde{k}}}, 1) \\ &ta(k, \sigma) = \sigma \end{aligned}$$

where $\tilde{k} = (k + 1 \mod 4)$ is the next cycle index, which has 4 phases: The low state (index 0), the rising edge (1), the high state (2), and the

falling edge (3). The duration of each phase is given by the corresponding T_k value.

During the low state, the output is u_0, and during the high state, it is u_2. During these two phases, the slopes, m_{u_0} and m_{u_2}, are zero. During the rising edge, we have $u_1 = u_0$, and the slope is $m_{u_1} = (u_2 - u_0)/T_1$. Similarly, during the falling edge, we have $u_3 = u_2$ and $m_{u_3} = (u_0 - u_2)/T_3$.

The DEVS generator representing the input trajectory produces only four events in each cycle. This is an important advantage, since the presence of the input wave only adds a few extra calculations. Moreover, since the representation is exact, it does not introduce any error, i.e., we can estimate the error bound using the inequality of Eq.(12.27) instead of that of Eq.(12.31).

We performed the simulation using the parameter values $R = 80\ \Omega$, $C = 0.2\ pF$, and $L = 20\ nH$. These parameter values correspond to a transmission line of one centimeter length divided into five sections, where the line resistance, capacitance, and inductance values are $400\ \Omega/cm$, $1\ pF/cm$, and $100\ nH/cm$, respectively.

The trapezoidal input has rising and falling times of $T_1 = T_3 = 10\ psec$, whereas the durations of the low and high states are $T_0 = T_2 = 1\ nsec$. The low and high levels are $0\ V$ and $2.5\ V$, respectively.

The quantization adopted was $\Delta v = 4\ mV$ for the state variables representing voltages, and $\Delta i = 10\ \mu A$ for the state variables representing currents. This quantization, in accordance with Eq.(12.27), ensures that the maximum error is smaller than $250\ mV$ in the variable V_{out}.

The input and output trajectories are shown in Fig.12.6.

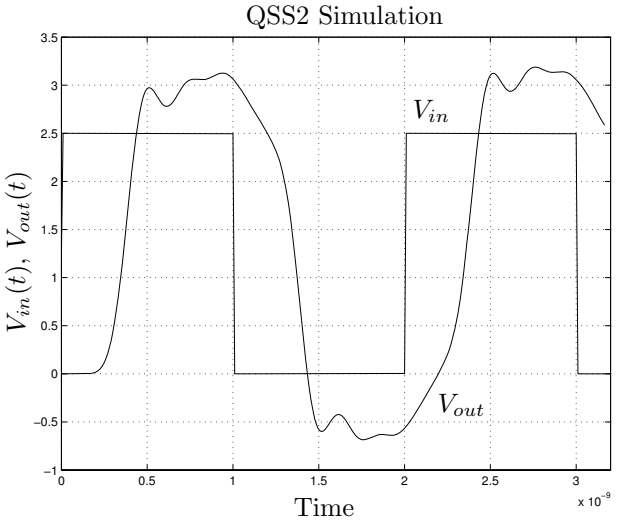

FIGURE 12.6. QSS2 simulation results in an RLC transmission line.

The simulation required a total of 2536 steps (between 198 and 319 internal transitions at each integrator) to obtain the first 3.2 $nsec$ of the system trajectories.

The experiment was repeated using a 100 times smaller quantization, which ensures a maximum error on the output voltage, V_{out}, of 2.5 mV. This new simulation was performed consuming a total of 26.883 internal transitions. Here we can see the effects of the second–order approximation: when reducing the quantum by a factor of 100, the number of events grows only by a factor of 10, i.e., it grows inverse to the square root of the quantum.

We also compared the trajectories of both simulations, and the difference in V_{out} was always less than 14.5 mV. The conclusion is that the error in the first simulation was less than 17 mV, in spite of the theoretical bound of 250 mV. The error bound formula given by Eq.(12.27) often produces highly conservative results, especially when it is applied to high order systems.

Although the number of steps in the simulations is big, it is important to remember that each step only involves scalar calculations at three integrators, the integrator that undergoes the internal transition, and the two integrators that are directly connected to its output. This is due to the sparsity of the **A**–matrix.

12.7 Algebraic Loops in QSS Methods

The circuit of Fig.12.7 can be modeled by the block diagram of Fig.12.8. The bold lines in this block diagram indicate the presence of an algebraic loop.

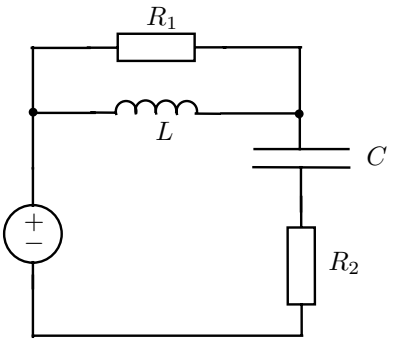

FIGURE 12.7. RLC circuit.

This algebraic loop expresses the algebraic restriction:

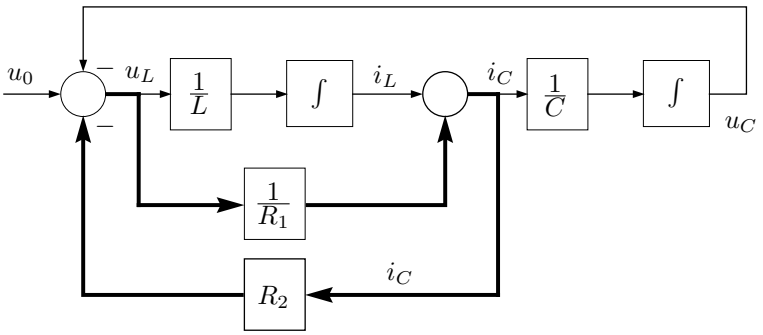

FIGURE 12.8. Block diagram representation of the RLC circuit.

$$i_C = i_L + \frac{1}{R_1}(u_0 - u_C - R_2 \cdot i_C) \tag{12.43}$$

We can implement the QSS method by transforming the integrators into DEVS models, e.g. of the M_5 class (quantized hysteretic integrators), and the static functions into their DEVS equivalent representation (DEVS models such as M_3). Then, we can couple these DEVS models according to the coupling scheme of Fig.12.8. In fact, that is precisely what we did to convert the system of Eq.(12.1) into the DEVS representation of Eq.(12.2) (cf. Fig.11.12).

As you were taught in the companion book on *Continuous System Modeling* [12.2], block diagrams are not necessarily the most convenient tool for modeling physical systems, but they are in use widely, and some of the most popular continuous system simulation tools, in particular SIMULINK [12.3], are built on this modeling paradigm. Hence also the graphical user interface of PowerDEVS was built around block diagrams.

Although the block by block translation from a block diagram representation of a continuous system to its corresponding DEVS model may result in inefficient simulation code from the point of view of computational cost, it is a very simple procedure that does not require any kind of symbolic manipulations. Thus, if we do not want to perform the translation manually, and if we do not have another automatic tool, such as Dymola [12.4], available for generating a set of equations, like those of Eq.(12.1), from a higher–level graphical representation of the system to be simulated, the block by block translation may be the most convenient way for applying a QSS method to the simulation of a continuous–time system.

However, if we apply this procedure to the block diagram of Fig.12.8, we encounter a problem. Due to the algebraic loop, the resulting DEVS model will turn out to be illegitimate. When an event arrives at the loop, it propagates forever through the static functions around the algebraic loop.

As we do not want to drastically alter the block diagram or the atomic model definitions, we tackle the problem by adding a new loop–breaking

block anywhere in the loop. For example, we can place the loop–breaking DEVS model in front of the R_2 gain element, as Fig.12.9 shows.

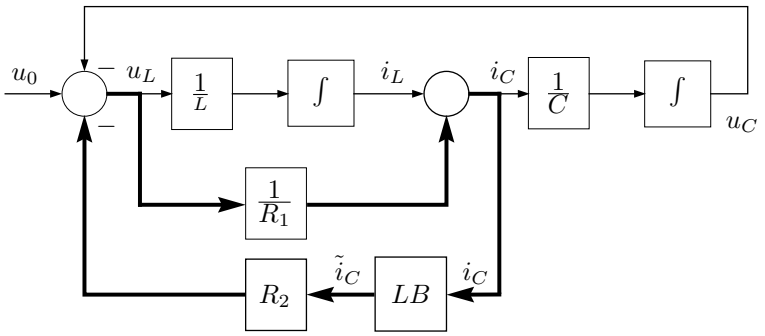

FIGURE 12.9. Addition of a loop-breaking model to the block diagram of Fig.12.8.

Now, Eq.(12.43) becomes:

$$i_C = i_L + \frac{1}{R_1}(u_0 - u_C - R_2 \cdot \tilde{i}_C) \tag{12.44}$$

where $\tilde{i}_C$ is the output value of the loop–breaking block.

Since this block is inside the loop, whenever it sends an event with a value $\tilde{i}_C$ out through its output port, it immediately (in terms of simulation time) receives an event with value i_C, calculated using Eq.(12.44), back through its input port.

If we want Eq.(12.44) to be equivalent to Eq.(12.43), we need to ensure that $\tilde{i}_C$ is equal to i_C. In other words, the value received by the loop–breaking model must be the same that it previously sent out.

Thus, the loop–breaking block could operate as follows: it sends $\tilde{i}_C$ out and receives i_C back. If the two signals differ from each other, it tries with a different $\tilde{i}_C$. Otherwise, the loop–breaking block becomes passive and doesn't send out any further events, until a new external event, caused by a transition of an integrator or an input function, arrives at the loop.

This technique should solve our problem. Notice that the proposed technique corresponds closely to the *tearing method* introduced in Chapter 7 of this book. i_C is a tearing variable. The user will need to introduce enough tearing variables (loop–breaking blocks) to break all algebraic loops in the system.

So far, we have not explained, how the value of $\tilde{i}_C$ is to be calculated. Yet, before providing an answer to this question, we need to reformulate our problem in a more general framework.

Let us call z the variable sent by the loop–breaking model. Then, when it sends an event with value z_1, it immediately receives a new event with value $h(z_1)$ calculated by the static functions.

Thus, the model should calculate a new value for z, let us call it z_2, that should satisfy:

$$h(z_2) - z_2 \triangleq g(z_2) \approx 0 \tag{12.45}$$

If $g(z_2)$ remains too large, the process must be repeated by sending a new value z_3.

Clearly, z_{i+1} must be calculated following some algorithm to find the solution of $g(z) = 0$. Taking into account that the loop–breaking block does not know the expression, and hence the derivative, of $g(z)$, a good alternative to Newton iteration is the use of the *secant method*.

Using this approach, z_{i+1} can be calculated as:

$$z_{i+1} = \frac{z_{i-1} \cdot g(z_i) - z_i \cdot g(z_{i-1})}{g(z_i) - g(z_{i-1})} \tag{12.46}$$

and since $g(z_i) = h(z_i) - z_i$, we obtain:

$$z_{i+1} = \frac{z_{i-1} \cdot h(z_i) - z_i \cdot h(z_{i-1})}{h(z_i) - h(z_{i-1}) + z_{i-1} - z_i} \tag{12.47}$$

Based on these ideas, the loop–breaking DEVS model can be represented as follows:

$$\begin{aligned}
&M_{11} = (X, Y, S, \delta_{\text{int}}, \delta_{\text{ext}}, \lambda, ta), \text{ where} \\
&X = \mathbb{R} \times \mathbb{N} \\
&Y = \mathbb{R} \times \mathbb{N} \\
&S = \mathbb{R}^3 \times \mathbb{R}_0^+ \\
&\delta_{\text{ext}}(s, e, x) = \delta_{\text{ext}}(z_1, z_2, h_1, \sigma, e, x_v, p) = \tilde{s} \\
&\delta_{\text{int}}(s) = \delta_{\text{int}}(z_1, z_2, h_1, \sigma) = (z_1, z_2, h_1, \infty) \\
&\lambda(s) = \lambda(z_1, z_2, h_1, \sigma) = (z_2, 1) \\
&ta(s) = ta(z_1, z_2, h_1, \sigma) = \sigma
\end{aligned}$$

where:

$$\tilde{s} = \begin{cases} (z_1, z_2, h_1, \infty) & \text{if} \quad |x_v - z_2| < tol \\ (z_2, \tilde{z}, x_v, 0) & \text{otherwise} \end{cases}$$

with:

$$\tilde{z} = \frac{z_1 \cdot x_v - z_2 \cdot h_1}{x_v - h_1 + z_1 - z_2} \tag{12.48}$$

The parameter *tol* represents the largest absolute error that we allow between z and h. Equation (12.48) is the result of applying the secant method to approximate $g(z) = 0$, where $g(z)$ is defined in accordance with Eq.(12.45). We can change the iteration algorithm by modifying Eq.(12.48).

A corresponding PowerDEVS model can be coded as follows:

ATOMIC MODEL LOOP-BREAK1
State Variables and Parameters:
float $z1, z2, h1, sigma$; //states
float y; //output
float tol, inf;

Init Function:
va_list $parameters$;
va_start($parameters$, t);
tol = va_arg($parameters$, double);
$inf = 1e10$;
$sigma = inf$;
$z1 = 0$;
$z2 = 0$;
$h1 = 0$;
$y = 0$;
Time Advance Function:
return $sigma$;

Internal Transition Function:
$sigma = inf$;

External Transition Function:
float xv;
xv =*(float*)($x.value$);
if (fabs($xv - z2$) < tol) {
$sigma = inf$;
}
else{
$z3 = z2$;
if (($z1 == 0$) && ($z2 == 0$)){
$z2 = xv$; //initial guess
}
else {
$z2 = (z1 * xv - z2 * h1)/(xv - h1 + z1 - z2)$;
};
$z1 = z3$;
$h1 = xv$;
$sigma = 0$;
};

Output Function:
$y = z2$;
return Event(&y, 0);

For the circuit example of Fig.12.7, we built a coupled DEVS model in accordance with Fig.12.9 and simulated it during 30 seconds using the QSS algorithm. We used the parameter values $R_1 = R_2 = L = C = 1$, and u_0 was chosen as a unit step. The quantum and hysteresis adopted were 0.01 in both state variables, and the error tolerance tol was chosen equal to 0.001.

The simulation, the results of which are shown in Fig.12.10, was completed after 118 and 72 internal transitions at each quantized integrator and a total of 377 iterations at the loop–breaking DEVS model.

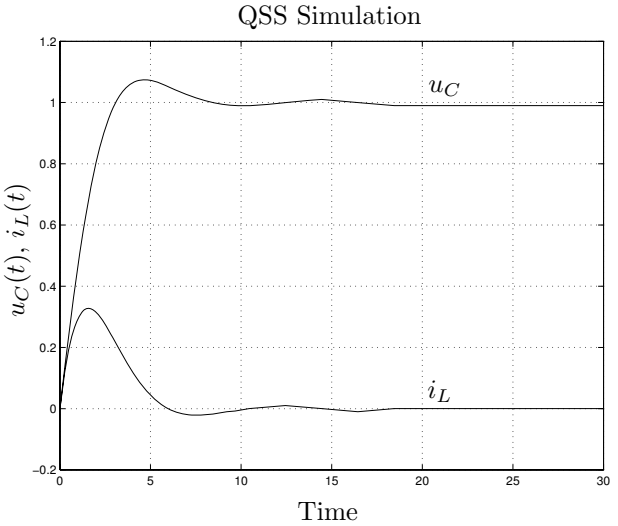

FIGURE 12.10. QSS simulation of the RLC circuit using a loop–breaking DEVS.

In this case, due to the linearity of the system, the secant method arrives at the exact solution of $g(z) = 0$ after only two iterations. This explains why the total number of iterations at the loop–breaking model was twice the total number of steps at both quantized integrators.

In a more general nonlinear case, more iterations per transition would probably be needed, and we would have to discuss the convergence criteria associated with the chosen iteration method.

An interesting observation is that the effect of the error in the calculation of i_C can be seen as an additional perturbation. If we can ensure that this error remains bounded, the perturbation is also bounded and can be seen as something equivalent to having a bigger quantum that only affects the error bound but does not modify the stability properties.

¿From the discussion in this section, the reader might reach the conclusion that the QSS methods deal with algebraic loops in the same, or at least a very similar, way as the discrete–time methods introduced in earlier chapters. However, such a conclusion would be totally wrong.

Almost all of the discrete–time methods presented earlier in this book are centralized integration schemes that require the iteration of *all* algebraic loops during *every* integration step, or even more accurately, during each function evaluation.

In contrast, the QSS methods operate in a completely asynchronous fashion. An algebraic loop will only be iterated upon when it gets triggered by

a transition occurring either in a quantized integrator or in an input function. Due to the inherent sparsity property of large–scale physical systems, plenty of transitions may take place in a larger model that do not affect any of the algebraic loops at all. These transitions can proceed without ever triggering an iteration on any of the loops.

Notice further that the discussion, presented in this section, focused on the QSS method, rather than the QSS2 method. In the QSS2 method, each loop variable carries with it a slope variable. Thus, the loop–breaking block will need to iterate on two variables simultaneously: the tearing variable, z, and its associated slope variable, m_z.

12.8 DAE Simulation with QSS Methods

In the previous section, we worked with a particular DAE system. We saw that, by adding a loop–breaking block, we can still use the QSS simulation method in the presence of an algebraic loop.

Although this technique can be easily implemented, it does not constitute a general solution yet. Moreover, the method may turn out to be fairy inefficient, since the iteration process involves a traffic of events across all blocks that constitute the loop.

This section is aimed at introducing a more general case and a more efficient solution. As usual, we shall start by analyzing an example.

Figure 12.11 shows the transmission line model of Fig.12.5, modified by the addition of a load.

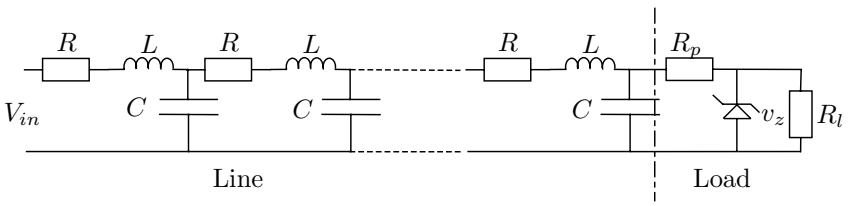

FIGURE 12.11. RLC transmission line with surge voltage protection.

The load is composed of a resistor R_l, possibly representing the gate of some electronic component, and a surge protection circuit formed by a Zener diode and a resistor R_p. The Zener diode satisfies the following nonlinear relationship between its voltage and its current:

$$i_z = \frac{I_0}{1 - (v_z/v_{br})^m} \tag{12.49}$$

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

If the transmission line is divided into five sections, as we did earlier, the following equations are obtained:

$$\begin{aligned}
\frac{di_1}{dt} &= \frac{1}{L}\cdot v_{in} - \frac{R}{L}\cdot i_1 - \frac{1}{L}\cdot u_1 \\
\frac{du_1}{dt} &= \frac{1}{C}\cdot i_1 - \frac{1}{C}\cdot i_2 \\
\frac{di_2}{dt} &= \frac{1}{L}\cdot u_1 - \frac{R}{L}\cdot i_2 - \frac{1}{L}\cdot u_2 \\
\frac{du_2}{dt} &= \frac{1}{C}\cdot i_2 - \frac{1}{C}\cdot i_3 \\
&\vdots \\
\frac{di_5}{dt} &= \frac{1}{L}\cdot u_4 - \frac{R}{L}\cdot i_5 - \frac{1}{L}\cdot u_5 \\
\frac{du_5}{dt} &= \frac{1}{C}\cdot i_5 - \frac{1}{R_pC}\cdot (u_5 - v_z)
\end{aligned} \tag{12.50}$$

Here, the state variables, u_j and i_j, represent the voltage and current in the capacitors and inductors of the transmission line, respectively, and the output voltage, v_z, is an algebraic variable that satisfies the equation:

$$\frac{1}{R_p}\cdot u_5 - \left(\frac{1}{R_p} + \frac{1}{R_l}\right)\cdot v_z - \frac{I_0}{1-(v_z/v_{br})^m} = 0 \tag{12.51}$$

Thus, we are confronted with a DAE that cannot be converted into an ODE by symbolic manipulation, and the simulation using any discrete–time method will have to iterate on Eq.(12.51) during each step, in order to solve for the unknown variable v_z.

If we want to apply the QSS method to the system defined by Eq.(12.50) and Eq.(12.51), we can try to proceed as before, replacing the state variables i_j and u_j by their quantized versions.

In order to be able to use our standard notation, we define $x_{2j-1} \triangleq i_j$ and $x_{2j} \triangleq u_j$ for $j = 1, \ldots, 5$. As before, we shall refer to the quantized version of variable x_j as q_j.

Then, the use of QSS transforms Eq.(12.50) to:

$$\begin{aligned}
\frac{dx_1}{dt} &= \frac{1}{L}\cdot v_{in} - \frac{R}{L}\cdot q_1 - \frac{1}{L}\cdot q_2 \\
\frac{dx_2}{dt} &= \frac{1}{C}\cdot q_1 - \frac{1}{C}\cdot q_3 \\
\frac{dx_3}{dt} &= \frac{1}{L}\cdot q_2 - \frac{R}{L}\cdot q_3 - \frac{1}{L}\cdot q_4 \\
\frac{dx_4}{dt} &= \frac{1}{C}\cdot q_3 - \frac{1}{C}\cdot q_5 \\
&\vdots \\
\frac{dx_9}{dt} &= \frac{1}{L}\cdot q_8 - \frac{R}{L}\cdot q_9 - \frac{1}{L}\cdot q_{10} \\
\frac{dx_{10}}{dt} &= \frac{1}{C}\cdot q_9 - \frac{1}{R_pC}\cdot (q_{10} - v_z)
\end{aligned} \tag{12.52}$$

and the implicit part of the system, i.e., Eq.(12.51), turns into:

$$\frac{1}{R_p} \cdot q_{10} - \left(\frac{1}{R_p} + \frac{1}{R_l} \right) \cdot v_z - \frac{I_0}{1 - (v_z / v_{br})^m} = 0 \tag{12.53}$$

Notice that Eq.(12.52) looks like a quantized state system, except for the presence of v_z. However, the variable v_z is algebraically coupled to q_{10}. Thus, each time that q_{10} undergoes a transition, we iterate on Eq.(12.53) to find the new value of v_z, and then use that value in Eq.(12.52), however and contrary to the previously proposed solution involving a loop–breaking block, this iteration occurs entirely within a single DEVS model, and therefore doesn't involve events being passed around between different blocks in a loop.

We can build a block diagram corresponding to Eq.(12.52) and Eq.(12.53) as follows:

- We start by representing system Eq.(12.52) with quantized integrators and static functions, treating v_z as if it were an external input.
- We then add a new atomic block that computes v_z as a function of q_{10}. Consequently, this block has q_{10} as an input and v_z as an output.

The latter block will be in charge of iteration to determine a new value of v_z, each time q_{10} changes. Ignoring round–off errors and assuming that the iteration block computes the value of v_z correctly, the resulting coupled DEVS model will exactly simulate the system defined by Eq.(12.52) and Eq.(12.53).

Notice that the iteration block is only activated by a change in q_{10}. In all other steps, i.e., when either one of the other nine quantized variables changes, or when the input changes, no iteration has to be performed, and the QSS models acts like in the case of an explicit ODE model. Clearly, QSS is still able to exploit the sparsity inherent in DAE models of large–scale physical systems.

Our original system, defined by Eq.(12.50) and Eq.(12.51), is a particular case of the implicit model introduced in Chapter 8:

$$\mathbf{f}(\mathbf{x}, \dot{\mathbf{x}}, \mathbf{u}, t) = 0 \tag{12.54}$$

In that chapter, we studied methods for simulating this model without transforming it to explicit ODE form first.

Let us check whether we can apply similar ideas to the situation of a QSS simulation involving algebraic loops.

For simplicity, we shall only considering the time–invariant case, although the explicit inclusion of time is not problematic, especially in the context of performing a QSS2 simulation.

We can rewrite Eq.(12.54) as follows:

$$\tilde{\mathbf{f}}(\dot{\mathbf{x}}_\mathbf{a}, \mathbf{x}_\mathbf{a}, \mathbf{u}) = 0 \tag{12.55}$$

As before, we call the state vector of the original system $\mathbf{x_a}$ to distinguish it from the quantized state vector.

Proceeding as we did before, we can modify Eq.(12.55) as follows:

$$\tilde{\mathbf{f}}(\dot{\mathbf{x}}, \mathbf{q}, \mathbf{u}) = 0 \tag{12.56}$$

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

Now, we can apply Newton iteration to Eq.(12.56) to solve for $\dot{\mathbf{x}}$. We shall assume that the perturbation index of Eq.(12.56) is 1. Otherwise, we shall apply the Pantelides algorithm first, in order to reduce the perturbation index of the DAE system to 1.

Once we have obtained numerical values for the state derivatives, they can be sent to the quantized integrators that perform the rest of the job, i.e., calculate the quantized variable trajectories. Each time a quantized variable changes, a new iteration process must be performed in order to recalculate $\dot{\mathbf{x}}$.

However in our previous example, we only needed to iterate, when q_{10} changed. For some reason, we lost the capability of exploiting sparsity in the compact notation of Eq.(12.56).

As we still want to be able to exploit sparsity, we need to rewrite Eq.(12.55) as follows:

$$\dot{\mathbf{x}}_\mathbf{a} = \mathbf{f}(\mathbf{x_a}, \mathbf{u}, \mathbf{z_a}) \tag{12.57a}$$

$$0 = \mathbf{g}(\mathbf{x}_{\mathbf{a_r}}, \mathbf{u_r}, \mathbf{z_a}) \tag{12.57b}$$

where $\mathbf{z_a}$ is a vector of tearing variables with dimension equal to or less than n. The vectors $\mathbf{x}_{\mathbf{a_r}}$ and $\mathbf{u_r}$ are reduced versions of $\mathbf{x_a}$ and $\mathbf{u}$, respectively.

A straightforward –but useless– way of transforming the system represented by Eq.(12.55) into the system of Eq.(12.57) is by defining $\mathbf{z_a} \triangleq \dot{\mathbf{x}}_\mathbf{a}$, which yields $\mathbf{x}_{\mathbf{a_r}} = \mathbf{x_a}$, $\mathbf{u_r} = \mathbf{u}$, and $\mathbf{g} = \mathbf{f} - \mathbf{z_a}$.

However in many cases, as in the case of the transmission line example, the dimensions of $\mathbf{x}_{\mathbf{a_r}}$ and $\mathbf{u_r}$ can be effectively reduced.

Equation (12.57b) expresses the fact that some state and input variables may not act directly on the algebraic loops.

Then, the use of the QSS methods transforms Eq.(12.57) into:

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{q}, \mathbf{u}, \mathbf{z}) \tag{12.58a}$$

$$0 = \mathbf{g}(\mathbf{q_r}, \mathbf{u_r}, \mathbf{z}) \tag{12.58b}$$

and now, an iteration will only be performed, when components of either $\mathbf{q_r}$ or $\mathbf{u_r}$ change.

The use of QSS and QSS2 methods with DAE systems is quite similar. In order to simplify the derivation, we shall only consider the first–order method for now. We shall add remarks relating to the QSS2 method later.

When Eq.(12.58b) defines the value of $\mathbf{z}$, we can see that the system of Eq.(12.58) defines something that behaves like a QSS. In fact, we can easily prove that the state and quantized variable trajectories correspond to a QSS (i.e., they are piecewise linear and piecewise constant, respectively). Moreover, the auxiliary variables $\mathbf{z}$ are also piecewise constant.

What still needs to be explained now is how an implicitly defined QSS can be translated into a DEVS model.

It is clear that Eq.(12.58a) can be represented by quantized integrators and static functions, as we did in Chapter 11. The only difference here is the presence of the auxiliary variables $\mathbf{z}$, acting as inputs just like $\mathbf{u}$. However, whereas the inputs $\mathbf{u}$ arrive from signal generators of the M_6 or M_7 class, the auxiliary variables must be calculated by solving the constraints of Eq.(12.58b).

Thus, a new DEVS model must be created. This DEVS model receives events with the values of either $\mathbf{q_r}$ or $\mathbf{u_r}$, and calculates a new value of $\mathbf{z}$ in return that it then sends out through its output port.

A DEVS model that solves a general implicit equation, such as:

$$\mathbf{g}(\mathbf{v}, \mathbf{z}) = \mathbf{g}(v_1, \ldots, v_m, z_1, \ldots, z_k) = 0 \tag{12.59}$$

can be written as follows:

$$\begin{aligned}
&M_{12} = (X, Y, S, \delta_{\text{int}}, \delta_{\text{ext}}, \lambda, ta), \text{ where} \\
&X = \mathbb{R} \times \mathbb{N} \\
&Y = \mathbb{R}^k \times \mathbb{N} \\
&S = \mathbb{R}^{m+k} \times \mathbb{R}_0^+ \\
&\delta_{\text{ext}}(s, e, x) = \delta_{\text{ext}}(\mathbf{v}, \mathbf{z}, \sigma, e, x_v, p) = (\tilde{\mathbf{v}}, \mathbf{h}(\tilde{\mathbf{v}}, \mathbf{z}), 0) \\
&\delta_{\text{int}}(s) = \delta_{\text{int}}(\mathbf{v}, \mathbf{z}, \sigma) = (\mathbf{v}, \mathbf{z}, \infty) \\
&\lambda(s) = \lambda(\mathbf{v}, \mathbf{z}, \sigma) = (\mathbf{z}, 1) \\
&ta(s) = ta(\mathbf{v}, \mathbf{z}, \sigma) = \sigma
\end{aligned}$$

where:

$$\tilde{\mathbf{v}} = (\tilde{v}_1, \ldots, \tilde{v}_m)^T; \;\; \tilde{v}_i = \begin{cases} x_v & \text{if} \quad p = i \\ v_i & \quad\;\; \text{otherwise} \end{cases}$$

and the function $\mathbf{h}(\tilde{\mathbf{v}}, \mathbf{z})$ returns the result of applying a Newton iteration or some other type of iteration to find the solution of Eq.(12.59) using an initial value $\mathbf{z}$.

When the size of $\mathbf{z}$ (i.e., k) is greater than 1, the output events of model M_{12} contain a vector. Thus, they cannot be sent to static functions such as M_3. However, we can use a DEVS *demultiplexer* (cf. Hw.[H11.3]), in order to tackle this problem.

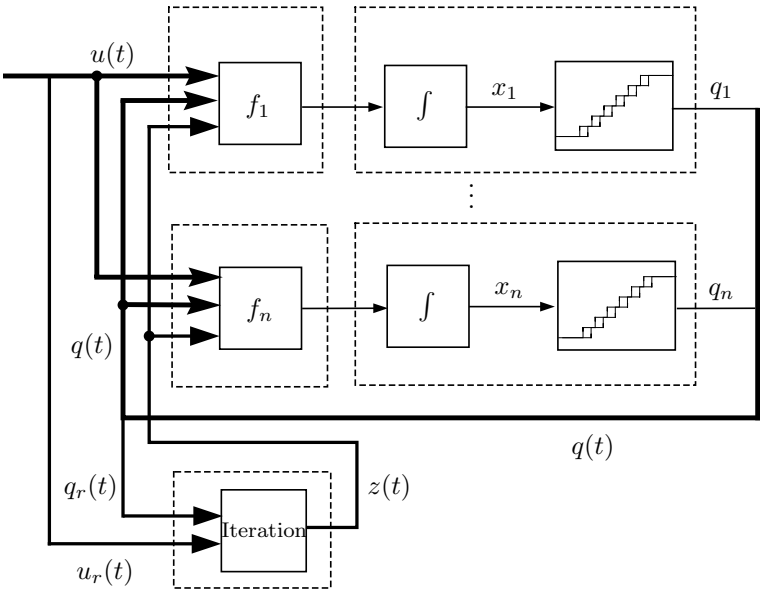

FIGURE 12.12. Coupling scheme for the QSS simulation of Eq.(12.57).

Figure 12.12 shows the new coupling scheme with the addition of a new DEVS model that calculates $\mathbf{z}$.

When it comes to the QSS2 method, the same ideas can be applied, and analogous DEVS models to M_{12} can be built. The only difficulty is that now the trajectory slopes must be taken into account. This is not a problem in linear systems, but it becomes a bit more complicated in nonlinear systems, where estimations of the partial derivatives should be used. However, this problem has already been solved, and a DEVS model replacing M_{12} for the QSS2 method is provided in [12.14].

PowerDEVS also offers a nonlinear implicit model that solves a constraint of the form $g(\mathbf{v}, z) = 0$, calculating both the value and slope of z. The symbolic expression g is a string parameter that can be modified by double clicking on the implicit model icon.

We are now ready to return to the transmission line example. Using the ideas expressed above, we modified the simulation of page 581 by adding a PowerDEVS model like the one introduced in the previous paragraph that solves Eq.(12.51), taking into account the slope.

Letting $R_l = 100\ M\Omega$, $I_0 = 0.1\ \mu A$, $v_{br} = 2.5\ V$, and $m = 4$ without modifying the remaining parameters, we obtained the results shown in Fig.12.13.

The first $3.2ns$ of the simulation were completed after 2640 steps (between 200 and 316 steps at each integrator). The implicit model performed a total of 485 iterations using the secant method. The reason for this is that the quantized integrator that calculates u_5 only performed 200 inter-

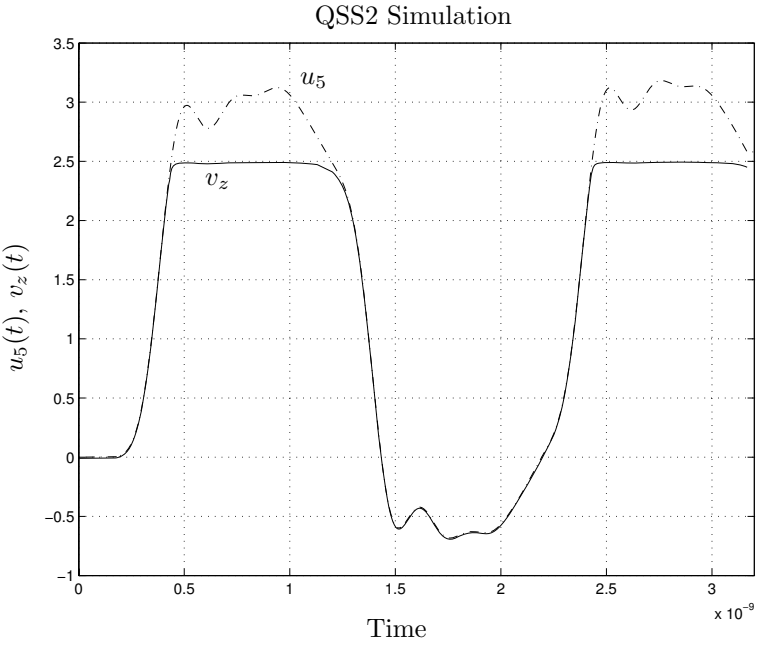

FIGURE 12.13. QSS2 simulation results in an RLC transmission line with surge protection.

nal transitions, and therefore, the implicit model received only 200 external events. The secant method needed between two and three iterations to find the solution of Eq.(12.51) with the required tolerance of $tol = 1 \times 10^{-8}$, which explains the fact that the total number of iterations was 485.

The advantages of the QSS2 method are evident in this example. In a discrete–time algorithm, the secant method would have been invoked at every integration step, whereas the QSS2 only called it after changes in u_5, i.e., about once every 13 steps. Thus, the presence of the implicit equation only adds a few calculations that do not affect significantly the total number of computations.

Returning once more to the issue of higher–index DAEs, we already mentioned that such DAEs can be simulated by reducing their perturbation index to index 1 first using the Pantelides algorithm, introduced in Chapter 7 of this book, and then applying the QSS approach, presented in this section, to the resulting index–1 DAE system. However, this may not be the only way to tackling higher–index DAE problems using a QSS method. An alternate solution was proposed in [12.9]. There, it is shown that applying the QSS method to a higher–index DAE results in a model that switches between two (or more) ODE systems. Thus, the simulation can be performed applying the same principles that rule the simulation of variable structure systems.

We shall not explore this idea any further here, since it was developed in the context of bond graph modeling only. Although we believe that this technique may also be applicable to general DAE systems, such a generalization would require more research.

12.9 Discontinuity Handling

In Chapter 9, we studied the simulation of discontinuous systems using discrete–time approaches. We remarked that numerical ODE solvers are based on Taylor–Series expansions, and therefore, their trajectories are approximated by polynomials or rational functions. Since neither of these functions exhibit discontinuities, the solvers will invariably be in trouble, when asked to integrate across discontinuous functions.

However, the same restriction does not hold in the case of the QSS and QSS2 methods. Here, the discontinuities in the input and quantized variable trajectories are in fact responsible for time advance. Simulation steps are only being calculated, when discontinuities are found in those trajectories.

Using discrete–time methods, the time instances of discontinuities had to be determined precisely, as we were unable to integrate across discontinuities accurately. Consequently, the primary difficulties in dealing with discontinuous functions using discrete–time methods are related to accurately detecting the time of occurrence of a discontinuity, and to designing suitable integration step–size control algorithms around them.

Two different kinds of discontinuities were distinguished in our analysis: *time events*, that could be scheduled ahead of time, as their time of occurrence was known in advance, and *state events*, that were specified indirectly by means of some threshold crossing function, which required the use of an iteration algorithm (a root solver algorithm) for locating them accurately in time.

As we shall discover soon, all of these problems disappear with the use of QSS and QSS2 methods.

The simulation of hybrid systems using QSS and QSS2 was studied in [12.15] and, as the reader may already have anticipated, the most important advantages of the discrete event approximations to continuous system simulation are to be found in these kinds of applications.

Let us begin by analyzing a simple example. The inverter circuit shown in Fig.12.14 is a device typically used to power electrical machines that are being operated off the grid.

The set of switches can assume two different positions. In the first position, switches 1 and 4 are closed, and consequently, the load receives a positive voltage. In the second position, switches 2 and 3 are closed, and the load sees a negative voltage accordingly.

The system can be represented by the following differential equation:

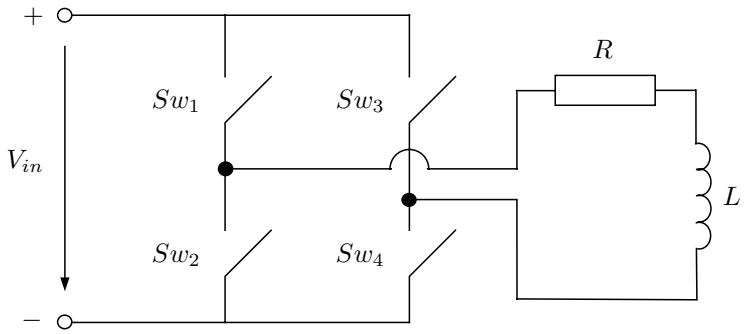

FIGURE 12.14. DC–AC full bridge inverter circuit.

$$\frac{di_L}{dt} = -\frac{R}{L} \cdot i_L + V_{in} \cdot s_w(t) \tag{12.60}$$

where s_w assumes a value of either $+1$ or -1, depending on the position that the four switches are operating in.

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

The PWM signal is obtained by comparing a triangular wave, the so–called carrier, with a modulating sinusoidal reference signal. The sign of the voltage to be applied, $+V_{in}$ or $-V_{in}$, and thereby the corresponding switch position, is determined by the sign of the difference between these two signals. Figure 12.15 illustrates this concept.

The difference between the carrier and modulation signal could be used as a zero–crossing function of a state–event description for the control strategy. However, since both the carrier signal and the modulating signal are simple static functions, the time of the next intersection between these two signals can easily be computed ahead of time, allowing an implementation of the PWM control strategy by means of time events.

The system can be thought of as the coupling of a continuous submodel, described by Eq.(12.60), and a discrete submodel that manages a sequence of time events for determining the correct value of s_w.

The discrete submodel can easily be represented as a DEVS model. It is a simple DEVS generator model of the M_6 class, introduced on page 565. The output alternates between $+1$ and -1, and the time elapse between commutations can be numerically computed ahead of time.

The continuous submodel can be approximated, using either the QSS or the QSS2 method, by transforming Eq.(12.60) to:

$$\frac{di_L}{dt} = -\frac{R}{L} \cdot q_{i_L} + V_{in} \cdot s_w(t) \tag{12.61}$$

where q_{i_L} is the quantized state associated with variable i_L.

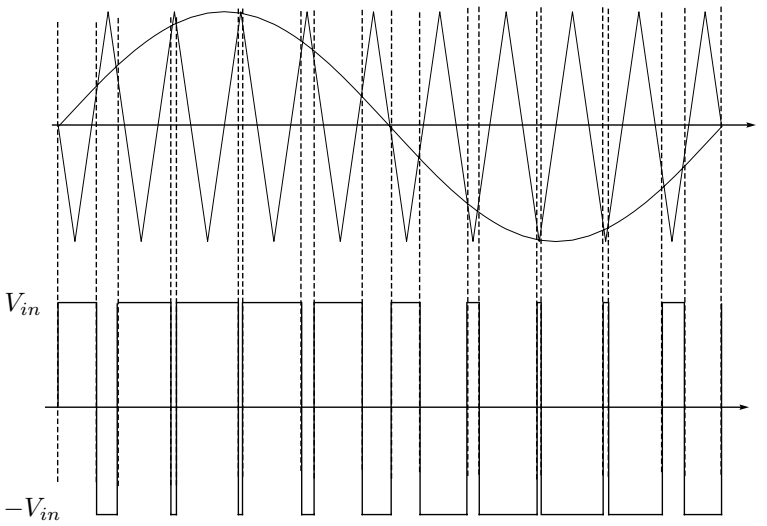

FIGURE 12.15. Pulse width modulation.

If we consider the last equation only, and if nobody tells us that $s_w(t)$ originates at a discrete submodel, we could think that Eq.(12.61) corresponds to the QSS or QSS2 approximation of a continuous model with an input trajectory $s_w(t)$.

Indeed, the QSS methods effectively treat $s_w(t)$ as an input without regard for where that signal originates. Since the changes in s_w are treated asynchronously, the QSS integration has no problems with accommodating this signal.

Thus, ignoring round–off errors, and assuming that the DEVS model generating $s_w(t)$ works properly, the system of Eq.(12.61) will be simulated exactly.

Moreover, as Eq.(12.60) is linear and $s_w(t)$ is piecewise constant, i.e., we can use the exact input trajectory, we can apply Eq.(12.27) to calculate the global error bound. Consequently, the error of i_L is bounded by the quantum used.

To corroborate these remarks, we simulated the system with the QSS2 method using PowerDEVS.

We first built a new block to provoke the correct sequence s_w. To this end, we assumed that the triangular carrier has a frequency of 1.6 kHz, and that the modulating sinusoidal signal has the same amplitude, but a frequency of 50 Hz.

Thus, the number of events per modulating cycle is $2 \cdot 1600/50 = 64$, which is sufficient for producing a fairly smooth sinusoidal current.

The PowerDEVS model is shown in Fig.12.16.

Employing the parameter values $R = 0.6\ \Omega$, $L = 100\ mH$, and $V_{in} = 300\ V$, the simulation starting from $i_L = 0$ and using a quantization of

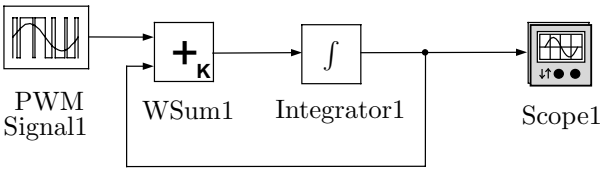

FIGURE 12.16. PowerDEVS model of the inverter circuit.

$\Delta i_L = 0.01\ A$ produced the results shown in Figs.12.17–12.18.

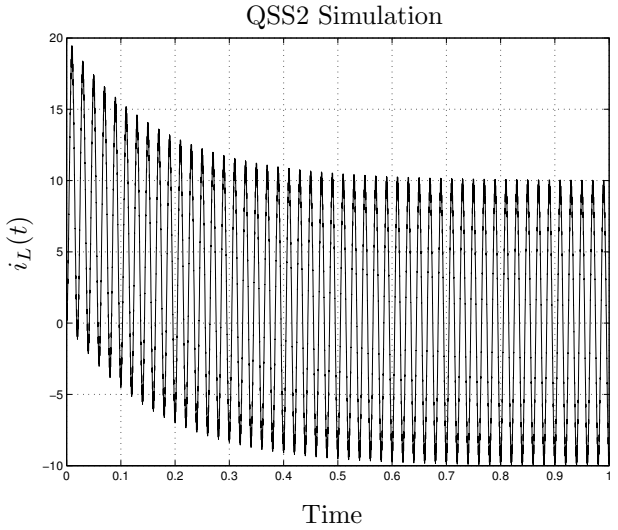

FIGURE 12.17. Load current with pulse width modulation.

The final time of the simulation chosen was 1 *sec*, and thus, the number of simulated cycles was 50. The discrete system underwent a total of 3200 changes in the switch positions, i.e., 3200 time events took place.

In spite of this large number of time events, the simulation was completed after only 3100 internal transitions at the second order quantized integrator. Thus, the total number of simulation steps was $3100 + 3200 = 6300$.

As mentioned before, the error is bounded by the quantum. The trajectories depicted in Figs.12.17–12.18 have an error that is no larger than 0.01 A at any instant of time, corresponding to roughly 0.1% of the oscillation amplitude.

The simulation of the same system with discrete–time methods, even using the most appropriate event handling techniques, requires many more integration steps, and there is little that we can say about the global error bound.

This example shows that time events are treated in a very natural way by QSS methods. The only new thing that we had to do is to express the discrete subsystem as a DEVS model and connect its output to the

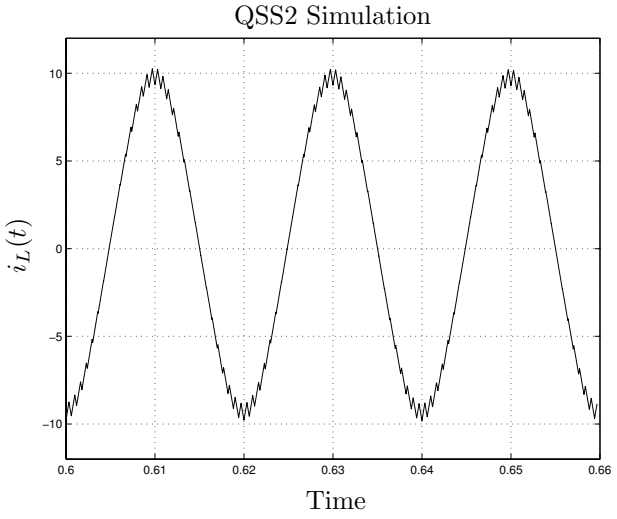

FIGURE 12.18. Steady–state behavior of load current.

continuous submodel.

Let us now discuss what happens in the presence of state events. To this end, we shall modify our previous example.

Due to failures (a short circuit in the load for instance) or during transients, the load current of the circuit in Fig.12.14 might take on values that are too large and thus could damage the components. To prevent this from happening, such circuits are usually protected.

A simple and cheap surge protection scheme consists in measuring the load current and, when it surpasses the allowed value, to close switches 2 and 4, so that the voltage applied to the load becomes 0. Then, when the current has returned to a level below the maximum allowed value, the circuit resumes its normal operation.

Applying this strategy to our example converts Eq.(12.60) to:

$$\frac{di_L}{dt} = -\frac{R}{L} \cdot i_L + V_{in} \cdot \tilde{s}_w(t) \tag{12.62}$$

where:

$$\tilde{s}_w(t) = \begin{cases} s_w(t) & \text{if } i_L(t) < i_M \\ 0 & \text{otherwise} \end{cases} \tag{12.63}$$

and i_M is the maximum allowed current.

In a real application, we should add hysteresis to the protection, and we should also prevent the condition $i_L < -i_M$. However, as this examples is introduced for illustrative purposes only, we shall limit our discussion to the simplified version.

Variable $\tilde{s}_w$ depends on the value of the state i_L. As the value of i_L cannot be computed analytically, the surge protection logic must be implemented

by means of state events. A state event occurs in the model, whenever $i_L = i_M$.

As we saw in Chapter 9, discrete–time algorithms must iterate to find the exact moment when $i_L = i_M$. We shall see that there is no need for iterating on state events, when employing a QSS method.

We can proceed as we did earlier, i.e., we change Eq.(12.62) to:

$$\frac{di_L}{dt} = -\frac{R}{L} \cdot q_{i_L} + V_{in} \cdot \tilde{s}_w(t) \tag{12.64}$$

and build a DEVS model of the discrete subsystem that calculates $\tilde{s}_w(t)$.

Since $\tilde{s}_w(t)$ depends on i_L, the discrete subsystem must receive from the continuous subsystem information concerning the value of $i_L(t)$.

At this point, we have two choices: we can either use the quantized variable q_{i_L} instead of i_L in Eq.(12.63), or we can use the true state variable.

¿From a formal point of view, using i_L appears as the correct choice. Indeed, this is the idea followed in [12.15].

However from a practical point of view, it is much simpler to use q_{i_L}. Although we can obtain the successive values of the state variables (we already studied the problem of output interpolation), the quantized variables are directly seen at the output of the quantized integrators.

Moreover, the state and quantized variables never differ from each other by more than the quantum ΔQ. Thus, the replacement will not introduce a large error in general. If i_M is a hard limit, it would suffice to modify the state condition to $i_L = i_M - \Delta Q$ to ensure that the current i_L will never surpass the value of i_M.

Thus, for the moment, we shall use q_{i_L} instead of i_L for the discrete subsystem. We shall revisit this issue later on in the chapter.

The discrete subsystem can be formed by two atomic models. The first block generates $s_w(t)$ as before (the block PWM signal of Fig.12.16), and the second block sends events out with either the value s_w or 0 depending on the value of q_{i_L}. This second block, in order to decide the value to be sent out, must receive the previously calculated value of s_w and the successive values of q_{i_L} through its input ports.

If we are using the first–order accurate QSS method, q_{i_L} is piecewise constant. Provided that a quantization level equal to i_M exists, the detection of the condition $q_{i_L} = i_M$ is straightforward.

If such a level does not exists, we will not be able to detect the exact condition, since it will never occur. However, we can easily detect the crossings, because, when they occur, we have that $q_{i_L}(t_{k^-}) < i_M$ and $q_{i_L}(t_k) > i_M$. Thus, the time of occurrence of the state event can be computed exactly.

In the case of the QSS2 method, the trajectory q_{i_L} will be piecewise linear. Thus, we can easily compute the precise instant in time, when q_{i_L} crosses i_M, by solving a linear equation. The time to the next crossing can be exactly calculated as:

$$\sigma = \begin{cases} (q_{i_L} - i_M)/m_q & \text{if } m_q \neq 0 \text{ and } (q_{i_L} - i_M)/m_q > 0 \\ \infty & \text{otherwise} \end{cases} \tag{12.65}$$

However, as q_{i_L} is discontinuous (cf. Fig.12.4), it can happen that q_{i_L} jumps over the event condition, and we detect a situation where $q_{i_L}(t_{k^-}) < i_M$ and $q_{i_L}(t_k) > i_M$. In this case, the time of occurrence of the crossing can again be detected exactly.

PowerDEVS offers several blocks that detect and handle discontinuities in accordance with these concepts. For our example, we used a *Switch* block, that predicts the intersection of a piecewise parabolic trajectory with a given fixed threshold value. Then, when this trajectory, entering the block through the second input port, is greater than the threshold value, the output sends out the trajectory received through the first input port. Otherwise, it sends out the trajectory received through the third input port.

The PowerDEVS model of the circuit with the surge protection is shown in Fig.12.19. A *Delay* block was added, modeling the fact that the switches don't react instantaneously.

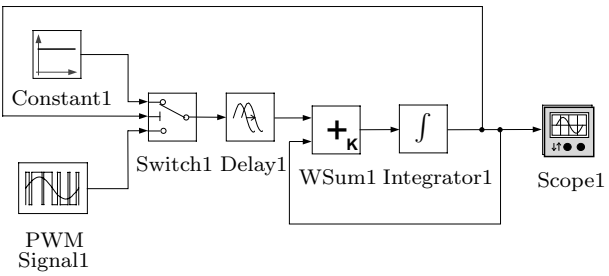

FIGURE 12.19. PowerDEVS model of inverter circuit with surge protection.

Had we not included the delay, the resulting model would have been illegitimate. The reason is that when the condition $i_L = i_M$ occurs, $\tilde{s}_w$ is set to 0, and consequently, the slope in i_L becomes negative. Thus, $\tilde{s}_w$ changes to 1 again, and a cyclic behavior is obtained without time advance.

As mentioned earlier in this chapter, the illegitimacy issue could also have been avoided by adding hysteretic behavior to the crossing condition. Such an approach might in fact be preferable to the delay solution, but the solution with the delay block suffices for illustrating, how PowerDEVS can deal with state events.

We simulated the model with the QSS2 method using the same parameters as before, while letting $i_M = 11\ A$, and choosing a delay of $\Delta T = 1 \times 10^{-6}\ sec$.

As in the previous example, the PWM block generated 3200 time events, but the switch now sent out 3288 events to the continuous subsystem. At

least 88 state events must consequently have occurred during the simulation. The quantized integrator now performed 3188 steps.

Figures 12.20–12.21 display the simulation results.

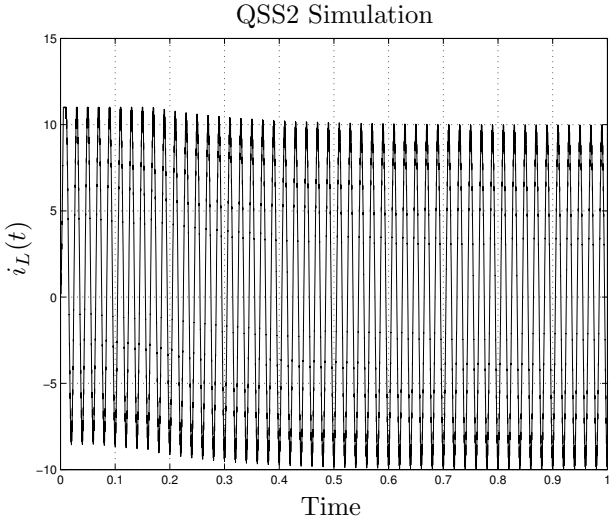

FIGURE 12.20. Load current with surge protection.

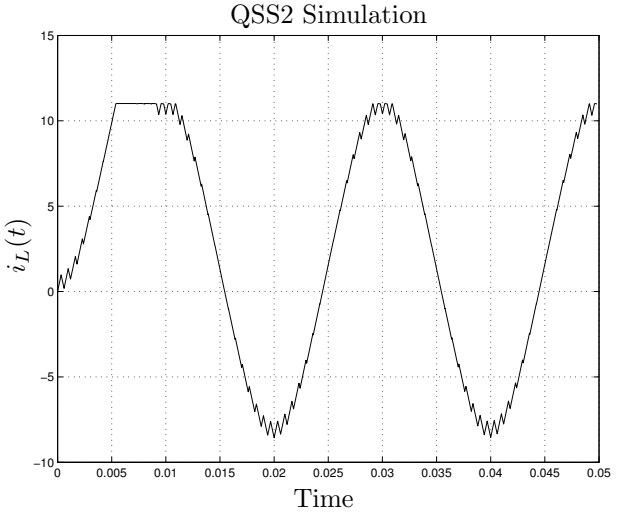

FIGURE 12.21. Initial behavior of load current with surge protection.

In this new situation, we cannot say anything about the global error bound. In the previous example, we knew that $s_w(t)$ was exactly generated. But now, $\tilde{s}_w(t)$ depends on i_L, which is not known exactly.

As mentioned above, we could have used i_L instead of q_{i_L} for the detection of the state events. However, the prediction is a bit more involved, since we must now solve a quadratic equation in Eq.(12.65). Moreover, in order to obtain the successive values of i_L, we need to look at the derivative of this signal and reintegrate it.

Thus, depending on the application, we can choose using either the continuous states or the quantized state variables for the detection of state events. Using the quantized state variables is more efficient computationally, but using the continuous states offers more accurate simulation results.

After these introductory examples, we are now ready to analyze a more general case.

We shall assume that the continuous subsystem can be represented by a set of DAEs, as specified below:

$$\dot{\mathbf{x}}_{\mathbf{a}}(t) = \mathbf{f}(\mathbf{x}_{\mathbf{a}}(t), \mathbf{u}(t), \mathbf{z}_{\mathbf{a}}(t), m_a(t)) \quad (12.66a)$$

$$0 = \mathbf{g}(\mathbf{x}_{\mathbf{a_r}}(t), \mathbf{u}_{\mathbf{r}}(t), \mathbf{z}_{\mathbf{a}}(t), m_a(t)) \quad (12.66b)$$

where $m_a(t)$ is a piecewise constant trajectory emanating at the discrete subsystem that defines the different operational modes of the system (this was the role of s_w and $\tilde{s}_w$ in our first and second example, respectively). Thus for each value of $m_a(t)$, there is a different DAE representing the system dynamics.

As we did in the previous section, we shall assume that the implicit equation, Eq.(12.66b), has a solution for each value of $m_a(t)$, which implies that the system of Eq.(12.66) does not exhibit conditional index changes.

Independently of the way in which $m_a(t)$ is being calculated, the submodel corresponding to the continuous part can be built as before, considering $m_a(t)$ as an input.

Then, the QSS and QSS2 methods will transform Eq.(12.66) to:

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{q}(t), \mathbf{u}(t), \mathbf{z}(t), m(t)) \quad (12.67a)$$

$$0 = \mathbf{g}(\mathbf{q}_{\mathbf{r}}(t), \mathbf{u}_{\mathbf{r}}(t), \mathbf{z}(t), m(t)) \quad (12.67b)$$

with the same definitions that we used in Eq.(12.58). Consequently, the simulation scheme for the continuous subsystem will be identical to the one shown in Fig.12.12, but now, $m(t)$ must also be calculated and included with the input $\mathbf{u}(t)$.

The way, in which $m(t)$ is calculated, is determined by the discrete subsystem.

One of the most important features of DEVS is its capability for representing any kind of discrete system. Taking into account that the continuous subsystem has been approximated by a DEVS model, it is only natural to also represent the discrete subsystem by another DEVS model. Then,

both DEVS models can be directly coupled building a single coupled DEVS model that approximates the entire system.

The DEVS model of the discrete subsystem will provoke events that carry the successive values of $m(t)$.

Taking into account the asynchronous fashion, in which the static functions and quantized integrators work, the events arriving from the discrete subsystem will be processed by the continuous subsystem as soon as they arrive, without a need of modifying anything in the QSS or QSS2 methods. Efficient event handling is an intrinsic characteristic of the QSS methods.

In the presence of state events, the discrete subsystem needs to detect the occurrence of such events by monitoring zero–crossing functions that are functions of inputs and state variables.

Here, the QSS and QSS2 methods have an even bigger advantage over discrete–time methods: input and state trajectories are known functions of time in a local context. They are either piecewise constant, or piecewise linear, or piecewise quadratic functions of time. For this reason, the time of occurrence of a state event that is about to take place can be calculated analytically, which makes it unnecessary to iterate on state events.

The only thing that has to be done is to provide those trajectories to the discrete subsystem, so that it can detect the occurrence of state events and calculate the trajectory $m(t)$, which it then passes on to the continuous subsystem. The continuous subsystem performs the state event handling in accordance with the changes in $m(t)$, as if these were mere time events.

The continuous state trajectories, $\mathbf{x}(t)$, are not directly available at the output of the quantized integrators. Only the quantized states, $\mathbf{q}(t)$, are generated by the quantized integrators. However, the state derivative functions are available. These can be integrated to obtain $\mathbf{x}(t)$. This is furthermore a very simple task that does not require much computational effort at all, since the state derivative trajectories are piecewise constant or piecewise linear (in QSS2), and obtaining their integrals only takes one or two simple calculations with the coefficients of the corresponding polynomials.

An alternative –and simpler– solution, which was the solution chosen in the introductory examples of this section, consists in using the quantized states instead of the continuous state variables in the discrete part. However, we may increase the error by doing so.

Using these ideas, the simulation model of a hybrid system, such as that of Eq.(12.66), using the QSS or QSS2 method will be a coupled DEVS with the structure shown in Fig.12.22.

Here the discrete part is a DEVS model that receives the events representing changes in the input trajectories and the quantized state variables. Alternatively to the quantized state variables, we could provide the state derivative signals to the discrete subsystem.

As state events can be detected before they occur, and since the discrete subsystem can set its time advance to that moment, ensuring the correct treatment of the corresponding event, there is no need for iterations or

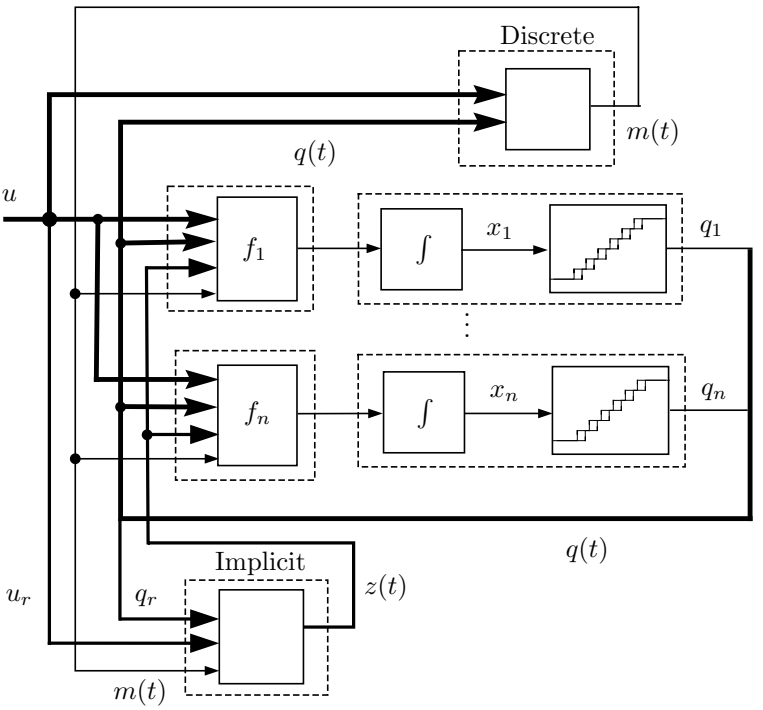

FIGURE 12.22. Coupling scheme for the QSS simulation of discontinuous systems.

back–stepping, in order to hit the event instants accurately. This is a very important advantage in the context of real–time simulation, as we shall discuss shortly.

In the following example, we shall see some further advantages of applying the QSS methods to the simulation of discontinuous systems.

A typical textbook example of a discontinuous system containing state events is the *bouncing ball* problem. We shall consider the case, where the ball moves in two directions, x and y, bouncing down a stairway. Thus, the bouncing condition depends on the two variables, x and y.

We postulate that the ball experiences air friction, which we shall assume linear for simplicity (in physics textbooks, air friction is usually assumed quadratic in the velocity), and we declare that, when the ball reaches the floor, it shall behave like a spring–damper system.

A corresponding differential equation model for the bouncing ball problem can be formulated as follows:

$$\dot{x} = v_x \tag{12.68a}$$

$$\dot{v}_x = -\frac{b_a}{m} \cdot v_x \tag{12.68b}$$

$$\dot{y} = v_y \tag{12.68c}$$

$$\dot{v}_y = -g - \frac{b_a}{m} \cdot v_y - s_w(t) \cdot (\frac{b}{m} \cdot v_x + \frac{k}{m} \cdot (y - y_f(t))) \tag{12.68d}$$

where m is the mass of the ball, b_a is the air friction constant, g is the gravitational constant, b is the damping constant, k is the spring constant, and:

$$s_w(t) = \begin{cases} 1 & \text{when } y_f(t) \triangleq h - \text{floor}(x) > y(t) \\ 0 & \text{otherwise, i.e., while the ball is in the air} \end{cases} \tag{12.69}$$

The function $y_f(t) = h - \text{floor}(x)$ calculates the height of the floor at any given horizontal position, x, where h is the height of the first step. We are considering steps of 1 m by 1 m, which correspond more to the dimensions of the steps on an Egyptian pyramid, than those of a modern–day stairway.

State events are being produced when x and y satisfy the condition:

$$y = h - \text{floor}(x)$$

or when $x = \text{floor}(x)$.

We decided to simulate the system using the QSS2 method. However this time around, we decided, not to use the quantized state variables for state event detection. Instead, we shall provide the derivatives of x and y, i.e., variables v_x and v_y to the discrete subsystem and re–integrate them.

The right hand side of Eq.(12.68), i.e., the continuous subsystem, depends only on the three variables v_x, v_y, and y. Consequently, the quantized state variable corresponding to x does not appear at all in the quantized system. Hence the quantized integrator that calculates it can be omitted.

Figure 12.23 shows the PowerDEVS model of the system. Here, the integrators to the left calculate v_y and y, whereas the integrator to the right computes v_x.

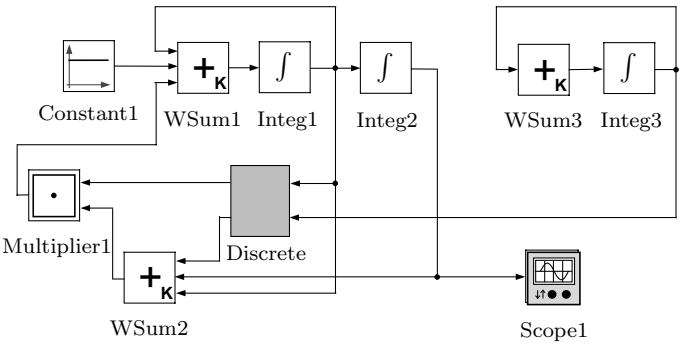

FIGURE 12.23. PowerDEVS bouncing ball model.

The discrete subsystem has been encapsulated. Figure 12.24 shows the complete discrete submodel.

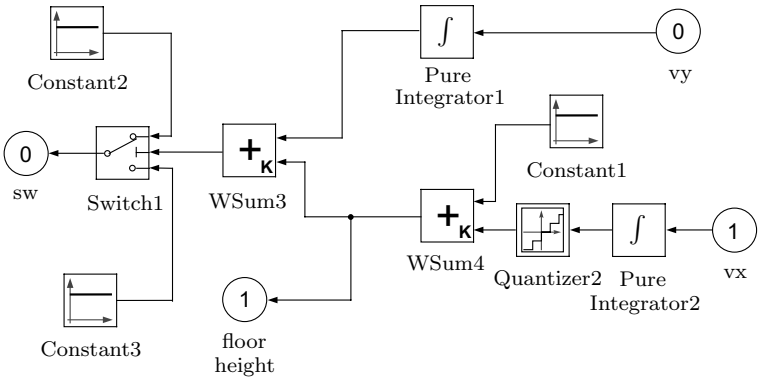

FIGURE 12.24. PowerDEVS bouncing ball model (discrete subsystem).

The discrete submodel contains two *Pure Integrator* blocks. These blocks compute exactly the integral of piecewise linear trajectories.

When one of these blocks receives an event with value $d_x(t_k)$ and linear slope $m_{d_x}(t_k)$, it immediately sends out an event with value $x(t_k) = x(t_{k-1}) + m_x(t_{k-1}) \cdot (t_k - t_{k-1}) + p_x(t_{k-1}) \cdot (t_k - t_{k-1})^2$, where the linear slope, m_x, is computed as $m_x(t_k) = d_x(t_k)$, and the quadratic slope, p_x, assumes a value of $p_x(t_k) = m_{d_x}(t_k)/2$. Then, it waits until a new input event arrives.

Thus, the trajectories calculated by these blocks are piecewise parabolic and continuous. In our example, they compute the state trajectories of $x(t)$ and $y(t)$.

The trajectory $x(t)$ is then sent to a *Quantizer* block that calculates $\text{floor}(x)$ in order to evaluate the height of the floor. This is a block provided by the *Hybrid* library of PowerDEVS.

We already encountered the *Switch* block once before in this section. In the given model, its second input port receives the values of $y(t) - y_f(t)$, and predicts, when this signal crosses the threshold of 0 in either direction, in order to detect the next floor contact or floor separation event.

For the simulation, we used the parameter values $k = 100,000\ N/m$, $b = 30\ kg/s$, $ba = 0.1\ kg/s$, and $m = 1\ kg$. To simplify the logistics of the simulation (quite significantly, to tell the truth!), the ball is assumed to be infinitely small. Thus, the spring–damper system represents in fact the elasticity of the floor, rather than that of the bouncing ball.

A quantum of $0.01\ m/s$ was chosen for the quantized integrators that calculate v_x and v_y, whereas a quantum of $0.0001\ m$ was chosen for the quantized integrator that calculates y. The initial conditions were $x(0) = 0.575\ m$, $v_x(0) = 0.5\ m/s$, $y(0) = 10.5\ m$ and $v_y = 0\ m/s$. We simulated

the system across 10 seconds of simulated time.

Figure 12.25 shows the simulation results for this system.

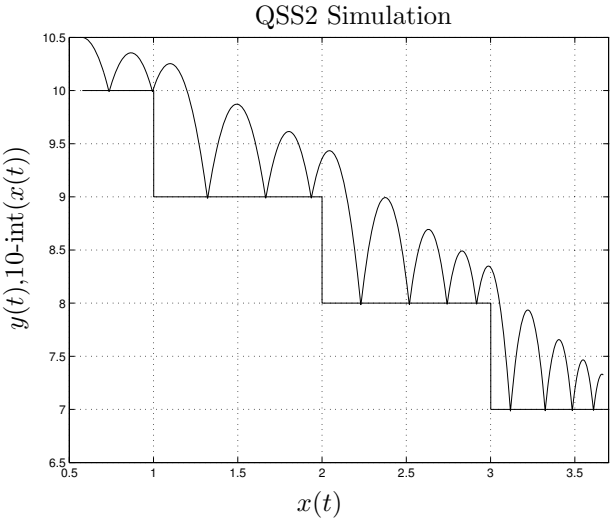

FIGURE 12.25. Ball bouncing down some stairs.

The integrator that calculates v_x required five steps, the integrator that calculates v_y required 518 steps, and the integrator that calculates y required 2413 steps. Hence the *Quantizer* block received only five external events. It detected four crossings by integer values, as the ball advanced four steps downward. Consequently, the routine that predicts crossings by solving a quadratic polynomial was invoked only nine times.

Similarly, the *Switch* block received 522 events through its second port, 518 of which originated at the quantized integrator of v_y, whereas the remaining four originated at the *Quantizer* block. It detected a total of 20 crossings, as each bounce results in two separate events, a floor contact event, and a floor separation event.

Notice that the second bounce was produced near the border of a step. Here, discrete–time methods will experience problems. Figure 12.26 details the result of simulating this system using a variable–step Runge–Kutta method (ode45 of MATLAB) with two different accuracy settings.

Using a sufficiently large tolerance value, the method skips the event. Since the time elapse between subsequent function evaluations is larger than the zone, in which the event condition is triggered, the algorithm does not recognize that it has passed through that zone.

An example of this problem was given in [12.5], where the authors proposed a solution based on decreasing the step size as the system approximates the discontinuity condition. In Chapter 9 of this book, we had proposed another solution involving the use of the derivative of the zero–crossing function as an additional (dummy) zero–crossing function.

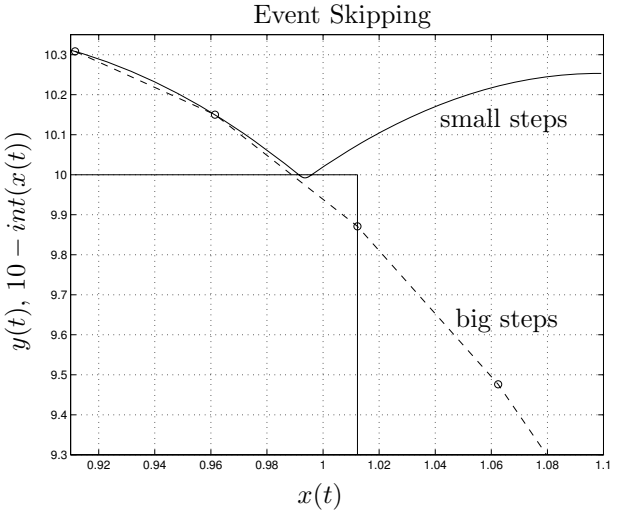

FIGURE 12.26. Event skipping in discrete–time algorithms.

In the QSS methods, this problem disappears. Provided that we use the continuous states instead of the quantized state variables for detecting the discontinuities, as we did in this example, it is impossible for QSS methods to skip *short–living state events* as discrete–time methods do.

12.10 Real–time Simulation

In Chapter 10, we studied, which algorithms were suitable in the context of real–time simulation. We concluded that fixed–step algorithms were necessary, and we only accepted explicit or semi–implicit methods. In the latter case, we tried furthermore to reduce the size of the matrix to be inverted using mixed–mode integration techniques. Thus at a first glance, QSS and QSS2 don't seem to fit the bill very well, since these are clearly variable–step algorithms.

We offered two reasons for discarding variable–step methods. The first reason was that we were not able to spend time discarding values and repeating steps, when we did not like the error obtained. The second reason was that the inputs and outputs of most real–time applications occur at fixed time intervals, which constitutes another good point in favor of fixed–step algorithms.

However, none of these reasons apply to quantization–based methods, since they never discard values, since they accept input changes at any point in time, and since their states are known at any instant of time also. QSS methods conveniently solve the *dense output* problem that we had encountered in Chapter 9 of this book.

We furthermore mentioned in Chapter 10 that real–time simulation of

discontinuous models is highly problematic, since it forces us to spend time detecting and handling events, i.e., we are back facing the same problems that had convinced us to discard variable–step algorithms.

Taking into account the way, in which QSS methods deal with discontinuities, quantization–based algorithms appear to be a very good choice for handling discontinuities in real time.

Moreover, not all variables have to be calculated at the same rate. We can let each variable update itself at the rate that it wants. We only need to ensure that the overall simulation clock proceeds faster than real time.

Let us illustrate these ideas with an example. The PWM strategy introduced in Fig.12.15 can be used to synthesize more general signals than sinusoidal functions. Another typical application of pulse width modulation is the control of DC motors. Since it is difficult and expensive to generate a variable continuous voltage with high power output, the desired power signal is replaced by a switching signal, in which the duration of the *on* state is proportional to the desired voltage. The comparison of the desired voltage with a fast triangular waveform permits achieving such behavior.

Figure 12.27 shows the block diagram of a control system based on this technique. The controller compares the speed $\omega(t)$ with the input reference, and calculates the desired input voltage of the motor, u_{ref}. This value is then compared to a triangular waveform, obtaining the actual input voltage u_a that oscillates between two values.

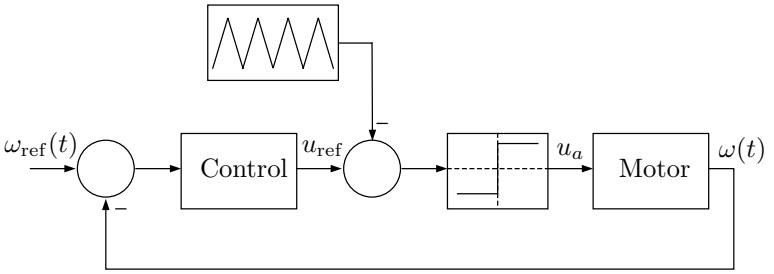

FIGURE 12.27. PWM controlled DC motor.

A real–time simulation of this kind of model may be needed in order to detect the presence of failures in the physical system that the model represents. As in the example of a watchdog monitor for a nuclear power station, mentioned in Chapter 10, the simulation output can be compared to the physical system output, and when the two signals differ significantly one from another, we can conclude that something went wrong with the system.

The simulation of this system requires accurate event detection. The moment, at which the triangular wave crosses the value given by u_{ref}, determines the actual voltage applied to the motor. A small error in the calculation of this time instant leads to a significant change in the output

waveform.

An additional problem is that the triangular wave operates at a high frequency, and therefore, state events occur with a high rate. Thus, the efficient treatment of discontinuities becomes crucial.

The motor can be represented by a second order model of the form:

$$\begin{aligned} \frac{di_a}{dt} &= \frac{1}{L_a} \cdot (u_a(t) - R_a \cdot i_a - k_m \cdot \omega) \\ \frac{d\omega}{dt} &= \frac{1}{J} \cdot (k_m \cdot i_a - b \cdot \omega - \tau(t)) \end{aligned}$$

where $L_a = 3\ mH$ is the armature inductance, $R_a = 50\ m\Omega$ is the armature resistance, $J = 15\ kg \cdot m^2$ is the inertia, $b = 0.005\ kg \cdot m^2/s$ is the friction coefficient, and $k_m = 6.785\ V \cdot s$ is the electro–motorical force (EMF) constant of the motor. These parameter values correspond to those of a real system.

The inputs are the armature voltage, $u_a(t)$, and the torque load, $\tau(t)$, respectively. In the given example, $u_a(t)$ switches between $+500\ V$ and $-500\ V$ depending on the PWM control law. A torque step of $2500\ N \cdot m$ is applied after 3 seconds of simulation.

For the PWM law, we consider a triangular waveform of $1\ kHz$ frequency with an amplitude of $1.1\ V$. This triangular wave is compared to the error signal, saturated at a value of $1\ V$. The control is using a proportional law, i.e., u_{ref} is proportional to the error $\omega_{ref}(t) - \omega(t)$.

The angular velocity reference signal, $\omega_{ref}(t)$ is a ramp signal that increases from 0 to 60 rad/sec in 2 seconds.

As a first step, we performed an off–line simulation using the PowerDEVS model of Fig.12.28.

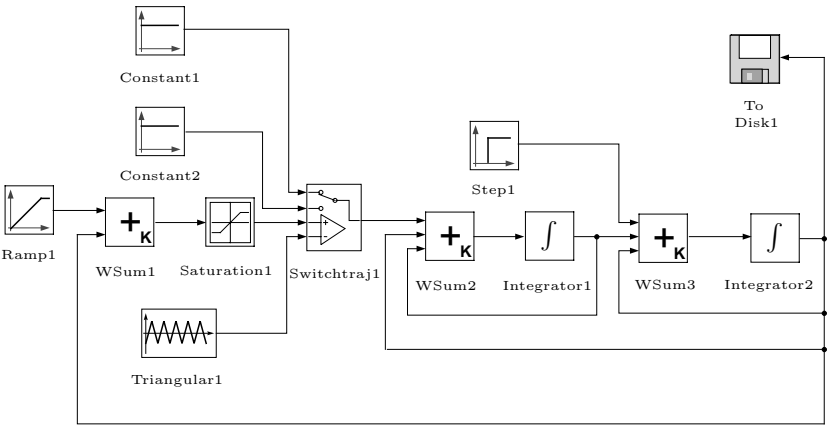

FIGURE 12.28. PowerDEVS model of the PWM controlled DC motor.

A simulation across 5 seconds of simulated time took about 0.63 seconds

of real time on a 950 Mhz computer running under Windows XP. This allows to predict that we may be able to simulate the system in real time. We used QSS2 with a quantum of 0.01 for both state variables, which produced 1952 and 43,263 steps in the integrators corresponding to the angular velocity and current, respectively. An additional 10,000 events were caused by the triangular wave generator, and another 10,000 events were provoked by the switch. The saturation block also produced 398 events, which led to a total of 65,613 steps.

Before proceeding to the real time experiment, we need to discuss the real time requirements of the input and output signals. We shall assume that the angular velocity, $\omega(t)$, needs to be measured once each 10 $msec$, and that it suffices to provide the model with updated values of the input signal, $\omega_{ref}(t)$, at the same rate.

Thus, we added *sample and hold* blocks that provoke events every 10 $msec$ to both the input and output signals.

In principle, it would not have been necessary to provide clocked events for the inputs and outputs, as DEVS models require input values only, when a change occurs in the input pattern, and as they are able to compute dense outputs on their own, as all trajectories are known signals in a local context. For the given input signal, it would thus have sufficed to provide a single event at the time, when the ramp goes into saturation. However, it may be convenient to operate with clocked input and output signals in a real time environment, as this allows to synchronize model inputs with signals, whose trajectories are not known in advance, and as it allows to hook the output trajectories of the real–time simulation to equipment that does not know anything about DEVS models.

One way to simulate this system in real time is to synchronize the entire simulation with a physical clock, so that each event is performed at a time that is as close as possible to the physical clock. We can do this directly with PowerDEVS, since it has the option of running a simulation synchronized with a physical clock.

However, this is unnecessary in our case. We only need to ensure that the values at the exits of the sample and hold blocks are sent out at the correct time instants. In other words, we only need synchronization once each 10 $msec$. Of course, in order to achieve this, we need to finish all calculations corresponding to each period of 10 $msec$, before that time window ends.

To obtain this behavior, we just added two new identical blocks called *Clock wait* that send out an event, as soon as they receive one in terms of *simulated* time, but contain an internal busy waiting loop that waits with sending out the event, until the physical clock has advanced to the value shown by the simulation clock. In this way, the synchronization routines are only invoked, when and where they are needed.

Figure 12.29 shows the modified PowerDEVS model that can be used for real–time simulation.

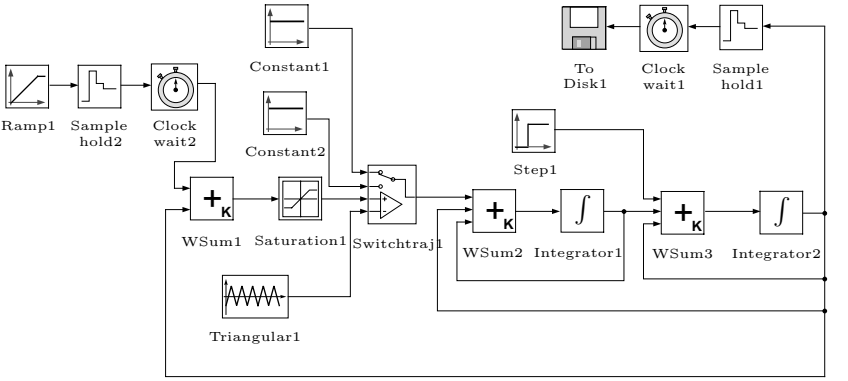

FIGURE 12.29. PowerDEVS model of the PWM controlled DC motor (RT).

The *Clock wait* block has been implemented using the following code in PowerDEVS:

ATOMIC MODEL CLOCK WAIT
State Variables and Parameters:
float $sigma$;
void $*xv$; //states
void $*y$; //output
float $itime$;

Init Function:
$inf = 1e10$;
$itime = 1.0 * clock()/CLOCKS\_PER\_SEC$;
$sigma = inf$;

Time Advance Function:
return $sigma$;

Internal Transition Function:
$sigma = inf$;

External Transition Function:
float $actime$;
$xv = x.value$;
$actime = 1.0 * clock()/CLOCKS\_PER\_SEC - itime$;
while $(actime < t)$ {
$actime = 1.0 * clock()/CLOCKS\_PER\_SEC - itime$;
}
$sigma = 0$;

Output Function:
$y = xv$;
return Event$(y, 0)$;

We simulated this new model, and saved the output data with both, the physical and the simulated time. All events at the outputs of the *Clock*

wait blocks were sent at the right physical time instants. The error was of the order of the accuracy of the physical clock access that the gcc compiler running under Windows permits. This means that the calculations were indeed performed on time.

The simulation results are shown in Figs.12.30–12.31.

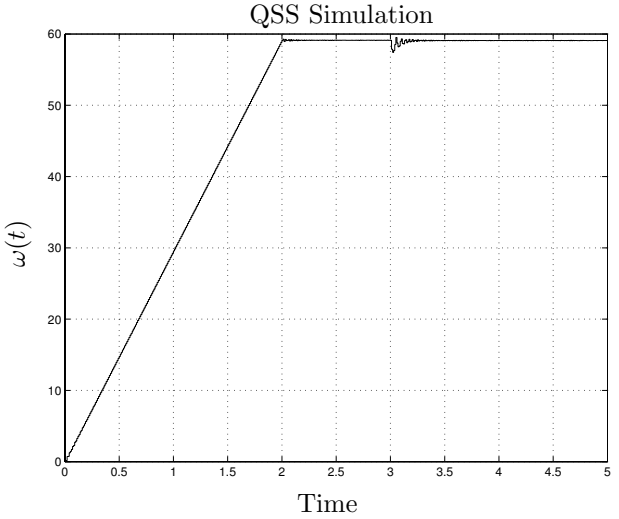

FIGURE 12.30. Simulation output of the DC motor.

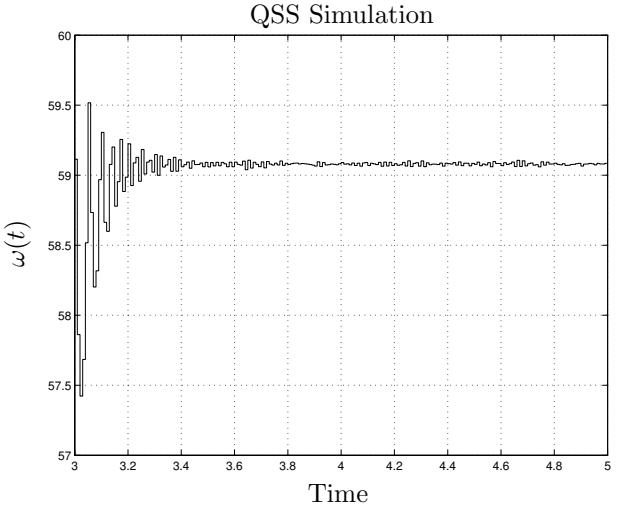

FIGURE 12.31. Simulation output of the DC motor (detail).

In this case, the most important advantage of using QSS2 is related to the efficient treatment of discontinuities. The off–line simulation of this

system with any of MATLAB's integration methods requires more than 20 seconds of real time under identical operating conditions, which makes a real–time simulation impossible.

There is another advantage connected to the fact that the system variables are not updated simultaneously. For example in our simulation, $\omega(t)$ was only updated 1953 times. Unless we use a very sophisticated multi–rate algorithm, any discrete–time method would calculate $\omega(t)$ at least 20,000 times, as there are 20,000 events in the system. Thus, we would have to perform all calculations corresponding to each step in a period shorter than 0.25 *msec*.

In the QSS2 case, this period becomes about 10 times longer. It is true that we might have more calculations between two steps in ω. However, it is much easier to perform 1000 calculations in 10 *msec*, than performing 100 calculations in 1 *msec*.

Of course, the same idea of synchronizing only when and where it is necessary can also be used in the discrete–time case. However, this is not as easy in the case of the QSS2 method. Moreover, when we are forced to change the step size because of the presence of discontinuities, time synchronization becomes rather tricky, as we discussed in Chapter 10.

12.11 Open Problems in Quantization–based Methods

As we had mentioned in the previous chapter, the discrete event simulation of continuous systems is a newly developing area of research.

Despite the theoretical and practical advantages that we showed along this chapter, there are still many important problems that should be solved, before it can be claimed that quantization–based approaches represent a good choice for the simulation of general continuous systems.

The most important problem is probably related to the solution of *stiff systems*. Let us introduce the issue by means of a simple example. We shall consider the second–order ODE system given by:

$$\begin{aligned} \dot{x}_1 &= 100 \cdot x_2 \\ \dot{x}_2 &= -100 \cdot x_1 - 10,001 \cdot x_2 + u(t) \end{aligned} \tag{12.70}$$

The eigenvalues of this linear system are located at $\lambda_1 = -1$ and $\lambda_2 = -10,000$, which means that this system, in spite of its simplicity, is stiff.

We simulated the system across 10 seconds of simulated time using the QSS method with quantum sizes of 1×10^{-2} for x_1, and 1×10^{-4} for x_2. According to Eq.(12.27), this quantization ensures that the error in x_1 is bounded by 0.01, whereas the error in x_2 is bounded by 0.0003.

The initial conditions were both set equal to zero, and the input was chosen as a step function, $u(t) = 100 \cdot \varepsilon(t)$. The simulation was completed

after 100 internal transitions in the quantized integrator that calculates x_1, and after 200 internal transitions in the quantized integrator that calculates x_2. The trajectory of the quantized state q_2 is shown in Figs.12.32–12.33.

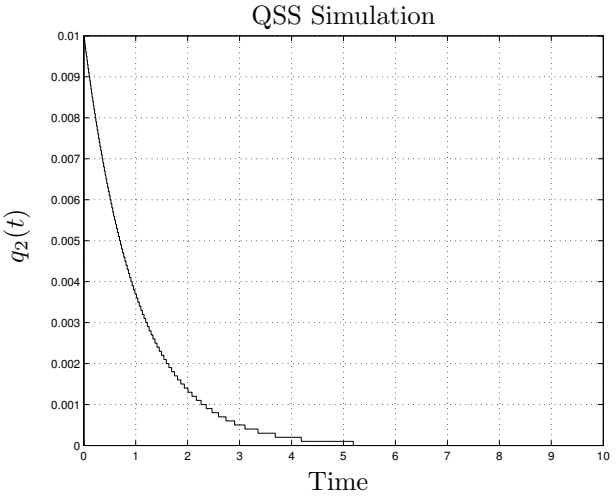

FIGURE 12.32. QSS simulation of the system of Eq.(12.70).

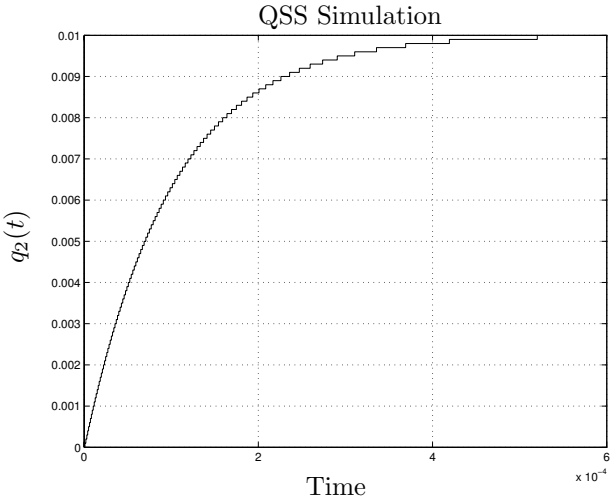

FIGURE 12.33. QSS simulation of the system of Eq.(12.70), startup period.

The stiffness of this system becomes evident, when we compare the time scale of Fig.12.32 with that of Fig.12.33. In this example, the performance of the QSS method is truly amazing. The method was able to adjust the step size in a very natural way, and performed the overall simulation in a surprisingly small number of integration steps. The number of transi-

tions performed can be calculated here by dividing the amplitude of the trajectory by the quantum.

Looking at this result, we may think that the QSS technique is a wonderful method for solving stiff systems: it is an *explicit* method that allows simulating a stiff system much faster and more efficiently than the most complex implicit variable–step algorithms that we have met throughout this book.

This idea seems coherent with the fact that the QSS method is always "stable." A simulation method that good must surely turn the entire existing vault of theories concerning numerical ODE solutions upside down. Armies of applied mathematicians will have to rebuild from scratch the foundations of their trade.

Yet, we must not forget the meaning that we have given to the term "stable." The QSS method does not ensure asymptotic stability of a solution, but only its boundedness, and, unfortunately, this can become a problem when dealing with stiff systems.

To illustrate our point, let us check what happens if we introduce a small modification to the previously discussed example. We changed the input step function from $u(t) = 100 \cdot \varepsilon(t)$ to $u(t) = 99.5 \cdot \varepsilon(t)$, and repeated the simulation.

In the first five seconds of simulated time, the quantized integrator that calculates x_1 performed 100 internal transitions, i.e., the number stayed approximately the same as before, but the quantized integrator that calculates x_2 underwent a total of $25,057$ transitions! The trajectory of q_2 is shown in Figs.12.34–12.35.

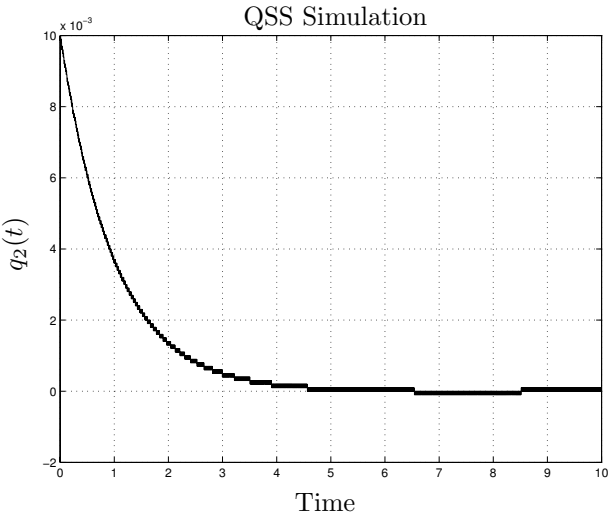

FIGURE 12.34. QSS simulation of the system of Eq.(12.70) with $u = 99.5$.

The results of this simulation are very similar to those obtained earlier.

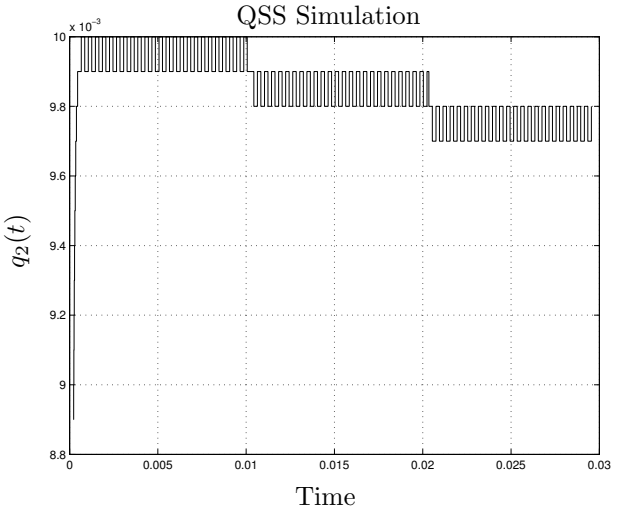

FIGURE 12.35. Details of the initial phase of Fig.12.34.

The error, in agreement with the theory, remains bounded, but the number of calculations is huge.

The reason is the appearance of ultra–fast oscillations in x_2 that ruined our triumph of having found an explicit, efficient, stable, and reliable method for dealing with stiff system. If those oscillations were not present, we could have calculated the number of transitions by dividing the trajectory amplitude by the quantum. However, this is not possible here.

Although these oscillations may have come as a surprise, they are in fact already familiar to us. Indeed, they are no different from the oscillations that we encountered in Fig.12.1 on page 557. There, the oscillations did not pose a problem, since their period was not much shorter than the settling time of the system.

Unfortunately, the oscillation frequency is related to the eigenfrequencies of the system, i.e., the location of its eigenvalues, which is bad news indeed.

How can we solve this problem? We do not have a final answer to this question yet.

An interesting trick that works in many cases is based on the use of concepts borrowed from the design of implicit ODE solvers. We have learnt very early on in this book that stiff systems require implicit algorithms for their solution. Whereas the FE algorithm:

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

was unsuccessful in dealing with stiff systems, the implicit BE algorithm:

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

could deal with them successfully. The difference between these two algo-

rithms consists in the time instant, at which the state derivative is being considered in computing the next state.

We may be able to use the same idea in the design of an *Implicit Quantized State System (IQSS)* method. If only we knew, what the next state and the next input would be, we could compute the next state derivative:

$$\dot{x}_{i_{k+1}} = f_i(\mathbf{q_{k+1}}, \mathbf{u_{k+1}}) \tag{12.71}$$

and then we could determine the next quantized state, $q_{i_{k+1}}$, together with the time of occurrence of the next internal transition by making use of that future state derivative.

Explicit integration algorithms are loop breakers, whereas implicit algorithms are not. We saw this already in Chapter 7 in the discussion of the classical DAE solvers. Implicit algorithms invariably led to much larger algebraic loops.

The same observation is true for QSS methods as well. Until now, we were able to deal with the static functions independently of the hysteretic quantized integrators, because the integrators are breaking the algebraic loops. This is no longer the case, when we are looking at IQSS methods.

Yet, the problem may not be as bad as it looks. There are two points in our favor:

1. Although we don't know the value of the next quantized state, $q_{i_{k+1}}$, there are only two possible values that this state can assume: $q_{i_{k+1}} = q_{i_k} \pm \Delta Q$.

2. Looking at the integrator that is going to transition next, we only need to consider the future state of that integrator, as all other integrators still carry their previous value.

Given an n^{th}–order system, we require an algorithm that can decide, which of the n integrators is going to transition next. Once we know this, we no longer need to iterate over n variables simultaneously. It then suffices to iterate over a single variable. Furthermore, "iteration" may be a rather fancy word for what needs to be done, since the future state can only assume one of two values.

The following algorithm could be used:

1. Given any state $q_i = Q$, where Q is the current value of the state q_i.

2. We replace Q by $Q + \Delta Q$ in the corresponding state equation: $\dot{x}_i = f_i(\mathbf{q}, \mathbf{u})$, assuming that the state variable x_i is about to increase.

3. We check, whether $\dot{x}_i$ is positive. If this is the case, we can now compute t_{k+1}, the next transition time. If $\dot{x}_i$ is negative, the assumption made was incorrect.

4. We now replace Q by $Q - \Delta Q$ in the corresponding state equation: $\dot{x}_i = f_i(\mathbf{q}, \mathbf{u})$, assuming that the state variable x_i is about to decrease.

5. We check, whether $\dot{x}_i$ is negative. If this is the case, we can now compute t_{k+1}, the next transition time. If $\dot{x}_i$ is positive, the assumption made was incorrect.

6. If one of the two assumptions turns out to be correct, whereas the other is incorrect, we know the next transition time for this state variable. If neither of the assumptions is correct, or if both are correct, we assume that the state isn't going to change on its own, and set the corresponding σ value to ∞.

7. We repeat the same algorithm for all integrators. The shortest t_{k+1} is the time of the next internal transition, and the corresponding state variable is the one that is going to transition next.

The algorithm is clearly more expensive than the explicit QSS algorithm, and it is more centralized. After each event, we need to recompute the time advance functions of all quantized integrators, and determine the one integrator that will transition next.

Notice that we should also have considered future values of the inputs in the calculation of the state derivative functions. However, the input values do not affect stability, and since our primary aim was to improve the stability behavior, i.e., get rid of the high–frequency oscillations, it may not be necessary to take future input values into account.

A much cheaper, and fully decentralized, version of this algorithm can also be proposed. In this new approach, we only look at the integrator that is undergoing an internal transition now, and compute its next value and time advance in the following way:

1. Let $q_i = Q$ be a quantized state that is currently going through an internal transition.

2. We replace Q by $Q + \Delta Q$ in the corresponding state equation: $\dot{x}_i = f_i(\mathbf{q}, \mathbf{u})$, assuming that the state variable x_i is about to increase.

3. We check, whether $\dot{x}_i$ is positive. If this is the case, we can now compute t_{k+1}, the next transition time. If $\dot{x}_i$ is negative, the assumption made was incorrect.

4. We now replace Q by $Q - \Delta Q$ in the corresponding state equation: $\dot{x}_i = f_i(\mathbf{q}, \mathbf{u})$, assuming that the state variable x_i is about to decrease.

5. We check, whether $\dot{x}_i$ is negative. If this is the case, we can now compute t_{k+1}, the next transition time. If $\dot{x}_i$ is positive, the assumption made was incorrect.

6. If one of the two assumptions turns out to be correct, whereas the other is incorrect, we know the next state value and the next internal transition time for this state variable. If neither of the assumptions is correct, or if both are correct, we assume that the state isn't going to change on its own, and set the corresponding σ value to ∞.

The cheap version of the IQSS algorithm is hardly more expensive than the explicit QSS method, yet it may improve the stability behavior of the method.

A remark here is that the quantized integrators have to be able to evaluate f_i, which means that we should no longer separate the quantized integrators from the static functions that calculate the derivatives.

Although the resulting atomic models are more complex, since they combine the features of a quantized integrator and a static function, the simulation becomes more efficient, since the number of events is reduced to less than one half, as we no longer have to transmit events from the static functions to the quantized integrators and from the quantized integrators back to the static functions that compute their own derivatives.

We implemented this idea for the above example, and repeated the simulation with $u(t) = 99.5 \cdot \varepsilon(t)$. The number of transitions was now approximately the same as when using the QSS method with $u(t) = 100 \cdot \varepsilon(t)$, i.e., the high–frequency oscillations have indeed disappeared.

However, we have not yet been able to prove that this approach works for all stiff systems, i.e., constitutes an efficient solution for stiff systems in general.

Furthermore, it may not be entirely trivial to generalize this technique to higher–order IQSS methods, as the number of combinations of states and slopes to be checked grows, and the search for consistent assumptions becomes quite a bit more involved.

Thus, in spite of the fact that we have been able to find an attractive and efficient quantization–based method for dealing with the above example, we cannot claim that the same approach will always work, and therefore, we consider that the problem is still open.

Another open problem in these methods has to do with the choice of the quantization. Although we mentioned algorithms and showed formulae for choosing the quantum in accordance with the desired error, this is not a completely satisfactory solution.

Except in applications, where we need to ensure a fixed error bound, which justifies performing a precise analysis prior to running the simulation, nobody wants to calculate a Lyapunov function or compute the eigenvalue and eigenvector matrices of a system to determine the quantum to be used.

An interesting idea would consist in developing quantum adaptation algorithms similar in concept to the step–size control algorithms of the variable–step discrete–time methods. Yet, there is no published research yet relating to this topic.

Another possibility might be to employ a logarithmic quantization scheme, so that the quantum becomes larger, when the variables assume bigger values. In that way, we might expect the algorithm to control the relative error, instead of the absolute error.

There are many other open problems that will probably be solved soon. We can easily imagine methods of orders greater than two, enjoying the same properties as QSS and QSS2, but reducing considerably the error bounds. A third–order accurate method (QSS3) has already been proposed and implemented in PowerDEVS [12.16, 12.17]. However, QSS3 is still not fully functional, as the currently implemented version does not work yet for general nonlinear blocks, and is not yet able to solve DAEs.

The use of high–order methods will also help with the choice of the quantum, since it will allow us to adopt a conservatively small quantum without a significant increase in the number of calculations. If we use a method of order five, for instance, the use of a quantum 1000 times smaller than the appropriate quantum would only increase the number of calculations by about a factor of four.

Another idea, mentioned in [12.13], is to apply QSS and QSS2 to the simulation of PDEs to exploit the natural sparsity of the resulting ODEs after applying the method of lines. In fact, we did something similar already in the transmission line examples.

The block diagrams of these ODEs have a very particular form, whereby a basic structure is repeated along the different spatial sections. Since we approximate each section by a DEVS model, we obtain in the process a sort of *cellular automaton*. Gabriel Wainer defined a particular formalism for describing cellular DEVS models, called *Cell–DEVS* [12.20], and he then combined it with QSS for the simulation of PDEs [12.21]. Similar approaches can also be found in [12.7] and [12.19].

Unfortunately, we saw that the method–of–lines approximation of PDEs of the parabolic type invariably leads to stiff ODEs. Thus, the problem of an efficient QSS simulation of stiff systems must be solved, in order to arrive at a general discrete event method for the simulation of distributed parameter systems.

Finally, another problem that has been treated recently is the application of QSS methods to marginally stable systems [12.12]. There, it was proven that, in the presence of purely imaginary eigenvalues, the error bound grows linearly with time and also depends linearly on the quantum.

Moreover, some simulation examples have shown that not only the error, but also the amplitude of the oscillations grows linearly with time, and hence the simulation becomes unstable.

However, as we saw, the use of QSS methods is equivalent to introducing a bounded perturbation. If that perturbation were not correlated with the state evolution, the presence of purely imaginary eigenvalues would not cause any problem. In fact, the response of a marginally stable system to a signal that does not contain spectral components at the resonance fre-

quency is not unstable. Unfortunately, the presence of hysteresis introduces a large perturbation at the resonance frequency.

Thus, a modification of the quantized integrators, that attempts to eliminate these perturbation components, was proposed. Although the idea noticeably improves the results, a slowly increasing unstable term still remains.

Again, the usage of an IQSS algorithm might help solve this problem.

12.12 Summary

This chapter introduced the main theoretical and practical issues related to a new family of numerical methods.

Based on the idea of replacing time discretization by state quantization, two new ODE solvers, QSS and QSS2, were developed that exhibit theoretical properties, which differ noticeably from those of the classic discrete–time methods. The existence of a global error bound that can be explicitly calculated is probably the most interesting feature in this context.

The asynchronous nature of these methods and the knowledge of the state trajectories at any instant of time permits dealing with discontinuous systems in a very efficient fashion. In the presence of state and/or time events that occur with a frequency of the same order as the eigenfrequencies of the system, QSS and QSS2 can reduce significantly the simulation time with respect to conventional algorithms.

Further advantages can be observed in the simulation of DAE systems, and in the way of dealing with input signals.

In spite of all this, QSS and QSS2 exhibit a major drawback in the presence of stiff systems due to the frequent appearance of fast oscillations, but we should not be discouraged by these findings. First attempts at tackling the problem by introducing modified QSS algorithms that are implicit algorithm, and yet, don't require true iterations, led to very promising results.

To us, discrete event integration methods constitute one of the most exciting recent developments in the field of numerical ODE solutions. There are still lots of open problems that can constitute subjects of research for future MS theses and PhD dissertations, which should be good news for aspiring young applied mathematicians in search of a research topic.

QSS methods may look exotic and unfamiliar at a first glance, yet it is always the departure to new shores and unexplored lands that ultimately reaps the most benefit. It is the unknown and unexplored that keeps science alive.

12.13 References

[12.1] François Baccelli, Guy Cohen, Geert Jan Olsder, and Jean-Pierre Quadrat. *Synchronization and Linearity: An Algebra for Discrete Event Systems*. John Wiley & Sons, 1992. 485p.

[12.2] François E. Cellier. *Continuous System Modeling*. Springer–Verlag, New York, 1991. 755p.

[12.3] James B. Dabney and Thomas L. Harman. *Masterink SIMULINK 4*. Prentice Hall, Upper Saddle River, N.J., 2001.

[12.4] Hilding Elmqvist. *Dymola — Dynamic Modeling Language, User's Manual, Version 5.3*. DynaSim AB, Research Park Ideon, Lund, Sweden, 2004.

[12.5] Joel M. Esposito, R. Vijay Kumar, and George J. Pappas. Accurate Event Detection for Simulating Hybrid Systems. In *Proceedings of the* 4^{th} *International Workshop on Hybrid Systems: Computation and Control*, volume 2034 of *Lecture Notes in Computer Science*, pages 204–217. Springer–Verlag, London, 2001.

[12.6] Yehea I. Ismail, Eby G. Friedman, and José Luis Neves. Figures of Merit to Characterize the Importance of On-chip Inductance. *IEEE Trans. on VLSI*, 7(4):442–449, 1999.

[12.7] Rajanikanth Jammalamadaka. Activity Characterization of Spatial Models: Application to Discrete Event Solution of Partial Differential Equations. Master's thesis, The University of Arizona, 2003.

[12.8] Hassan K. Khalil. *Nonlinear Systems*. Prentice–Hall, Upper Saddle River, N.J., 3^{rd} edition, 2002. 750p.

[12.9] Ernesto Kofman and Sergio Junco. Quantized Bond Graphs: An Approach for Discrete Event Simulation of Physical Systems. In *Proceedings International Conference on Bond Graph Modeling and Simulation*, volume 33 of *Simulation Series*, pages 369–374. Society for Modeling and Simulation International, 2001.

[12.10] Ernesto Kofman and Sergio Junco. Quantized State Systems: A DEVS Approach for Continuous System Simulation. *Transactions of SCS*, 18(3):123–132, 2001.

[12.11] Ernesto Kofman, Jong Sik Lee, and Bernard Zeigler. DEVS Representation of Differential Equation Systems: Review of Recent Advances. In *Proceedings European Simulation Symposium*, pages 591–595, Marseille, France, 2001. Society for Modeling and Simulation International.

[12.12] Ernesto Kofman and Bernard Zeigler. DEVS Simulation of Marginally Stable Systems. In *Proceedings of IMACS 2005*, Paris, France, July 2005.

[12.13] Ernesto Kofman. A Second Order Approximation for DEVS Simulation of Continuous Systems. *Simulation*, 78(2):76–89, 2002.

[12.14] Ernesto Kofman. Quantization–based Simulation of Differential Algebraic Equation Systems. *Simulation*, 79(7):363–376, 2003.

[12.15] Ernesto Kofman. Discrete Event Simulation of Hybrid Systems. *SIAM Journal on Scientific Computing*, 25(5):1771–1797, 2004.

[12.16] Ernesto Kofman. A Third Order Discrete Event Simulation Method for Continuous System Simulation. Part I: Theory. In *Proceedings of RPIC'05*, Río Cuarto, Argentina, 2005.

[12.17] Ernesto Kofman. A Third Order Discrete Event Simulation Method for Continuous System Simulation. Part II: Applications. In *Proceedings of RPIC'05*, Río Cuarto, Argentina, 2005.

[12.18] Ernesto Kofman. Non–Conservative Ultimate Bound Estimation in LTI Perturbed Systems. *Automatica*, 41(10):1835–1838, 2005.

[12.19] James J. Nutaro, Bernard P. Zeigler, Rajanikanth Jammalamadaka, and Salil Akerkar. Discrete Event Solution of Gas Dynamics within the DEVS Framework. In *Proceedings of International Conference on Computational Science*, volume 2660 of *Lecture Notes in Computer Science*, pages 319–328, Melbourne, Australia, 2003. Springer–Verlag, Berlin.

[12.20] Gabriel Wainer and Norbert Giambiasi. Application of the Cell–DEVS Paradigm for Cell Spaces Modeling and Simulation. *Simulation*, 76(1):22–39, 2001.

[12.21] Gabriel Wainer. Performance Analysis of Continuous Cell–DEVS Models. In *Proceedings* 18$^{\text{th}}$ *European Simulation Multiconference*, Magdeburg, Germany, 2004.

12.14 Bibliography

[B12.1] Norbert Giambiasi, Bruno Escude, and Sumit Ghosh. GDEVS: A Generalized Discrete Event Specification for Accurate Modeling of Dynamic Systems. *Transactions of SCS*, 17(3):120–134, 2000.

[B12.2] Ernesto Kofman. *Discrete Event Simulation and Control of Continuous Systems*. PhD thesis, Universidad Nacional de Rosario, Rosario, Argentina, 2003.

[B12.3] Herbert Praehofer. *System Theoretic Foundations for Combined Discrete–Continuous System Simulation*. PhD thesis, Johannes Kepler University, Linz, Austria, 1991.

[B12.4] Bernard Zeigler and Jong Sik Lee. Theory of Quantized Systems: Formal Basis for DEVS/HLA Distributed Simulation Environment. In *SPIE AeroSense'98 Proceedings: Enabling Technology for Simulation Science (II)*, volume 3369, pages 49–58, Orlando, Florida, 1998.

12.15 Homework Problems

[H12.1] Error Bound in LTI Systems

Given the following LTI system:

$$\begin{aligned} \dot{x}_{a_1} &= x_{a_2} \\ \dot{x}_{a_2} &= -2 \cdot x_{a_1} - 3 \cdot x_{a_2} \end{aligned}$$

with initial conditions, $x_{a_1}(0) = x_{a_2}(0) = 1$.

1. Find the analytical solution, $x_{a_1}(t)$ and $x_{a_2}(t)$.
2. Obtain an approximate solution, $x_1(t)$ and $x_2(t)$, with the QSS technique, using a uniform quantization in both variables of $\Delta Q = \varepsilon = 0.05$.
3. Draw the error trajectories, $e_i(t) = x_i(t) - x_{a_i}(t)$, and compare them with the error bound given by Eq.(12.27).

[H12.2] Input Quantization Error

Prove the validity of Eq.(12.31).

[H12.3] Approximate Input Signals

Build DEVS models that generate events representing piecewise constant trajectories that approximate the following signals:

1. A ramp
2. A pulse
3. A square wave
4. A saw-tooth signal
5. A trapezoidal wave

Develop also corresponding PowerDEVS models, and try them out with some simple systems.

[H12.4] QSS2 Linear Static Function

Obtain a DEVS model for a linear static function:

$$f_i(\mathbf{z}) = \sum_{j=1}^{l} a_{i,j} \cdot z_j \tag{H12.4a}$$

that takes into account values and slopes.

Implement the model in PowerDEVS, and try it out, together with the second–order quantized integrator, to simulate the system of Eq.(11.11).

[H12.5] QSS2 Nonlinear Static Functions

Obtain DEVS models and the corresponding PowerDEVS models of the following static functions for QSS2:

1. $f_a(v) = v^n$, with n being an integer parameter.
2. $f_b(v) = e^v$.
3. $f_c(v) = \cos(2\pi \cdot f \cdot v + \phi)$, where f and ϕ are real–valued parameters.
4. $f_d(v_1, v_2) = v_1 \cdot v_2$

Exploit the fact that you can analytically calculate the partial derivatives of these functions.

[H12.6] Input Signals in QSS2

Repeat problem H12.3 by building signal generators for the QSS2 method, now considering piecewise linear trajectories.

[H12.7] Approximate Sinusoidal for QSS2

Propose a DEVS model that generates a piecewise linear approximation to a sinusoidal wave.

Can you obtain a solution similar to that of model M_7 provided on page 566? Explain the differences.

[H12.8] QSS2 Loop–breaking Block

Obtain a loop–breaking DEVS model for the QSS2 method analogous to M_{11}. Implement it in PowerDEVS, and simulate the circuit of Fig.12.7 using the QSS2 method.

[H12.9] Hybrid DAE Simulation

The circuit of Fig.H12.9a represents a modification of the example presented in Fig.12.14. Here, a resistor, R_p, and a nonlinear component were included to limit the voltage of the load resistor, R.

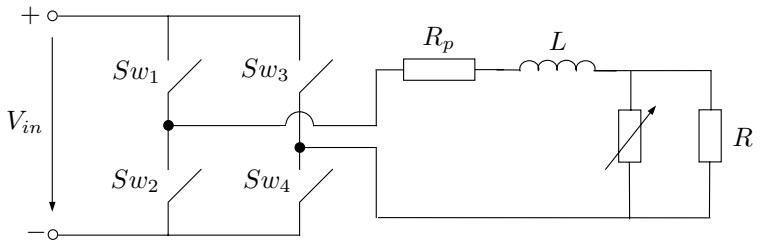

FIGURE H12.9a. DC–AC inverter with surge protection.

We shall assume that the nonlinear component is characterized by a varistor–like voltage–current relationship:

$$i(t) = k \cdot u(t)^{\alpha} \tag{H12.9a}$$

Under this assumption, the equation describing the system dynamics becomes a nonlinear DAE:

$$\frac{di_L}{dt} = \frac{1}{L} \cdot (-R_p \cdot i_L - u + s_w \cdot V_{in}) \tag{H12.9b}$$

where u must satisfy the nonlinear equation:

$$i_L - k \cdot u^{\alpha} - \frac{u}{R} = 0 \tag{H12.9c}$$

Simulate this system in PowerDEVS using the QSS2 method. Use the same parameters as in the example of Fig.12.14, choosing in addition $R_p = 0.01\ \Omega$, $k = 5^{-7}\ mho$, and $\alpha = 7$.

Do you notice any further advantage of using quantization–based integration techniques in this example?

12.16 Projects

[P12.1] Pulse on a Transmission Line

Consider the transmission line model of Fig.12.5. It is composed of five sections of RLC circuits.

The goal of this project is to study the effects of varying the number of sections on the computational cost of the simulation.

To this end, a short pulse input will be considered. A pulse with a duration of $2 \times 10^{-10}\ sec$ may be appropriate.

Then, the idea is to start with a transmission line circuit consisting of a single section, and then to gradually increase the number of sections until at least 20.

In each case, measure the total number of transitions and the simulation time, and try to find a law that relates the computational cost to the number of sections.

Then, repeat the same experiment with longer pulses, and with periodic waves, such as the trapezoidal wave used a few times in the chapter.

If you find any difference in the relationship between the computational cost and the number of segments for any of the new inputs, try to explain the reason for this difference.

[P12.2] Logarithmic Quantization

Obtain a DEVS model of a logarithmically quantized integrator for the QSS and the QSS2 methods. To this end, use a quantum proportional to the state variable value. You will also need to define a minimum value for the quantum, as otherwise, you might obtain illegitimate behavior.

Once you have built the corresponding PowerDEVS models, take some of the examples presented in this chapter, and simulate them using the newly developed logarithmically quantized integrators. Study the advantages and disadvantages of this new approach.

Using logarithmic quantization, can you still guarantee stability as we did before?

Study the problems related to stability and global error bound from a theoretical point of view. Start with a first–order linear system, and then try to extend the analysis, if at all possible, to the general LTI case.

12.17 Research

[R12.1] Integration of PDEs

The use of the method of lines in PDEs produces a system of sparse ODEs or DAEs. As we saw, the QSS and QSS2 methods exploit sparsity in a very efficient fashion.

Study the advantages and disadvantages of using the quantization–based integration methods together with the method of lines for the simulation of PDEs. Hyperbolic PDEs may be of particular interest in this context. Do not forget to take into account the particular problems of shock waves and discontinuous PDEs.

[R12.2] QSS simulation of Stiff Systems

Investigate the possibility of introducing modifications to the QSS method, in order to improve their ability for dealing with stiff systems.

You can use the remarks of Section 12.11 as a starting point.