Quantized State System Simulation

François E. Cellier
Inst. Computational Science CAB G82.1
ETH Zürich
Universitätsstr. 6
8092 Zürich
Switzerland
FCellier@Inf.ETHZ.CH

Ernesto Kofman, Gustavo Migoni, Mario Bortolotto
Laboratorio de Sistemas Dinámicos
Universidad Nacional de Rosario
Riobamba 245 bis
2000 Rosario
Argentina
{Kofman,Migoni,Bortolotto}@Cifasis-Conicet.Gov.AR

ABSTRACT

The paper introduces a new family of numerical ODE solvers called Quantized State System (QSS) methods. Given a set of ODEs in its state-space representation, the QSS methods replace the classic time slicing by a quantization of the states, leading to an asynchronous discrete-event simulation model instead of a discrete-time difference equation model.

QSS methods applied to stable linear time-invariant systems give always practically stable numerical results, irrespective of the quantization adopted. Taking into account that the QSS methods are explicit algorithms, this property has strong theoretical implications and offers a promising perspective for applications such as real-time simulation of stiff systems, where implicit solutions are usually unacceptable.

Also discussed are the main properties of the methods in the context of simulating discontinuous systems (the asynchronous nature of these algorithms gives them important advantages for discontinuity handling). Another class of systems that can be simulated by means of these algorithms are marginally stable (Hamiltonian) systems[1].

Keywords: Quantized State System Simulation, QSS Simulation, Discontinuous System Simulation, Explicit Stiff System Solver, Real-time Stiff System Simulation, Marginally Stable System Simulation, Distributed Parallel Simulation.

INTRODUCTION

When solving ordinary differential equations on a digital computer, it is necessary to perform some type of discretization, as a digital computer can only perform a finite number of operations within a finite time span.

Commonly, it is the time axis that is being discretized. We refer to algorithms that discretize the time axis as time-slicing algorithms. The discretization may either be equidistant (fixed-step algorithms) or dependent on the numerical properties of the differential equations to be solved (variable-step algorithms). The approximation order may be either fixed (fixed-order algorithms) or time-dependent (variable-order algorithms).

Most algorithms operate on a single clock, i.e., multiple differential equations are discretized using the same time-slicing algorithm. Such algorithms are referred to as synchronous algorithms. Simulations may also operate on multiple clocks (asynchronous algorithms), but the additional computations necessary to synchronize the solutions make asynchronous simulation methods rarely profitable.

The solution to the differential equation is approximated by fitting an interpolation polynomial through past state and state derivative values. The interpolation polynomial is then used to extrapolate in time to the next sampling instant in the case of explicit algorithms. Implicit algorithms avoid the extrapolation by making use of the (unknown) state derivative at the next sampling instant in the construction of the interpolation algorithm.

The numerical ODE solver converts the original differential equations to equivalent difference equations. The algorithm designer needs to worry about issues of numerical stability (errors do not accumulate over time) and accuracy (the numerical solution of the difference equations approximates well the analytical so-

[1]This work was partially supported by Agencia Nacional de Promoción Científica y Tecnológica (ANPCYT) of Argentina under grant PICT 31653

lution of the differential equations).

Not every algorithm is equally well suited for simulating all systems. Stiff systems are systems with Jacobians, whose eigenvalues are spread widely along the negative real axis of the complex plane. They require special classes of implicit algorithms for their solution. Marginally stable systems are systems with Jacobians, whose dominant eigenvalues are spread widely up and down along the imaginary axis of the complex plane. They also require special classes of numerical algorithms for their solution.

Many books have been written that deal with these issues explicitly and extensively, including one by the authors of this paper [1].

QUANTIZED STATE SYSTEM SOLVERS

Time slicing is not the only possible approach to numerically solving a set of ODEs on a digital computer, although it is by far the most commonly used approach. In this paper, we shall present another discretization method. Instead of discretizing time, the Quantized State System (QSS) algorithms discretize the state values, while keeping time continuous.

Hence whereas a time-slicing method attempts to answer the question:

> Given that the state value at time t_k is equal to x, what is the state value at time $t_{k+1} = t_k + \Delta t$?

a QSS method provides an answer to the following modified question:

> Given that the state has a value of x_k at time t, what is the earliest time instant, at which the state assumes a value of $x_{k\pm1} = x_k \pm \Delta x$?

The QSS approach to numerical ODE solution describes thus not a single algorithm, but rather an entire class of algorithms that is, at least conceptually, as rich and varied as that of the time-slicing algorithms.

Of course, since the QSS approach is of a fairly recent vintage, the currently available QSS codes are not yet as sophisticated as the classical numerical ODE solvers.

Currently available are explicit, asynchronous, variable-step, fixed-order algorithms of orders one through three (QSS1 [5], QSS2 [3], and QSS3 [4]), an (also explicit!) stiff system solver of order one (BQSS [6]), and a geometric solver for dealing with marginally stable systems also of order one (CQSS). BQSS2 and CQSS2 algorithms are currently under development.

QSS algorithms have some striking properties that make them interesting for the solution of a variety of different problems, including the simulation of ODE systems with heavy discontinuities, for real-time simulation of stiff systems, and for parallel simulation on multiple processors, e.g. in embedded system design.

THE QSS1 ALGORITHM

Given an ODE system in state-space form:

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u}, t) \tag{1}$$

where $\mathbf{x} \in \mathcal{R}^n$ is the state vector, $\mathbf{u} \in \mathcal{R}^m$ is the input vector, and t denotes time.

We approximate the ODE system in the following fashion:

$$\dot{\mathbf{x}} = \mathbf{f}(floor(\mathbf{x}), floor(\mathbf{u}), t) \tag{2}$$

where the $floor()$ function replaces the argument by the next lower quantized value of the argument. If $\Delta x_i = 1.0$, then $floor(x_i)$ denotes the conventional round-off operation, i.e., x_i gets rounded off to the next integer value below.

Evidently, as long as none of the states or inputs crosses through the next threshold, all state derivatives remain constant, and the states are linear functions of time. If $\dot{x} > 0$, x increases linearly toward x_{k+1}, and if $\dot{x} < 0$, x decreases linearly toward x_k. In both cases, the time at which the upper or lower threshold is being reached can be computed explicitly. If $\dot{x} = 0$, x remains constant, and never reaches either threshold.

The time, at which a state variable reaches its next threshold is different for separate states. Hence QSS algorithms are naturally asynchronous. Each integrator proceeds at its own pace. State variables with large gradients will pass through their thresholds more frequently than states with small gradients.

If a state or input variable passes through its next threshold, that information must be passed on to those integrators that process state equations that depend on the variable in question, i.e., information is passed selectively to those integrators only that depend on this information.

QSS algorithms do not convert ODE systems to equivalent difference equation systems, but instead, convert the continuous-time model to an equivalent discrete-event model that can be simulated using any suitable discrete-event simulation engine, such as DEVS [7].

Unfortunately, the algorithm as described up to now doesn't always work. It frequently leads to illegitimate discrete-event models, i.e., to models that switch infinitely often within a finite time period.

Let us discuss the following simple first-order model:

$$\dot{x} = -floor(x) - 0.5 \tag{3}$$

with initial condition $x(t = 0) = 0.0$. Initially, $floor(x) = 0$, and therefore $\dot{x} = -0.5 < 0$. Thus, the state switches immediately to $floor(x) = -1$. However now, $\dot{x} = +0.5 > 0$, and the state switches immediately back to $floor(x) = 0$.

In order to avoid the problem of illegitimacy of the resulting discrete-event model, we need to introduce hysteresis into the discretization. Thus, the $floor()$ function must now be implemented as shown in Fig. 1, where $q(t) = floor(x(t))$.

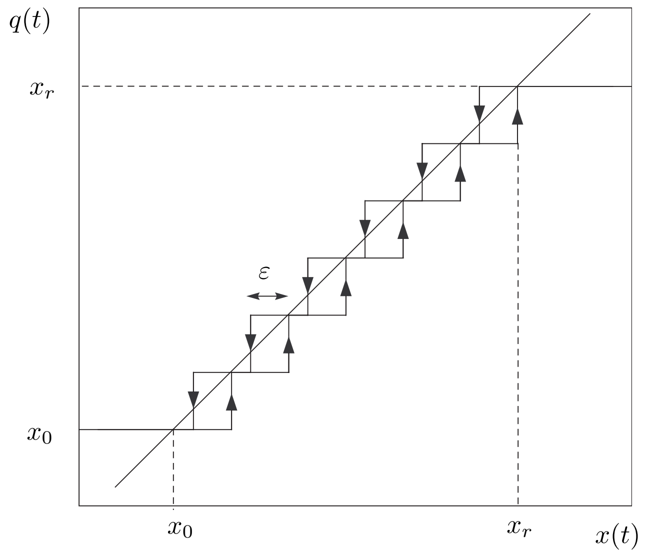

Figure 1: Hysteretic discretization

The hysteresis prevents the generation of illegitimate models. The smaller the hysteresis width is chosen, the higher may be the resulting switching frequency, but the switching frequency can never become infinite. It is usually best to choose the same value for the quantization height and for the hysteresis width, i.e., $\Delta x = \varepsilon$.

THE QSS2 and QSS3 ALGORITHMS

Let us now try to simulate the simple model:

$$\dot{x} = 1.0 \tag{4}$$

The model is so simple that the numerical ODE solver should hopefully be able to solve it within a single step. Yet, QSS1 is not capable of solving this problem efficiently. Each time, the state x passes through the next threshold, a discrete event is being issued, although the state derivative doesn't depend on it.

This is the problem with a first-order accurate QSS algorithm. The number of events grows inverse proportional to the discretization, Δx. When we make Δx ten times smaller, we generate ten times more events.

We can improve the situation using the following idea. Until now, we let the discretized states:

$$q_i(t) = floor(x_i(t)) \tag{5}$$

be constant. Consequently, also the state derivatives were constant in between events. We now allow the discretized states to be linear functions. Unfortunately, as the state derivatives:

$$\dot{x}_i = f_i(\mathbf{q}, t) \tag{6}$$

are nonlinear functions of the discretized state vector $\mathbf{q}$, the derivatives are not necessarily linear themselves. However, we can linearize the nonlinear functions at the time of the event around the current state, thereby forcing the state derivatives to be linear.

As $\dot{x}_i$ is a linear function, it can be integrated analytically. the state variable x_i is thus a parabolic function.

An event will be scheduled whenever the parabolic $x_i(t)$ deviates from the linear $q_i(t)$ by more than Δx_i.

This method, called QSS2, solves the trivial problem of Eq.(4) in a single integration step. The method is second-order accurate, and the number of events grows inverse to the square root of the discretization. If we choose Δx 100 times smaller, we only generate 10 times as many events as before.

In the same fashion, we also implemented a third-order accurate QSS3 method. The number of events of QSS3 grows inverse proportional to the cubic root of the discretization.

SIMULATING ACROSS DISCONTINUITIES

A problem that occurs frequently in the simulation of technical systems is that the state equations contain discontinuities.

As time slicing algorithms are based on interpolation polynomials that don't ever exhibit discontinuities, such algorithms cannot integrate across discontinuities. Instead, the discontinuity is being formulated as a state event [1], and an iteration algorithm (root solver) is invoked, whenever the solution crosses through a state event detector function to locate the time instant of the discontinuity accurately.

Thus, the location of discontinuities can slow down the simulation significantly due to the necessary iteration, and in the case of real-time simulation, an accurate location of state events may not be feasible at all, as this would cause massive over-runs.

To avoid the need of iterating on state events, some integration algorithms use a dense output feature instead. These algorithms are able to calculate the solution not only at the sampling instants with full accuracy, but also at arbitrary time instants in between sampling instants.

An integration algorithm that offers dense output allows the user to replace the iteration on state events by interpolation. Yet, there is computational overhead associated with providing dense output, and even the interpolation itself may be too costly.

QSS algorithms don't have this problem. Determining when a solution crosses through a given threshold is what these algorithms "do for a living." There is zero overhead associated with locating state events.

Hence QSS algorithms are particularly good at simulating systems with heavy discontinuities, such as switched power converter circuits, efficiently and accurately.

STIFF SYSTEM SOLUTION

Let us now look at the simulation of stiff systems.

Stiff systems require special integration algorithms, so-called stiff system solvers, because other algorithms lose their numerical stability for large time steps. Unless we use a stiff system solver, the largest time step that can be used is related to the location of the fastest eigenvalue of the Jacobian of the system, instead of being related to the fastest gradient in the solution vector.

Unfortunately, it can be shown that all stiff system solvers are invariably implicit algorithms [1], although not all implicit algorithms are stiff system solvers.

Implicit algorithms, however, require a Newton iteration for each integration step, since the solver computes the future state from the future state derivative, whereas the model computes the future state derivative from the future state. This makes these algorithms unattractive for real-time simulation.

How do the QSS algorithms fare? It has been shown that QSS algorithms, when applied to an analytically stable model, cannot ever become numerically unstable. However, when we apply a QSS algorithm to a stiff system, we frequently obtain high-frequency oscillations in some of the state variables, i.e., the algorithm "adjusts" its step size to a very small value, equivalent to step sizes that a classical explicit solver would need to use in order to preserve numerical stability. Hence neither of the three algorithms introduced up to now can be used to simulate stiff systems.

Let us now propose an "implicit" QSS algorithm. Instead of using the current discretized state in the computation of the state derivative, we shall use the next (anticipated) discrete state for this purpose.

We now have an implicit algorithm, because the solver computes the future state from the derivative, and the model computes the derivative from the future state.

However, the algorithm doesn't require any iteration, because there are only two possible future states. The next state is either the current state plus one discretization level, or it is the current state minus one discretization level. No other options exist.

Hence we compute the derivatives under both assumptions. If the derivative is positive under both assumptions, we know that the state is increasing. If it is negative under both assumptions, we know that it is decreasing, and if we get two different answers, we know that the derivative will be changing directions sometime during the next step. Hence we set the derivative to zero, and wait for the next event to happen elsewhere in the system.

This algorithm has been implemented. It has been coined BQSS (backward QSS). It is a stiffly stable algorithm. It adjusts its step sizes with the sizes of the actual gradients in the system, and not with the locations of the eigenvalues of the Jacobian. BQSS doesn't lead to high-frequency oscillations when simulating stiff systems.

It is conceptually an implicit algorithm, but as there are only two choices for the next state, BQSS doesn't require an iteration, and in fact, the computational overhead in comparison with QSS is rather small. When we apply BQSS to a non-stiff system, it simulates maybe 10% slower than QSS.

For real-time simulation, we usually prefer to use low-order algorithms, because we are dealing with unknown input functions that need to be sampled frequently. For this reason, no one ever uses a high-order algorithm for real-time simulation. Most people use either forward Euler (FE), FE with a correction term to make the algorithm second order accurate, or possible an Adams-Bashforth 3rd order (AB3) algorithm.

Unfortunately, neither of these algorithms is suitable for simulating stiff systems. Hence if a stiff system is to be simulated in real time, we have a problem. Backward Euler (BE), on the other hand, is too inefficient. The iteration overhead can hardly ever be tolerated. A single step of BE is roughly three times as expensive as a single step of FE.

BQSS is the right tool for this type of problem. The algorithm is practically explicit, as it doesn't require an iteration. We can calculate the maximum computational load of a single step of BQSS, and thereby avoid ever obtaining over-runs.

Gustavo Migoni is dealing with stiff QSS solvers in his Ph.D. research. He is currently working on extending the BQSS algorithm to second and third orders. Unfortunately the extension is not so easy as in the case of the explicit algorithms. When a derivative exhibits opposite signs under the two assumptions, it no longer suffices to set it to zero. A BQSS2 algorithm has already been designed, but the corresponding code has not yet been fully implemented and debugged.

SIMULATING MARGINALLY STABLE SYSTEMS

Another problem that engineers and scientists are frequently dealing with is the simulation of Hamiltonian systems, i.e., systems without any damping. This is a subclass of the class of marginally stable systems, i.e., systems that have the dominant eigenvalues of their Jacobians spread up and down in the vicinity of the imaginary axis of the complex plane.

Most explicit algorithms have the tendency of considering these systems unstable, i.e., they add energy to a system that should merely preserve its energy. Stiff system solvers, on the other hand, have the tendency of considering these systems damped, i.e., they remove energy from the system, rather than preserving it.

Algorithms that are particularly well suited for simulating such systems are the F-stable algorithms, i.e., algorithms, whose numerical stability domain coincides precisely with the left half complex plane [1]. The simplest F-stable algorithm is the trapezoidal rule.

Is there such a thing as an "F-stable" QSS algorithm? Unfortunately, the concept of a numerical stability domain doesn't carry over easily to the class of QSS algorithms. Hence we need to go another route.

Let us simulate the simple Hamiltonian problem:

$$\begin{aligned} \dot{x}_1 &= x_2 \\ \dot{x}_2 &= -x_1 \end{aligned} \tag{7}$$

We should obtain a sine wave. When we use QSS1, the amplitude of the sine wave is slowly increasing. When we use BQSS, it is slowly decreasing. This is what we would expect to happen.

Hence we tried calculating the derivatives once using QSS and once using BQSS, and we used one half of each. This is also very inexpensive. We called the algorithm CQSS (centered QSS).

We obtained a stable sine wave oscillation. But why did this work?

Luckily for us, Hairer and Wanner have developed a beautiful theory for algorithms that are designed to solve precisely this kind of problems [2]. Such algorithms, called geometric algorithms, preserve geometric properties such as symmetry, energy, symplecticity, etc..

A symmetric algorithm is one that exhibits the following property. If we integrate across one step forward in time, then reverse time and integrate across the same step backward in time, we need to end up at exactly the same spot from where we started.

It can be easily shown that CQSS is indeed a symmetric algorithm, whereas neither QSS nor BQSS are symmetric algorithms.

The second property is more tricky, because it applies to a system of at least two state equations, rather than to an individual state equation.

In a synchronous algorithm, we can check the energy at the beginning and at the end of the step. Yet in an asynchronous algorithm, this doesn't work as easily. The two integrators operate on separate clocks.

Mario Bortolotto started recently working on geometric QSS solvers in his Ph.D. research. The problem of proving the conservation property has meanwhile been solved, but the results haven't been published yet. Hence we cannot offer a reference to the corresponding publication yet, and the proof is too mathematical to include it right here. Mario plans on extending CQSS also to second and third orders, but this work is still in its infancy.

DISRIBUTED SIMULATION ON PARALLEL ARCHITECTURES

Real-time simulation often involves distributing the simulation across multiple processors. It is advantageous to run such simulations asynchronously.

When using QSS algorithms, running asynchronously comes naturally. Such algorithms operate asynchronously even when executed on a single processor. Hence it is much easier, distributing a QSS simulation over multiple processors than a classical simulation.

When distributing a simulation across multiple processors, the communication overhead must be minimized. It is thus important to communicate as seldom as possible, and when a communication is needed, minimize the amount of information that has to be transmitted.

QSS algorithms do well on both fronts.

QSS algorithms only communicate when a state variable crosses through its next threshold, i.e., at event times, and they communicate asynchronously, i.e., they communicate only to those other neighboring integrators that need the information. Hence the communication frequency is kept as low as possible.

Also, when a state variable communicates its event, all it needs to tell the neighboring integrators is whether it increases its level by one, or whether it decreases it by one. Hence 1 bit is sufficient. There is never any need to communicate complete state information.

CONCLUSIONS

In this paper, a new class of numerical ODE solvers has been presented. These algorithms represent a drastic departure from the classical numerical ODE solvers, in that they are based on state quantization instead of time slicing.

The article looks at these algorithms with a bird's eye's view, i.e., it aims at explaining the basic concepts and ideas behind these methods, rather than going into much mathematical detail.

Yet, rigorous mathematical descriptions of these algorithms have been recently published and are available. Both the numerical stability and accuracy conditions have been analyzed thoroughly.

It has been established that an analytically stable system cannot become numerically unstable when simulating it using a QSS algorithm. What may happen is that the algorithm produces high-frequency oscillations, which is equivalent to simulating using a very small step size.

In addition, a global error bound has been established, i.e., simulation accuracy can be guaranteed not only locally, but even globally. Unfortunately, the error bound is often conservative, i.e., the simulation results may in reality be quite a bit more accurate than what the error bound indicates.

A simulation tool, PowerDEVS, has been made available [1] that implements the QSS algorithms using a DEVS simulation engine [7].

References

[1] F.E. Cellier and E. Kofman. *Continuous System Simulation*. Springer, New York, 2006.

[2] Ernst Hairer, Christian Lubich, and Gerhard Wanner. *Geometric Numerical Integration Structure-Preserving Algorithms for Ordinary Differential Equations*. Springer, 2002.

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

[4] E. Kofman. A Third Order Discrete Event Simulation Method for Continuous System Simulation. *Latin American Applied Research*, 36(2):101–108, 2006.

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

[6] G. Migoni, E. Kofman, and F. Cellier. Integración por Cuantificación de Sistemas Stiff. *Revista Iberoamericana de Automática e Informática Industrial*, 4(3):97–106, 2007.

[7] B. Zeigler, T.G. Kim, and H. Praehofer. *Theory of Modeling and Simulation. Second edition.* Academic Press, New York, 2000.