Simulation of Smart–Grid Models using Quantization–Based Integration Methods

X. Floros[a1] F. Bergero[b1] N. Ceriani[c] F. Casella[c] E. Kofman[b] F. E. Cellier[a]

[a]Department of Computer Science, ETH Zurich, Switzerland
[b]CIFASIS-CONICET, Rosario, Argentina
[c]Politecnico di Milano, Dipartimento di Elettronica, Informazione e Bioingegneria, Italy

Abstract

Concepts such as smart grids, distributed generation and micro–generation of energy, market–driven as well as demand–side energy management, are becoming increasingly important and relevant as emerging trends in the design, management and control of energy systems. Appropriate modeling and design, efficient management and control strategies of such systems are currently being studied. In this line of research a very important enabling component is efficient and reliable simulation. However those energy models are typically large, stiff and exhibiting heavy discontinuities, and at the same time consist of interconnected multi–domain subsystems encompassing electrical, thermal, and thermo-fluid models. Object-Oriented (O–O) languages such as Modelica are obviously well-suited for the modeling of such systems; however, traditional state-of-the-art hybrid differential algebraic equation solvers cannot efficiently simulate these systems especially when their size grows to the order of hundreds, thousands, or even more interconnected units.

The goal of this paper is to show, through a couple of exemplary case studies, that Quantized State System (QSS) integration methods are ideally suited to solve models of such systems, as they scale up better than traditional methods with the system size, and provide time savings of several orders of magnitude, while achieving comparable numerical precision.

Keywords: *Quantization–Based Integration Methods, QSS, DASSL, Smart–Grids, EnergyMarket, Modelica*

[1]The authors contributed equally to this work.

1 Introduction

The growing interest in new paradigms for energy systems such as Smart Grids (SG) is posing new challenges in the control of procurement, conversion, distribution and use of energy to meet environmental and economic objectives.

Computer simulation of SG systems is a fundamental tool for production planning and control, price regulation, logistics, etc. To carry out the simulation one must deal with two problems. First, modeling complex SG systems involves taking into account components from various domains such as thermal, electrical, ventilation, etc. Each component could be developed by different specialists, possibly using different languages or formalisms that must then be coupled to produce the complete model. SG models are commonly composed of energy production facilities, energy transmission networks and usually hundreds or thousands of energy consumption units. Thus the modeling of these types of systems is a difficult task. Second, once the problem is modeled, the actual simulation of a large hybrid model (with continuous and discrete subcomponents) can turn out to be prohibitively expensive in terms of CPU time, as the scale of the system grows.

Modelica [13] is an Object–Oriented (O–O) language for modeling and simulation of complex multi-domain physical systems, described by hybrid differential-algebraic equations. In the literature several research efforts show the use of Modelica as a language for modeling SG problems [6, 7, 8, 19, 20, 21]. State-of-the-art Modelica simulation tools generate simulation code that solves the differential equations using classical numerical integration methods, such as Euler, Runge-Kutta, or DASSL, which are based on time discretization.

Another approach is the use of Quantized State System (QSS) methods [4, 14, 15] which replace time discretization by state quantization. The QSS methods have certain features (sparsity exploitation, efficient discontinuity handling, explicit stiffness treatment) that make them particularly effective for large, sparse, hybrid, dynamical systems like the SG models.

In this work we investigate the suitability of the QSS methods for simulating SG models described in Modelica and compare their efficiency, as well as simulation quality, against classical integration methods.

We shall focus on two models, first a District Cooling System taken from [5] and then an Energy Market with houses as energy consumption units adapted from [6].

The paper is organized as follows: Section 2 introduces the main concepts used along the article, then Section 3 describes the two smart–grid applications, and in Section 4 we compare different numerical integration methods. Finally, Section 5 concludes the article and outlines future work.

2 Background

This Section introduces the main concepts used along the remainder of the article.

2.1 Classical Numerical Integration Methods

The mathematical models describing Energy Management problems such as the SG are usually time dependent dynamical systems. These can be expressed in the form of a set of Differential Algebraic Equation (DAEs) or directly in a set of Ordinary Differential Equations (ODEs) as:

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

where $\mathbf{x}(t)$ is the state vector.

Classical numerical integration methods discretize the time variable computing the states variables for certain time points.

We shall focus on two well–known numerical integration methods [4]:

Runge-Kutta An explicit variable step algorithm of fourth order.

DASSL An implicit variable step algorithm based on a series of Backward Difference Formulae (BDF) of different orders of approximation accuracy.

As both methods are based on time discretization, discontinuity detection is an expensive mechanism. Also, only implicit algorithms are able to efficiently simulate stiff systems, i.e. systems that exhibit simultaneous fast and slow dynamics, without resorting to unnecessarily short time steps.

2.2 QSS Integration Methods

Quantized State System (QSS) methods replace the time discretization of classic numerical integration algorithms by the quantization of the state variables.

Given the ODE of Eq.(1), the first order Quantized State System method (QSS1) [16] approximates it by

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

Here, $\mathbf{q}$ is the *quantized state vector*. Its entries are component-wise related with those of the state vector $\mathbf{x}$ by the following *quantization function*:

$$q_j(t) = \begin{cases} x_j(t) \text{ if } |x_j(t) - q_j(t^-)| \geq \Delta Q_j \\ q_j(t^-) \text{ otherwise} \end{cases} \tag{3}$$

where ΔQ_j is called *quantum* and $q_j(t^-)$ denotes the left-sided limit of q_j at time t. The quantization function $\mathbf{q}(t)$ in QSS methods also contains a *hysteretic* term (not shown here for simplicity) that is necessary in order to avoid illegitimate models [16].

It can be easily seen that $q_j(t)$ follows a piecewise constant trajectory that only changes when the difference between $q_j(t)$ and $x_j(t)$ becomes equal to the quantum. After each change in the quantized variable, it results that $q_j(t) = x_j(t)$.

The QSS1 method has the following features:

- In the solution, the quantized states $q_j(t)$ follow piecewise constant trajectories.
- The state variables $x_j(t)$ follow piecewise linear trajectories.
- The state and quantized variables never differ by more than the quantum ΔQ_j. This fact ensures stability and global error bound properties [4, 16].
- The fact that the state variables follow piecewise linear trajectories makes the detection of discontinuities a trivial task. Moreover, after a discontinuity is detected, its effects are no different

DOI 10.3384/ECP14096787

from those of a normal step (because changes in q_j are discontinuous). Thus, QSS1 is very efficient in simulating discontinuous systems [14].

- Each step is local to a state variable x_j (the one that reaches the quantum change), and it only provokes evaluations of the state derivatives that explicitly depend on it. This fact implies that QSS1 performs intrinsic sparsity exploitation.
- If some state variables do not change significantly, they will not provoke any step or evaluation at all. This feature reinforces the efficient sparsity exploitation.

The last two points show that QSS methods integrate each state variable at its own pace i.e. a fast changing variable would provoke more local integration steps than a slow one.

As QSS1 only performs a first order approximation, good accuracy cannot be obtained without a significant increment in the number of steps. Also as QSS1 is an explicit solver, the algorithm is not suitable for simulating stiff systems. The former limitation was solved with the introduction of higher order QSS methods like QSS2 and QSS3 [15]

For the simulation of stiff systems, a family of linearly implicit QSS methods (LIQSS) of orders 1 to 3 was also proposed in [17]. LIQSS methods are semi–implicit methods that can handle certain types of stiff systems.

In the context of this work, the efficient sparsity exploitation, the the semi-implicit treatment of stiff systems and the native handling of frequent discontinuities compose the main advantages of the QSS methods.

2.3 Stand–Alone QSS Solver

It was shown that the behavior of the QSS approximation of Eq.(2) can be described as a Discrete EVent System (DEVS)[22]. Thus, a straightforward implementation of these algorithms is through their equivalents in a DEVS simulation engine.

DEVS–based implementations of QSS methods are simple but they are not efficient. The problem is that the DEVS simulation engines waste a large amount of the computational load attending the DEVS simulation mechanism. This fact motivated the development of a stand-alone QSS solver.

Recently, the complete family of QSS methods was implemented in a *stand–alone QSS solver* coded in plain C language [9]. This solver simulates models described in a subset of the Modelica language, called μ-Modelica [3].

In this work all simulations are performed using the stand–alone QSS solver.

2.4 OpenModelica

OpenModelica [12] is an open-source Modelica-based modeling and simulation environment. OpenModelica offers different numerical integration methods for simulation, amongst them, the before mentioned DASSL and Runge-Kutta. In this article we shall use this tool as a reference to compare the performance of different integration methods.

2.5 Related Work

The goal of this article is to study the application of QSS methods to Smart Grid problems described in the Modelica language. To the best of the authors' knowledge, this problem has not previously been studied.

The application of QSS methods to Modelica models was studied in [3, 10, 11], showing the benefits of QSS methods for problems with frequent discontinuities. Also the use of QSS methods (not using Modelica) for a large hybrid sparse load management problem was studied in [18].

The use of the Modelica language for Energy Management problems was studied in [6, 7, 8, 19, 20, 21] showing the powerful advantages that the language offers to this set of models.

An advanced control system for the optimal energy management of a building cooling system is studied in [5]. In this system, a centralized facility produces chilled water that is then distributed among a certain number of thermal zones (e.g. small houses or apartments). The optimal control algorithm requires multiple simulations of the whole system model for different parameter settings. A simplified, equation-based version of that model is presented in this paper. Although the original system was designed for a reasonably low number of users (5 to 20, i.e., a micro-grid), this simplified model is representative of a larger class of systems with a centralized heat or cooling source, and many end users with their independent control systems. Such systems can easily scale up to contain hundreds, thousands, or even more individual units. The controllers have also been simplified in this work, as the achieved speed-up factor is not depending on the specific control laws, but rather on the efficient way the QSS algorithm exploits

the sparsity and weak coupling of the system model, combined with the efficient event handling.

3 Case Studies

In this section we present two case studies. First a District Cooling System and second an Energy Market model. As stated before, the Modelica language is suitable for describing multi-physics and multi-energy problems like SG models. In fact there are Modelica libraries for modeling energy problems that help the development process.

The QSS stand–alone solver accepts models described in a subset of the Modelica language, therefore all models used in the article are coded in this simplified language called μ-Modelica. Being a subset of the complete language, μ-Modelica is accepted by all Modelica simulation tools enabling us to simulate the same model both in OpenModelica and in the QSS stand–alone solver. For more information regarding the transformation of Modelica to μ-Modelica models, we refer the reader to [3].

3.1 Case Study I: A District Cooling System

As mentioned above, the District Cooling System is adapted from [5] and consists of the following elements:

- The *Cooling Plant* that generates cooling power used to control the temperature of a cooling load.
- The *Chilled Water Circuit* that connects the cooling plant to the cooling load allowing heat transfer between the two.
- The *Cooling Load* that transfers heat to the chilled water circuit. The load is composed by a group of zones affected by heat exchange with the outside ambient and by internal heat gains such as occupants and office equipment. The chilled water circuit exchanges heat with the zones by means of fan coils.
- The *Chilled Water Temperature Controller* that keeps circuit temperature at a specified set-point. The control variable is the cooling plant cooling power set-point.
- The *Zone Temperature Controller* that keeps each zone at the desired temperature. The control variable is the fan coil valve opening.

In the following paragraphs, the model adopted for each element is described.

Cooling Plant The cooling plant is simplified in this article to deliver exactly the energy needed for the Chilled Water Circuit set-point Q_{CSP}.

Chilled Water circuit Chilled water circuit dynamics are described using a lumped RC model. The following power balance equation can be written:

$$C_{CW}\frac{dT_{CW}}{dt} = \sum_{i}^{N}(Q_{ZA\,i}(t) - Q_C(t))$$

where T_{CW} is the circuit temperature and C_{CW} its thermal capacity. $Q_{ZA\,i}$ is the heat exchanged with the i-th zone and $Q_C(t)$ is the cooling power contribution provided by the Cooling Plant. Heat losses in the circuit are neglected.

Cooling Load Zones are modeled as lumped RCs as well. Their power balance equation is:

$$C_{ZA}\frac{dT_{ZA\,i}}{dt} = -Q_{ZA\,i} + k_{out}(T_{OA}(t) - T_{ZA\,i}(t)) + Q_{INT\,i}(t)$$

where $T_{ZA\,i}$ is the zone temperature and C_{ZA} its thermal capacity, $Q_{ZA\,i}$ is the heat exchanged with the chilled water circuit, $X_{C,Z\,i}$ is the heat exchanger valve opening, and $Q_{INT\,i}$ is the heat produced by zone occupants. $Q_{ZA\,i}$ evolves according to the following expression:

$$\tau_{ex}\cdot\dot{Q}_{ZA\,i}(t) + Q_{ZA\,i}(t) = X_{C,Z\,i}(t)k_{cw}(T_{ZA\,i}(t) - T_{CW}(t))$$

where τ_{ex} is the heat exchanger time constant. It is worth noticing that the introduction of the exchanger dynamics has the twofold purpose of a more accurate modeling and of obtaining a stiff model in order to test the QSS solver's performance in such conditions. $Q_{INT\,i}$ is modeled according to the following polynomial function of the zone temperature:

$$Q_{INT\,i}(t) = (p_1 T_{ZA\,i}^2(t) + p_2 T_{ZA\,i}(t) + p_3)n_{\text{people}\,i}(t),$$

where $n_{\text{people}\,i}$ is the number of zone occupants. Such a model is proposed in [2] where suitable coefficient values p_1, p_2 and p_3 can be found.

The number of occupants is generated by a simple stochastic model: the next arrival/departure time of a person is given by a fixed time (1000 seconds) plus a uniform random variate between 0 and 1000 seconds. At each event, a person either comes in or leaves with

DOI 10.3384/ECP14096787

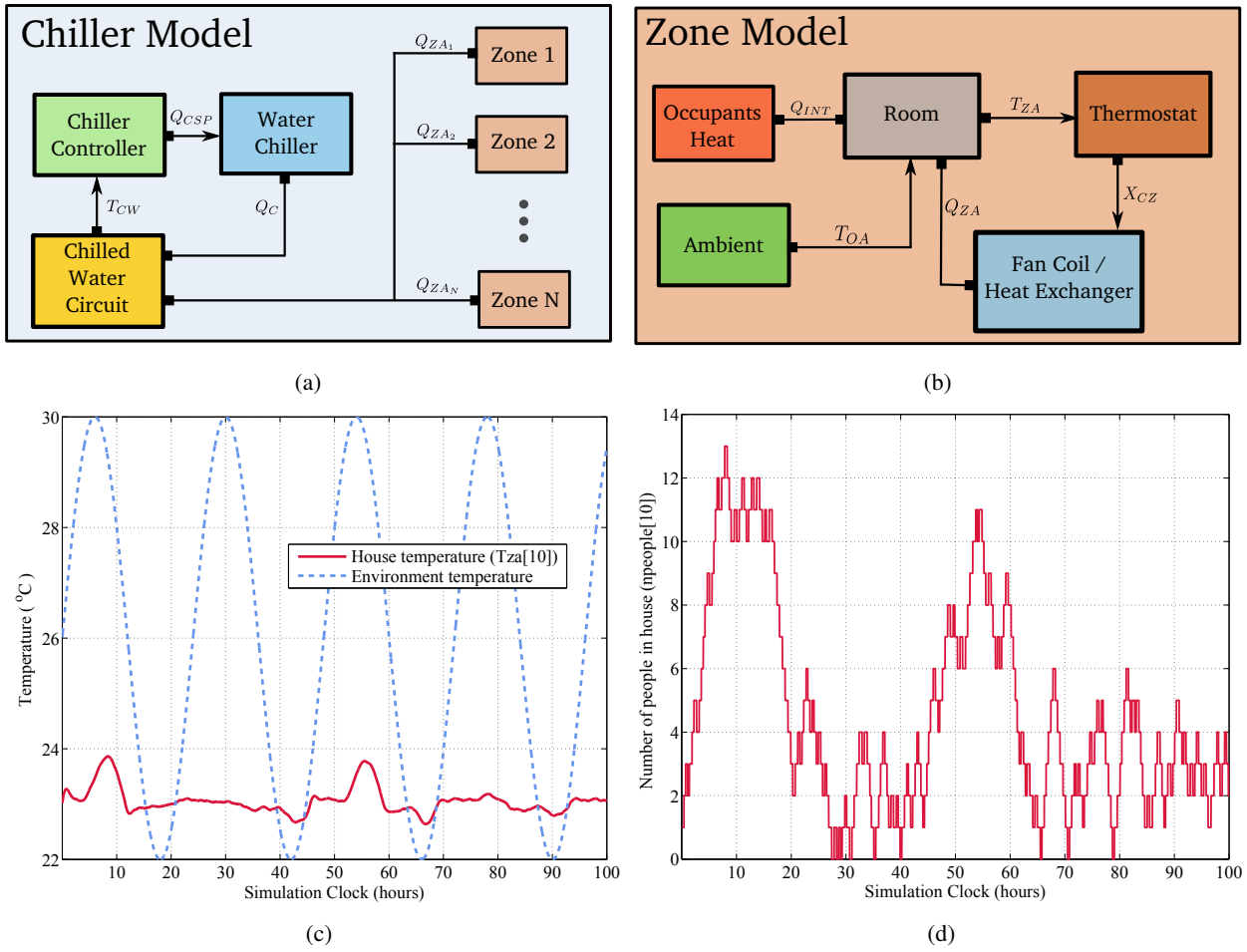


Figure 1: District Cooling System model graphical representation (a) and, in more detail, the submodel used for each zone (b). In (c) the simulated trajectory of the temperature state variable in zone 10 is plotted against the ambient temperature, while in (d) the number of the people being present in zone 10 is depicted over time.

a 50% probability. Although this model does not represent any specific realistic pattern for the coming and going of occupants, it serves the purpose of generating a number of *uncorrelated* time events, whose density in time grows with the number of zones. Any realistic system will exhibit this kind of behavior, as events happening in different buildings will usually be uncorrelated, no matter what their actual probability distribution is.

Chilled Water Temperature Controller A PI controller is adopted to control chilled water circuit temperature. The control variable is the power set-point for the cooling plant Q_{CSP}. Since no dynamics are considered for the chillers, the power set-point coincides with the actual power generated. The controller is modeled as follows:

$$\dot{z}_{C,CW}(t) = -\frac{k_{I,CW}}{k_{P,CW}} \cdot z_{C,CW}(t) + \frac{k_{I,CW}}{k_{P,CW}} \cdot Q_{CSP}(t)$$
$$Q_{CSP}(t) = \Phi_{[0,Q_{C,max}]}\left(k_{P,CW} \cdot e_{C,CW}(t) + z_{C,CW}(t)\right)$$
$$e_{C,CW}(t) = T_{CW}(t) - T_{CW\,SP}(t)$$

where $z_{C,CW}(t)$ is the integral state, $k_{P,CW}$ and $k_{I,CW}$ are the PI gains, and $\Phi_{[a,b]}(\cdot)$ is the saturation function:

$$\Phi_{[a,b]}(x) = \begin{cases} a, & x < a \\ x, & x \in [a,b] \\ b, & x > b. \end{cases}$$

Zone Temperature Controller Each zone temperature is controlled by a PI controller as well. The control variable is the heat exchanger valve opening

$X_{C,Z}$, spanning the range $[0,1]$.

$$\dot{z}_{C,Z}(t) = -\frac{k_{I,Z}}{k_{P,Z}} \cdot z_{C,Z}(t) + \frac{k_{I,Z}}{k_{P,Z}} \cdot X_{C,Z}(t)$$
$$X_{C,Z}(t) = \Phi_{[0,1]}\left(k_{P,Z} \cdot e_{C,Z}(t) + z_{C,Z}(t)\right)$$
$$e_{C,Z}(t) = T_{ZA}(t) - T_{ZA\,SP}(t),$$

Model scaling The presented model is designed to scale with the number of zones N, while providing a reasonable behavior for all controlled variables. For this purpose, certain model parameters are proportional to the number of zones N. In particular, the cooling plant maximum power $Q_{C,max}$ and the cooling plant controller (chilled water temperature controller) gains $k_{I,CW}$ and $k_{P,CW}$ are proportional to N. The chilled water circuit thermal capacity C_{CW} is also linearly scaled with the number of zones N.

3.2 Case Study II: An Energy Market

The Energy Market model was introduced in [6] as a typical toy model written in Modelica and capturing many of the aspects typically found in realistic smart grid applications. The interactions between the different components in the Energy Market model are graphically sketched in Fig. 2. The model consists of the following components:

Environment We assume that the temperature of the environment is given by a sinusoidal function:

$$T_{amb} = \overline{T_{amb}} + \Delta T \sin(\omega t + \phi)$$

where the mean temperature T_{amb} is set to 10^oC, while the frequency and offset are selected such that the minimum temperature is reached every midnight.

Heaters Each house has a heater that is controlled by an agent that switches it on and off, according to:

$$\dot{Q}_i^{heater} = \begin{cases} 0 & \text{if } T_i > T_i^{max} \\ P_{heat} & \text{if } T_i < T_{min} \end{cases} \quad (4)$$

Walls Each house has one wall that acts as a thermal resistor with the heat flow given by:

$$\dot{Q}_i^{wall} = \frac{1}{R_{th}}(T_i - T_{amb}), \quad (5)$$

where R_{th} is the thermal resistance of the wall.

Windows We assume that each house has one window that exhibits a stochastic behavior. More specifically, we assume that the opening time of each window is drawn randomly from a uniform distribution. It is closed again a random amount of time later.

$$openNextT[i] \sim \mathscr{U}(pre(openNextT[i]) + 1000, 50)$$
$$closeNextT[i] \sim \mathscr{U}(openNextT[i]) + 100, 200),$$

Each time a window is open, heat is exchanged between the environment and the house according to:

$$\dot{Q}_i^{window} = G(T_i - T_{amb}), \quad (6)$$

where G is a large heat conductance constant.

Agents Each house has a simple controller that controls the heater settings optimizing the power consumption. The agent turns the heater on at a lower goal temperature T_{min} and turns it off at an upper temperature T_{max}. If the energy price calculated in the energy market exceeds a threshold p_{max}, the agent decreased the upper level T_{max} to T_{max}^{alt}.

$$T_i^{max} = \begin{cases} T_{max} & \text{if } p < p_{max} \\ T_{max}^{alt} & \text{if } p \geq p_{max} \end{cases} \quad (7)$$

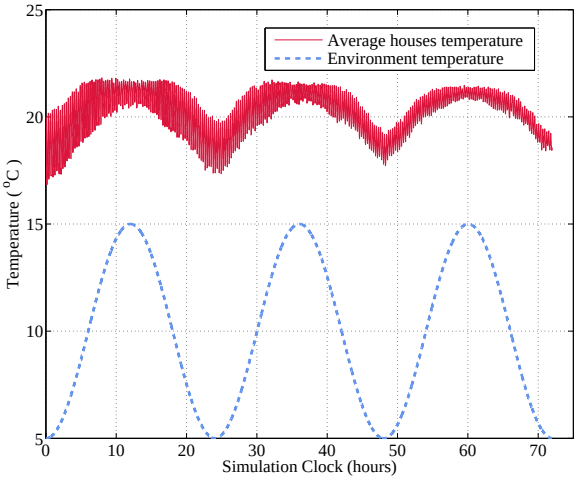
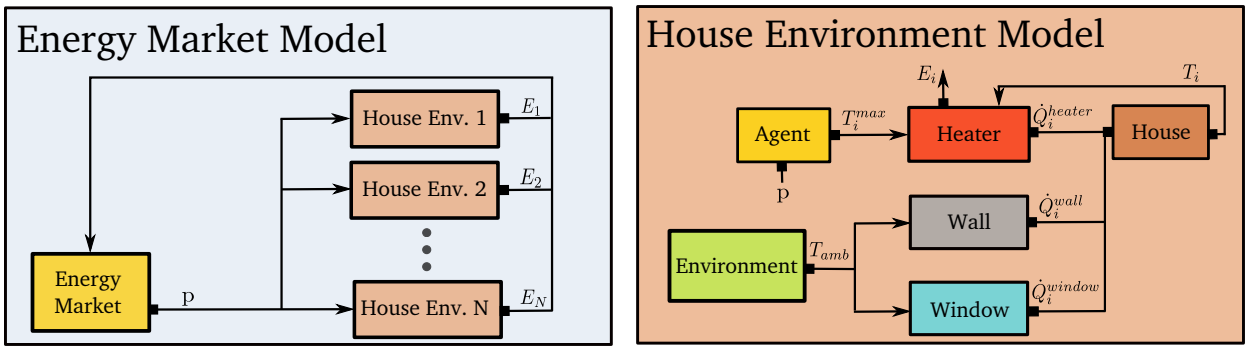


Figure 3: Simulated trajectories of the average temperature in the houses against the ambient temperature.

Houses Houses act as energy consumption units. The temperature inside each house is related to the heat flows described in the previous paragraphs with the following formula:

$$\dot{T}_i = \frac{1}{\rho V C_{th}}(\dot{Q}_i^{heater} - \dot{Q}_i^{window} - \dot{Q}_i^{wall}) \quad (8)$$

DOI 10.3384/ECP14096787



(a) (b)

Figure 2: Energy Market model graphical representation (a) and, in more detail, the submodel used for each house(b)

The energy consumed per unit of time for a specific house is the power of its heater integrated over a time unit, as given by:

$$E_i = \dot{Q}_i^{heater} \cdot t_{unit} \tag{9}$$

Energy Market The energy market component simulates the behavior of an energy price regulator for the whole network. According to the estimated energy price the house agents decide if they should reduce the energy consumption or not. Various energy price models could be employed, but for simplicity we choose to linearly relate the energy price to the mean energy consumed in the houses:

$$p = \overline{p} + p_1 \times \frac{1}{N} \sum_{i=1}^{N} E_i$$

4 Results

In this Section we show the simulation results for the two models presented before. Both μ-Modelica models can be downloaded from [1]. We compare the runtime efficiency and quality of the solutions obtained by different integration methods for different system sizes and tolerance values.

Simulation Benchmark

The simulation benchmark is as follows:

- Runge-Kutta and DASSL results were computed using OpenModelica 1.9.1 (r18381) (RML version).
- LIQSS2,3 and QSS3 results were computed using the QSS Stand Alone Solver from [9] r645.
- The simulation platform is a Dell 32bit desktop with a quad core processor @ 2.66 GHz and 4 GB of RAM.
- The Jacobian matrices of the presented models are quite sparse and banded. This information could be exploited by all algorithms to make the simulation more efficient, however this has not been investigated for DASSL and Runge-Kutta methods. The QSS methods exploit this fact natively without having to get any information on the structure of the Jacobian matrix.

Calculating the accuracy of the simulations can only be performed approximately, since the state trajectories of the models cannot be computed analytically. To estimate the accuracy of the simulation algorithms for a given setting, reference trajectories $(\mathbf{t}^{\mathbf{ref}}, \mathbf{y}^{\mathbf{ref}})$ were obtained using LIQSS3 with a tight tolerance of $1 \cdot 10^{-9}$ on an equidistant grid consisting of 5000 points. To calculate the simulation error, all methods were forced to output points on the same equidistant grid, without changing the integration step, thus obtaining simulated trajectories $(\mathbf{t}^{\mathbf{ref}}, \mathbf{y}^{\mathbf{sim}})$. Then, the mean absolute error is calculated as:

$$error = mean(|\mathbf{y}^{\mathbf{sim}} - \mathbf{y}^{\mathbf{ref}}|). \tag{10}$$

Regarding the error calculation we have to note that special care had to be taken in order to achieve comparable solutions from two independent runs of each model, since both models are based on the generation of random event sequences. To this end, we implemented a special random generator that, at every call, outputs the seed for its next call. Therefore, starting from the same seed, two independent model simulations generate the exact same sequence of random events.

The measured CPU time (simulation time) should not be considered as an absolute ground-truth since it will vary from one computer system to another, but the scaling of the algorithms as well as their relative ordering is expected to remain the same. Another important aspect is that, in order to objectively compare the simulation time needed by different algorithms we did not compare the time measurements of the algorithms for the same requested tolerance, but for the same achieved error.

4.1 Case Study I

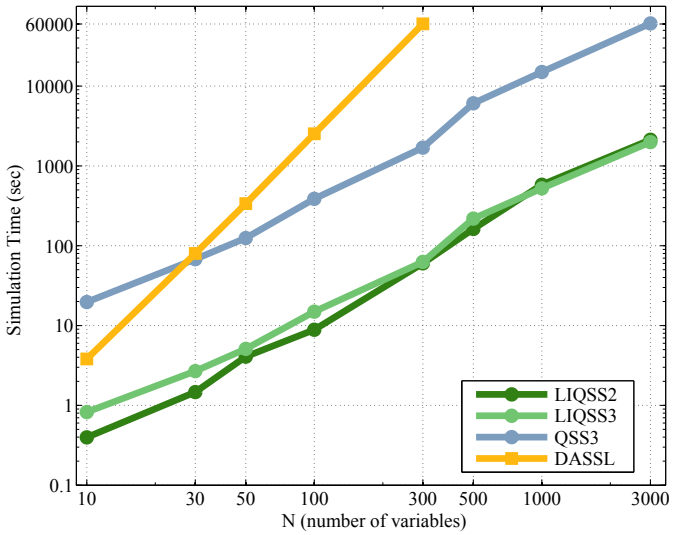


Figure 4: Simulation time for varying size of the District Cooling System model. All algorithms achieve a mean error on the order of 10^{-4}.

The District Cooling System model is a comparatively stiff model and this becomes apparent when comparing the measured CPU time of the QSS3 and LIQSS methods in Fig. 4. This is confirmed by estimating the different time constants. The time constants of the temperature controlled zone ($\sim 3,8 \times 10^3$ sec) and of the temperature controlled chilled water circuit ($\sim 7,5 \times 10^3$ sec) proved to be greater by three orders of magnitude than the time constant of the heat exchanger (~ 1 sec).

Regarding the scaling of the algorithms, Fig. 4 suggests that DASSL scales quadratically ($\sim 6N^2$), while QSS methods scale linearly with the number of variables ($\sim 3N$). Due to the time constraints for the present study we had to stop the DASSL experiments at N=300, but the linear scaling of the QSS methods allowed us to test their performance up to N=3000, thus for a 10 times larger model.

Besides time, another factor that prohibited us from performing further experiments with DASSL on larger models, was that the OpenModelica compiler failed to compile larger models. For the largest model, that all methods could simulate (N=300), the LIQSS methods are more than two orders faster than DASSL (10^2 sec compared to 600×10^2 sec).

Finally, a very interesting and useful aspect of the QSS methods is that they exhibit a strong correlation between the requested tolerance and the achieved error. This correspondence is depicted in Fig. 5a for the LIQSS2 method (the other QSS methods exhibit similar behavior). In contrast, the performance of DASSL was only slightly affected by changing the requested tolerance. This is a very important feature of the QSS methods as it allows the user to exploit the trade-off between computational speed and achieved error. Indeed in Fig. 5b, we see that a user who is willing to sacrifice one order of simulation accuracy will be rewarded by a simulation that executes roughly ten times faster when using LIQSS2.

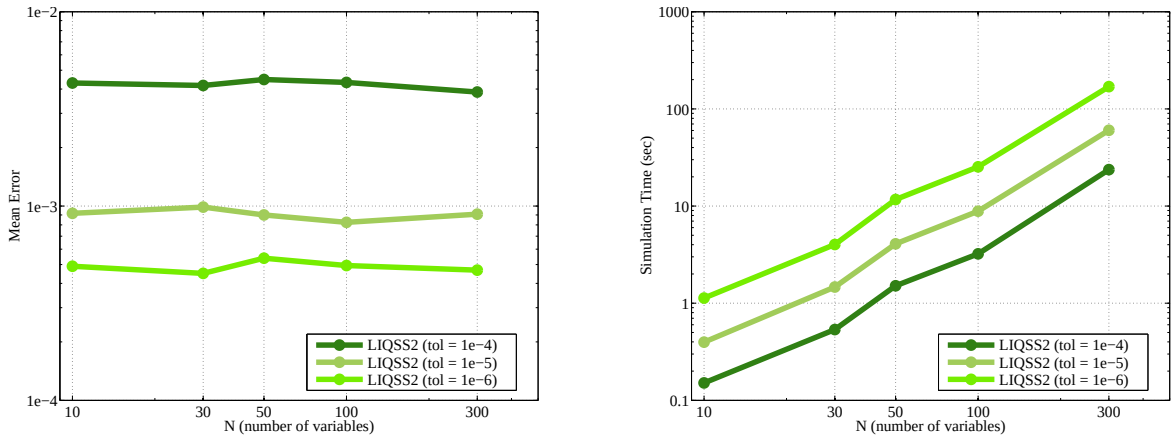
4.2 Case Study II

As the Energy Market model is not stiff we simulated it in OpenModelica using both DASSL and the fourth-order explicit Runge-Kutta algorithm. The measured simulation timings are shown in Fig. 6 where all methods achieve a mean error of order 10^{-5}. The scaling of the algorithms, as well as their relative performance, agrees with the one obtained for the District Cooling System model. However, all methods perform better on this benchmark, because it is a simpler model in general.

More precisely, DASSL scales quadratically with the number N of variables ($\sim 2N^2$) while Runge-Kutta scales linearly ($\sim 5N$) since it is an explicit algorithm and does not have to perform any matrix inversion calculations. All three QSS methods exhibit a linear scaling ($\sim N$) with the explicit QSS3 being marginally faster than the LIQSS3 algorithm (QSS3 needed 250 sec for N=10000, while LIQSS3 270 sec). For N=300, the LIQSS methods are over three orders more efficient than DASSL and over two orders more efficient than Runge-Kutta (10 sec compared to 5000×10 sec and 180×10 sec respectively).

Besides the linear scaling of the QSS methods becomes prominent their advantage over classical methods when simulating large sparse hybrid models with discontinuities. Each variable is being updated locally at its own speed, with no need of making global computations on the whole system matrix. Further-

DOI 10.3384/ECP14096787



(a) (b)

Figure 5: District Cooling System model - Mean Simulation Error (a) and Simulation Efficiency (b) for LIQSS2 and varying requested tolerances.

more, discontinuities are being handled as native simulation steps without the need of backtracking to detect the zero–crossings. Therefore, we observe that even though Runge-Kutta methods scale linearly with the system order just like the QSS methods, the latter are much more efficient than the former. QSS methods can simulate a system with N=10000 states within the same execution time as a Runge-Kutta algorithm needs to simulate a much smaller system with N=300 states.

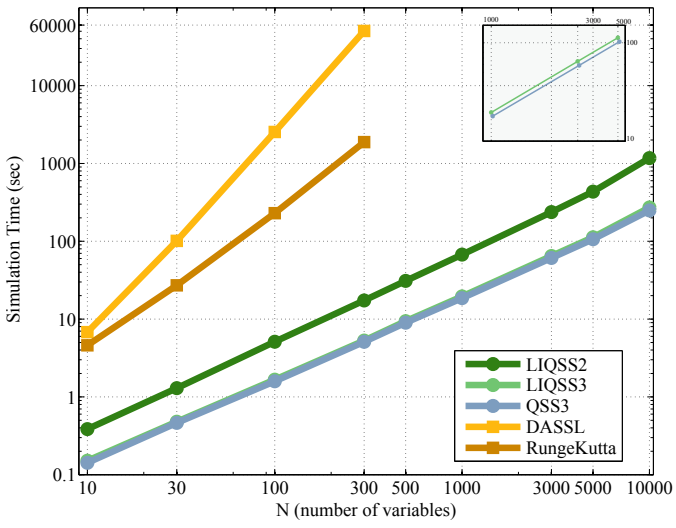


Figure 6: Simulation time for varying size of the Energy Market model. All algorithms achieve a mean error on the order of 10^{-5}.

5 Conclusion and Future Work

In this article we study the use of Quantized State System (QSS) integration methods for Smart–Grid (SG) simulation problems. The QSS methods have certain features (intrinsic sparsity exploitation, semi-implicit stiffness treatment and efficient discontinuity handling) that make them suitable for simulating SG models.

After analyzing two large hybrid SG models and comparing the efficiency and the quality of the solution obtained by the QSS methods against standard numerical integration methods we can conclude that:

- In both cases QSS methods outperform DASSL (and Runge-Kutta) by more than two orders of magnitude in terms of simulation speed, while at the same time, achieving a comparable simulation error.
- The QSS methods scale linearly with system size, while DASSL scales quadratically. The Runge-Kutta solvers also scale linearly, but they are far less efficient than their QSS competitors nevertheless.
- In both examples the QSS stand–alone simulator is able to handle larger model without running out of memory.
- We were able to simulate with the QSS methods up to 1000 times larger models than with DASSL, while still needing much less time to perform the simulations.

However, there still remain open problems to be addressed in the future. First of all, we need to perform experiments with a larger set of models typically used in the SG community, while at the same time testing more numerical integration methods against the QSS family. We note here that other implicit methods included in the OpenModelica environment, such as Radau and Lobatto, have been tested but failed to simulate the models. A necessary step that has to be performed in the future is including the family of QSS methods as integration methods in OpenModelica.

Finally, an interesting line of research could be the utilization of QSS methods in energy optimization algorithms, such as the one proposed in [5].

6 Acknowledgments

This work was in part funded by CTI grant Nr.12101.1;3 PFES-ES and supported by the OPENPROD-ITEA2 project. The authors would like to thank the developer of the stand–alone QSS solver, Joaquín Fernández, for fixing several bugs that enabled us to correctly and efficiently simulate the analyzed models.

References

[1] http://people.inf.ethz.ch/florosx/modelica2014/.

[2] *CIBSE Guide A: Environmental Design*. CIBSE Publications, Norwich, UK, 2006.

[3] F. Bergero, X. Floros, J. Fernández, E. Kofman, and F. E. Cellier. Simulating Modelica models with a Stand–Alone Quantized State Systems Solver. In *9th International Modelica Conference*, 2012.

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

[5] N. M. Ceriani, R. Vignali, L. Piroddi, and M. Prandini. An approximate dynamic programming approach to the energy management of a building cooling system. In *European Control Conference, Zurich (Switzerland), July 17-19*, pages 2026–2031, 2013.

[6] A. Elsheikh, E. Widl, and P. Palensky. Simulating complex energy systems with modelica: A primary evaluation. In *Digital Ecosystems Technologies (DEST), 2012 6th IEEE International Conference on*, pages 1–6, 2012.

[7] F. Felgner, S. Agustina, R. C. Bohigas, R. Merz, and L. Litz. Simulation of thermal building behaviour in modelica. In *2nd International Modelica Conference*, March 2002.

[8] F. Felgner, R. Merz, and L. Litz. Modular modelling of thermal building behaviour using modelica. *Mathematical and Computer Modelling of Dynamical Systems*, 12(1):35–49, 2006.

[9] J. Fernández and E. Kofman. Implementación autónoma de metodos de integración numérica QSS. Technical report, FCEIA - UNR, Rosario, Argentina, 2012.

[10] X. Floros, F. Bergero, F. E. Cellier, and E. Kofman. Automated simulation of modelica models with qss methods - the discontinuous case -. In *8th International Modelica Conference*, March 2011.

[11] X. Floros, F. Cellier, and E. Kofman. Discretizing time or states? a comparative study between dassl and qss. In *3rd International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools, EOOLT, Oslo, Norway, October 3, 2010*, pages 107–115, 2010.

[12] P. Fritzson, P. Aronsson, H. Lundvall, K. Nystrom, A. Pop, L. Saldamli, and D. Broman. The OpenModelica Modeling,Simulation, and Development Environment. *Proceedings of the 46th Conference on Simulation and Modeling (SIMS'05)*, pages 83–90, 2005.

[13] P. Fritzson and V. Engelson. Modelica - A Unified Object-Oriented Language for System Modelling and Simulation. In *ECOOP*, pages 67–90, 1998.

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

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

[16] E. Kofman and S. Junco. Quantized State Systems. A DEVS Approach for Continuous

System Simulation. *Transactions of SCS*, 18(3):123–132, 2001.

[17] G. Migoni, M. Bortolotto, E. Kofman, and F. E. Cellier. Linearly implicit quantization-based integration methods for stiff ordinary differential equations . *Simulation Modelling Practice and Theory*, 35:118 – 136, 2013.

[18] C. Perfumo, E. Kofman, J. Braslavsky, and J. Ward. Load Management: Model-Based Control of Aggregate Power for Populations of Thermostatically Controlled Loads. *Energy Conversion and Management*, 55:36–48, 2012.

[19] A. Sodja and B. Zupancic. Modelling thermal processes in buildings using an object-oriented approach and modelica. *Simulation Modelling Practice and Theory*, 17(6):1143 – 1159, 2009.

[20] M. Wetter. Modelica-based modelling and simulation to support research and development in building energy and control systems. *Journal of Building Performance Simulation*, 2(2):143–161, 2009.

[21] M. Wetter. Co-simulation of building energy and control systems with the building controls virtual test bed. *Journal of Building Performance Simulation*, 4(3):185–203, 2011.

[22] B. Zeigler, H. Praehofer, and T. G. Kim. *Theory of Modeling and Simulation - Second Edition*. Academic Press, 2000.