Build Simul (2018) 11: 405–418
https://doi.org/10.1007/s12273-017-0400-1

On the efficiency of quantization-based integration methods for building simulation

Federico Martín Bergero[1] (✉), Francesco Casella[2], Ernesto Kofman[1,3], Joaquín Fernández[1]

1. Laboratorio de Sistemas Dinámicos, CIFASIS-CONICET, Rosario, Argentina
2. Politecnico di Milano, Italy
3. FCEIA-UNR, Rosario, Argentina

Abstract

Models describing energy consumption, heating, and cooling of buildings usually impose difficulties to the numerical integration algorithms used to simulate them. Stiffness and the presence of frequent discontinuities are among the main causes of those difficulties, that become critical when the models grow in size. Quantized State Systems (QSS) methods are a family of numerical integration algorithms that can efficiently handle discontinuities and stiffness in large models. For this reason, they are promising candidates for overcoming the mentioned problems. Based on this observation, this article studies the performance of QSS methods in some systems that are relevant to the field of building simulation. The study includes a performance comparison of different QSS algorithms against state-of-the-art classic numerical solvers, showing that the former can be more than one order of magnitude faster.

Keywords

quantized state systems,
building simulation,
HVAC,
large scale system,
hybrid models

Article History

Received: 24 January 2017
Revised: 3 July 2017
Accepted: 10 July 2017

1 Introduction

Models in building simulation usually combine phenomena from different domains evolving on very different time scales, leading to stiff systems. Additionally, the continuous time submodel representing the physical phenomena often interacts with digital controllers, opening valves, and other discrete dynamics leading to hybrid systems. This interaction implies the presence of discontinuities in the differential equation systems. Also, the models sometimes contain several instances of similar components (air conditioning units, for instance), resulting in large scale systems.

Classic numerical integration methods have problems in all these cases. First, stiffness enforces the usage of implicit algorithms since explicit methods must significantly reduce the integration step in order to satisfy stability requirements (Hairer and Wanner 1996; Cellier and Kofman 2006). This is the reason why most simulation tools use DASSL (Petzold 1983) or one of its variants as the default ODE or DAE solver. Second, the presence of discontinuities requires that the algorithms detect them and restart the simulation at the point of their occurrence in order to avoid unacceptable errors caused by the integration across discontinuous functions (Cellier and Kofman 2006). Implicit methods and discontinuity detection algorithms make use of iterative routines, whose computational cost significantly grows with the model size. In consequence, the simulation of large stiff and discontinuous systems is computationally expensive.

A way to reduce these computational costs is given by the Quantized State Systems (QSS) methods (Kofman and Junco 2001; Cellier and Kofman 2006), that replace the time discretization of classic numerical algorithms by the quantization of the state variables. State quantization implies that QSS methods work in an asynchronous way, and they only perform computations when and where changes occur. In large sparse models with localized changes, this fact provides significant advantages.

QSS algorithms are also characterized by a very efficient

E-mail: bergero@cifasis-conicet.gov.ar

handling of discontinuities (Kofman 2004), as they are detected without performing iterations and their treatment does not require to re-initialize the simulation. There is also a family of QSS methods called Linearly Implicit QSS (LIQSS) (Migoni et al. 2013), that can efficiently simulate certain stiff systems without performing iterations at all.

Taking into account their advantages in the simulation of stiff, discontinuous and large scale models, QSS algorithms appear as promising candidates to integrate large scale sparse hybrid systems as those appearing in many applications related to building simulation. In fact, preliminary work has shown that QSS methods offer enormous benefits in the simulation of systems with similar features. In particular, in the simulation of a district cooling system the results showed that QSS algorithms were more than two orders of magnitude faster than classic methods (Floros et al. 2014).

Based on the previous remarks, the goal of this paper is to demonstrate the appropriateness and effectiveness of QSS integration algorithms for building simulation in presence of frequent discontinuities, stiffness, and large scale models. Towards this goal, the use of QSS integration methods in four case studies and their computational performance are discussed and compared against classic ODE solvers like DASSL, CVODE, and IDA, the last two belonging to the SUNIDALS library (Hindmarsh et al. 2005). The first case study corresponds to an air conditioning systems composed of several AC units together with their control, as described in Perfumo et al. (2012). This model is analyzed and then extended in order to include the heat capacitance in the room walls and some fast dynamics in the actuators. Then, two models of a district cooling systems composed of a chiller plant operating over a large number of cooling zones, based on a model taken from Ceriani et al. (2013), are considered. All these cases exhibit the different simulation challenges before mentioned (large scale, frequent discontinuities, stiffness).

The article is organized as follows: Section 2 introduces the main concepts used along the article. Then, Section 3 describes the case studies in detail outlining their features and the challenges they impose to the numerical integration methods. After that, Section 4 presents the simulation results and performance comparisons. Finally, Section 5 concludes the article and proposes different lines for future work.

2 Background

In this section we present the main concepts used along the article. We first discuss about the use of classic numerical integration algorithms in building simulation and introduce the family of quantized state systems methods. Then, we briefly describe the Modelica language and analyze some previous and related work.

2.1 Classic numerical integration in building simulation

Building simulation involves dynamical systems usually described by Ordinary Differential Equations (ODEs) of the form

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

where $\mathbf{x}(t)$ is the vector of state variables and $\dot{\mathbf{x}}(t)$ are their time derivatives.

Classical numerical integration methods (Hairer et al. 1993; Hairer and Wanner 1996; Cellier and Kofman 2006) discretize the time variable computing the whole state vector for certain time points t_k, using explicit formulas like

$$\mathbf{x}(t_{k+1}) = \mathbf{F}(\mathbf{x}(t_k),t_k) \tag{2}$$

or implicit equations like

$$\mathbf{F}(\mathbf{x}(t_{k+1}),\mathbf{x}(t_k),t_k) = 0 \tag{3}$$

that require iterations at each step. The time advance is ruled by a parameter called step size $h = t_{k+1} - t_k$, that is usually adapted during the simulation in order to fulfill certain accuracy settings.

Explicit algorithm steps are usually cheaper than those of implicit algorithms as the later require performing iterations. However, when the step size h becomes large with respect to the fastest dynamics of the system, explicit methods produce unstable numerical results. For that reason, in stiff systems (i.e., in presence of simultaneous fast and slow dynamics) implicit methods must be used in order to be able to increase the step size.

Besides their implicit or explicit nature, classic numerical algorithms can be classified as *one-step methods*, where the value of $\mathbf{x}(t_k)$ is computed using only the information in t_{k-1}, and *multi-step methods*, where the computations use values from previous steps. One-step methods are usually referred to as Runge–Kutta (RK) algorithms. The most used and well known multi-step methods, in turn, are those of Adams-Bashforth, Adams-Moulton, and Backward Difference Formulae (BDF).

There are several ODE solvers implementing different algorithms. Among the most popular and efficient we can mention the implementation of Dormand-Prince (DOPRI) (Dormand and Prince 1980), an explicit fifth-order variable step RK algorithm, and DASSL (Petzold 1982), an implicit variable step and variable order BDF method. Due to its robustness and its capability to integrate stiff systems, DASSL is the default solver in most modeling and simulation tools.

As we mentioned earlier, building simulation introduces some challenges to classic numerical integration solvers. First, the models usually contain discontinuities (an opening window, a person entering a room, an air conditioner turning on and off, etc.). Since the numerical algorithms

cannot integrate across discontinuities without introducing unacceptable errors, they must detect the occurrence of those events, go back to the exact time point at which the event took place, process the discrete changes and restart the simulation from that point (Cellier and Kofman 2006). These processes of event detection—also called zero crossing detection—and simulation restart slow down the simulations, especially in presence of frequent discontinuities, as the time elapsed between consecutive events imposes an upper limit to the step size h. In addition, building models usually involve components with noticeable time scale separation (thermal sub-models, for instance, evolve very slowly in comparison with electrical components). As it was mentioned above, the simultaneous presence of slow and fast dynamics is called *stiffness* and it normally enforces the usage of implicit integration algorithms, with their additional computational costs related to the iterations performed to solve the implicit equations involved.

These issues become more problematic when the models are large. Regarding discontinuities, the density of events over time usually grows with the model size, enforcing the numerical integration methods to take tiny global steps. For this reason, even in a non-stiff case using explicit solvers, the computational cost grows at least quadratically with the model size (Cellier et al. 2013). When it comes to stiffness, implicit algorithms require to perform iterations at each step, where each iteration must solve a linear system of equations of the size of the entire model. Even using efficient sparse techniques for those operations, the computational costs grow at least quadratically with the model size, leading to very slow simulations.

While in most applications it is preferable to use higher order solvers with step size control, recent results showed that some performance improvements can be achieved for the simulation of some building simulation models through the use of fixed-step low-order explicit numerical integration methods (Jorissen et al. 2015). Anyway, the usage of these algorithms requires knowing a correct value for the step size, that in many situations may be very difficult to estimate. Additionally, they can introduce unacceptable errors in presence of discontinuities.

2.2 Quantized state system 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) (Kofman and Junco 2001) approximates it by

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{q}(t),t) \tag{4}$$

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 *hysteretic quantization function*:

$$q_j(t) = \begin{cases} x_j(t) & \text{if } |\, x_j(t) - q_j(t^-)\,| \geq \Delta Q_j \\ q_j(t^-) & \text{otherwise} \end{cases} \tag{5}$$

where ΔQ_j is called *quantum* and $q_j(t^-)$ denotes the left hand limit of q_j at time t.

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 $q_j(t) = x_j(t)$. Due to the particular form of the trajectories, the solution of Eq. (4) is straightforward and can be easily translated into a simple simulation algorithm.

For $j = 1,..,n$, let t_j denote the next time at which $\left|q_j(t) - x_j(t)\right| = \Delta Q_j$. Then, the QSS1 simulation algorithm works as follows:

Algorithm 1: QSS1

```
while(t < t_f) // simulate until final time tf
  t = min(t_j), j ∈ [1,n] // adavance simulation time
  i = argmin(t_j), j ∈ [1,n] // the i-th quantized state changes first
  e_xi = t − tx_i // elapsed time since last xi update
  x_i = x_i + ẋ_i · e_xi // update i-th state value
  q_i = x_i // update i-th quantized state
  t_i = min(τ > t) subject to |q_i − x_i(τ)| = ΔQ_i // compute next i-th
        quantized state change. Here x_i(τ) = x_i + ẋ_i · (τ − t)
  for each j ∈ [1,n] such that ẋ_j depends on q_i
    e_xj = t − tx_j // elapsed time since last xj update
    x_j = x_j + ẋ_j · e_xj // update j-th state value
    tx_j = t // last xj update
    ẋ_j = f_j(q,t) // recompute j-th state derivative
    t_j = min(τ > t) subject to |q_j − x_j(τ)| = ΔQ_j // recompute j-th
          quantized state changing time. Here x_j(τ) = x_j + ẋ_j · (τ − t)
  end for
  tx_i = t // last xi update
end while
```

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 more than the quantum ΔQ_j. This fact ensures stability and global error bound properties (Kofman and Junco 2001; Cellier and Kofman 2006).
- Each step is local to a state variable x_j (the one that reaches the quantum change), and it only involves evaluations of the state derivatives that explicitly depend on that state. This fact implies that QSS1 performs intrinsic sparsity exploitation in large systems.
- If some state variables do not change significantly, they will not trigger any simulation step or function evaluation. This feature reinforces the efficient sparsity exploitation.
- The fact that the state variables follow piecewise linear trajectories simplifies the detection of discontinuities. Moreover, after a discontinuity is detected, its effects are not different from those of a normal step (because changes in q_j are discontinuous). Thus, QSS1 is very efficient in simulating discontinuous systems (Kofman 2004).

In spite of these advantages, QSS1 only performs a first order approximation and good accuracy cannot be obtained without a significant increment in the number of steps. This limitation was solved with the introduction of higher order QSS methods like QSS2 (Kofman 2002) and QSS3 (Kofman 2006).

Another problem is that QSS algorithms are not suitable to simulate stiff systems, as they introduce spurious oscillations in the numerical solution that result in additional simulation steps (Cellier and Kofman 2006). For this reason, a family of linearly implicit QSS methods (LIQSS) of order 1 to 3 was also developed (Migoni et al. 2013). Although the formulation of LIQSS methods is implicit, their implementations are explicit and do not require performing iterations. LIQSS methods share the advantages of QSS methods and, additionally, they are able to efficiently handle stiff systems, provided that the stiffness is due to the presence of large entries in the main diagonal of the system Jacobian matrix[1].

Consequently, in the simulation of systems that are large, sparse, discontinuous or exhibit the type of stiffness that is properly handled by these algorithms, the usage of Quantized State solvers can offer a better performance than that of classic discrete time methods.

2.3 Stand-alone QSS solver

The implementation in software of QSS algorithms is more involved than that of classic numerical integration methods. The reason is that each QSS step involves a change in a single variable and only some components of the right hand side of the ODE must be computed. The first implementations of these algorithms were based on the fact that the behavior of the QSS approximation given by Eq. (4) can be easily described by a discrete event system using the DEVS formalism (Zeigler et al. 2000). Thus, QSS algorithms were originally implemented inside DEVS simulation engines. Unfortunately, DEVS-based implementations of QSS methods are inefficient as DEVS simulation engines waste a large amount of the computational load attending the DEVS simulation mechanism.

Recently, the complete family of QSS methods was implemented in a *stand-alone QSS Solver* coded in plain C language (Fernández and Kofman 2014). This tool simulates models that can contain discontinuities represented as:

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x},\mathbf{d},t) \tag{6}$$

where $\mathbf{d}$ is a vector of discrete variables that can only change when a condition

$$ZC_i(\mathbf{x},\mathbf{d},t) = 0 \tag{7}$$

for some $i \in \{1,\cdots,z\}$ is met. The components ZC_i form a vector of zero-crossing functions $\mathbf{ZC}(\mathbf{x},\mathbf{d},t)$. When a zero-crossing condition of Eq. (7) is verified, the state and discrete variables can change according to the corresponding event handler:

$$(\mathbf{x}(t),\mathbf{d}(t)) = H_i(\mathbf{x}(t^-),\mathbf{d}(t^-),t) \tag{8}$$

These models are simulated using QSS methods that approximate Eq. (6) by

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{q},\mathbf{d},t) \tag{9}$$

where each component $q_i(t)$ is a piecewise polynomial approximation of the corresponding component of the state $x_i(t)$.

The simulation is performed by three modules interacting at runtime:

(1) The **Integrator**, that integrates Eq. (9) assuming that the piecewise polynomial quantized state trajectory $\mathbf{q}$(t) is known.

(2) The Quantizer, that computes $\mathbf{q}$(t) from $\mathbf{x}$(t) according to the QSS method in use and their tolerance settings (there is a different **Quantizer** for each QSS method). That way, it provides the polynomial coefficients of each quantized state $q_i(t)$ and computes the next time at which a new polynomial section starts (i.e., when the condition $|\, q_i(t) - x_i(t)\,| = \Delta Q_i$ is met).

(3) The **Model**, that computes the scalar state derivatives $\dot{x}_i = f_i(\mathbf{q},\mathbf{d},t)$, the zero-crossing functions $ZC_i(\mathbf{x},\mathbf{d},t)$, and the corresponding event handlers $H_i(\mathbf{q},\mathbf{d},t)$. Besides, it provides the structural information required by the algorithms.

The structure information of the **Model** is automatically extracted at compile time by a **Model** Generator module. This module takes a standard model described in a subset of the Modelica language called μ-Modelica (Bergero et al. 2012) and produces an instance of the **Model** module as required by the QSS solver.

The Stand Alone QSS Solver also offers a front-end to classic numerical solvers like DASSL, DOPRI, and CVODE. Taking into account that the current implementation of QSS methods requires that the models are described in Modelica, we provide below a brief introduction to this modeling language.

2.4 Modelica and μ-Modelica

Modelica (Mattsson et al. 1998; Fritzson 2015) is an object-oriented declarative modeling language that allows the combination of models belonging to different physical and

[1] When the stiffness obeys to other reasons, LIQSS methods may also introduce spurious oscillations in the numerical solution.

technical domains in a unified way. Elementary Modelica components are usually described by means of differential-algebraic equations, with the eventual presence of discontinuities and discrete evolutions. These elementary components can be connected to compose more complex models.

Making use of the object-oriented features of the language, a repository of models from different domains (thermal, mechanical, electrical, etc.) called *Modelica Standard Library* (MSL) was developed. Derived from the MSL, a specific library for building simulation called *Modelica Buildings Library* was also developed and applied to several problems in the discipline (Wetter and Haugstetter 2006; Wetter et al. 2014, 2016; Fuchs et al. 2015; Nytsch-Geusen et al. 2013; Baetens et al. 2015).

From a mathematical point of view, Modelica models are collections of differential algebraic equations (DAEs). Modelica compilers translate these descriptions into a programming language[2] piece of code that allows to evaluate the right hand side of an equivalent ODE. The conversion from DAE to ODE requires reducing the DAE index (in presence of structural singularities), solving the algebraic loops (or producing the iterative code that solves them), and sorting the system of equations.

Several Modelica software tools are available, both commercial (like Dymola (Brück et al. 2002) and Wolfram SystemModeler) and open source (like OpenModelica (Fritzson et al. 2005) and JModelica (Åkesson et al. 2009)). These tools combine user-friendly modeling environments with Modelica compilers and different ODE solvers.

There exists also a reduced subset of Modelica language called μ-Modelica (Bergero et al. 2012), that contains only the minimal statements and functions that are necessary to describe plain systems of ODEs with discontinuities. As it was mentioned earlier, this language is used by the stand-alone QSS Solver to describe the models. Anyway, models described in general Modelica language can be also simulated by this tool making use of automatic translators from Modelica to μ-Modelica like that of OpenModelica (Bergero et al. 2012) and ModelicaCC, a new Modelica compiler optimized for large scale models (Bergero et al. 2015).

The mentioned Modelica to μ-Modelica translators are in fact Modelica compilers like those described above that convert the DAEs to ODEs reducing the DAE index, solving the algebraic loops (or producing the iterative code that solves them), and sorting the system of equations. Unlike the regular Modelica compilers, these translators write μ-Modelica code instead of the C language code. Thus, complex Modelica models containing algebraic loop or structural singularities can be simulated using the QSS methods through these translations.

[2] C language is used in most Modelica tools.

2.5 Previous and related work

Some previous work has been done regarding the usage of QSS methods in the field of building simulation. Preliminary results reported in Floros et al. (2014) showed the advantages of using QSS in some problems related to building performance simulation. Anyway, the examples analyzed in that work used unrealistic parameters for the context of buildings. Also, the experiments were performed using different simulation tools for QSS and DASSL, leading to unfair comparisons (DASSL simulations were run with OpenModelica while QSS simulations were run on the stand alone QSS solver that produces more efficient simulation code).

QSS simulations of heat transfer in multi-layered walls were studied by Frances et al. (2014, 2015). The results were compared against those of EnergyPlus (Crawley et al. 2001) and found to be in concordance. As these works were limited to performing an early feasibility study, QSS simulations were carried on using PowerDEVS (Bergero and Kofman 2011), a DEVS-based implementation of QSS that is not optimal in terms of efficiency. Anyway, the discontinuity handling features of QSS allowed the simulation of hybrid systems including some phenomena that could not be simulated in EnergyPlus. Motivated by these results, some work is currently being done in order to include the QSS methods into EnergyPlus (Wetter et al. 2015).

3 Case studies

In this section, we present four case studies that are then simulated in the next section. The models exhibit one or more features that impose difficulties to classic numerical integration algorithms: stiffness, frequent discontinuities, and all of them have a parameter that defines the model size, so they can become of large scale type.

The first two cases correspond to a centralized control of the total power consumed by a population of air conditioner (AC) units. Both models contain frequent discontinuities caused by the AC units turning on and off. In addition, the second case has more realism including the modeling of a wall that increases the system size, and an actuator that introduces stiffness.

The remaining two cases represent a District Cooling System. Here, the models have a more complex structure than that of the first cases. In addition, the models are stiff due to the local cooling controllers and the heat exchangers. Moreover, one of these models involves frequent discontinuities.

3.1 Case study I. Air conditioner population

This model, taken from Perfumo et al. (2012), allows the

study of a centralized system that controls the power consumed by the AC population of a building.

The model considers that each AC unit refrigerates one room, so that the temperature of the i-th room, $\theta_i(t)$, follows the equation

$$\dot{\theta}_i(t) = -\frac{1}{C_i \cdot R_i}[\theta_i(t) - \theta_a + R_i \cdot P_i \cdot m_i(t)] \quad (10)$$

Here, R_i and C_i are parameters representing the thermal resistance and capacity of the i-th room, respectively. P_i is the power of the i-th air conditioner in *on* state and θ_a is the ambient temperature.

The variable $m_i(t)$ represents the state of the i-th air conditioner, where $m_i(t)$=1 is the *on* state, and $m_i(t)$=0 is the *off* state. This variable evolves according to the hysteretic on–off control law:

$$m_i(t^+) = \begin{cases} 0 & \text{if } \theta_i(t) \le \theta_r^k - 0.5 \text{ and } m_i(t) = 1 \\ 1 & \text{if } \theta_i(t) \ge \theta_r^k + 0.5 \text{ and } m_i(t) = 0 \\ m_i(t) & \text{otherwise} \end{cases} \quad (11)$$

where θ_r^k is the global reference temperature calculated by the centralized control system.

The power consumption of the entire AC population is computed as:

$$P(t) = \sum_{i=1}^{N} m_i(t) \cdot P_i$$

and a centralized digital control system regulates it, so that it follows a desired power profile $P_r(t)$. This centralized control system uses a discrete time Proportional Integral (PI) law to compute the reference temperature θ_r^k as:

$$\theta_r^k = \theta_0 + K_P \cdot [P_r(t) - P(t)] + K_I \sum_{j=1}^{k-1} [P_r^j(t) - P^j(t)]$$

where K_P and K_I are the parameters of the PI controller.

In the experiments, the power profile $P_r(t)$ follows a pulse trajectory that starts at the 50% of the total power, then it falls to the 40% from t=1000 until t=2000, when it comes back to its original value. The complete model contains N AC units with the set of parameters listed in Table 1. The parameters corresponding to the room dimensions and AC power have a 25% variation around the reported mean value.

Simulation issues

This model contains N rooms with one AC unit each having different parameters (heat capacitance, power, etc), so the temperature in the different rooms evolves at different pace and the on/off events occur at different time points. That way, as N increases, the number of on/off events per time unit also grows. Since the time between consecutive events impose an upper limit to the step size of classic numerical algorithms, the number of simulation steps grows with N while the complexity of evaluating the right hand side of Eq. (1) also grows. In consequence, we expect that the computational cost grows quadratically with the model size.

Table 1 Model parameters—AC population example

Parameter	Value
Room interior	
Room dimension	3 m × 3 m × 3 m
Heat capacitance of air	1.012 kJ/(kg·K)
Mass of air	35 kg
Room heat capacitance	35.4 kJ/K
Initial temperature	22°C
Room walls	
Brick conductivity	0.89 W/(m·K)
Brick capacitance	0.840 kJ/(kg·K)
Wall thickness	0.3m
Wall resistance	9.363×10^{-3} m^2·K/W
Wall capacitance	1.52×10^6 kJ/K
Initial temperature	26°C
Control	
K_I	1
K_P	1
θ_0	20 °C
AC unit	
AC power	1 kW
Environment	
θ_a	32 °C

This model is not stiff. Taking N=50, for instance, the Jacobian matrix eigenvalues are located in the interval [−0.00275, −0.00198]. Thus, the usage of implicit algorithms will only increase the computational costs due to the computation of the Jacobian matrix and the iterations on a system of N equations.

3.2 Case study II: A more realistic room

The second case study adds more realism to the previous model, including the heat capacitance of the brick walls and the dynamics of the AC actuator. That way, the i-th room temperature $\theta_i(t)$ evolves as follows:

$$\begin{aligned} \dot{\theta}_i(t) &= \frac{1}{C_i}\left[\frac{\theta_i^w(t) - \theta_i(t)}{R_i} - p_i(t)\right] \\ \dot{\theta}_i^w(t) &= \frac{1}{C_i^w}\left[\frac{\theta_a - \theta_i^w(t)}{R_i^w} - \frac{\theta_i^w(t) - \theta_i(t)}{R_i}\right] \\ \dot{p}_i(t) &= \frac{1}{\tau}[m_i(t) \cdot P_i - p_i(t)] \end{aligned} \quad (12)$$

Here, $\theta_i^w(t)$ represents the temperature of the i-th room

walls and $p_i(t)$ is the cooling power of i-th AC unit. The parameters C_i^{w} and R_i^{w} are the thermal capacitance and resistance of the wall, and τ is the time constant of the AC actuator. The remaining parameters and variables coincide with those of the previous case and their values are listed in Table 1.

Simulation issues

This modified model has three times as many state variables as the previous one, so the evaluation of the right hand side of the ODE is more expensive. Additionally, the actuator dynamics imposes a fast dynamics, so the model becomes stiff. In fact, the Jacobian eigenvalues are located in the interval [−1000, −0.000047]. On the other hand, the presence of heat storage at the brick walls attenuates the temperature changes in the room slowing down the frequency of on–off events in the AC units. In conclusion, we expect that the simulation of this model has more computational costs associated to the size and stiffness, but less discontinuities than the previous one.

3.3 Case study III: A district cooling system

The following case, adapted from Ceriani et al. (2013), represents a centralized system used to distribute cooling power to several rooms (zones) through a water circuit as shown in Fig. 1.

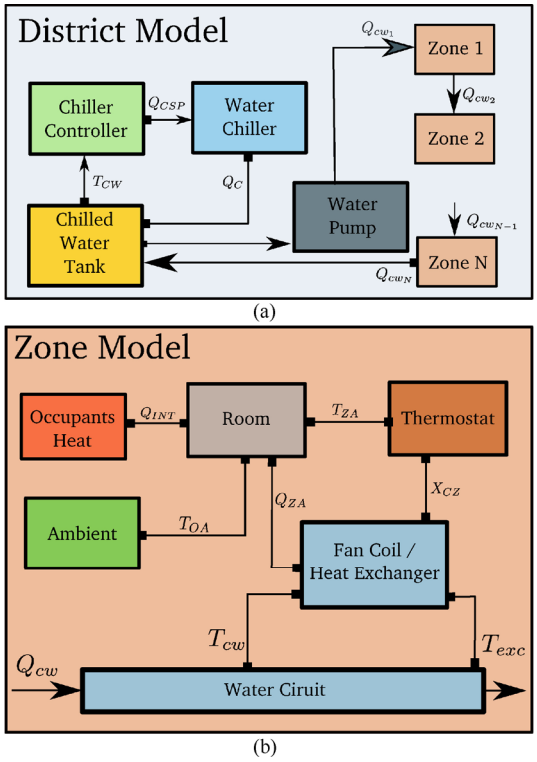

Fig. 1 (a) District cooling system, and (b) detail of a *zone* submodel

The model is formed by the following components:

- The *Water Chiller* that cools the water that is then pumped to the cooling circuit.
- The *Chilled Water Tank* consisting of a reservoir connected to the *Water Chiller*, from which the water is pumped to the water circuit.
- The *Chilled Water Circuit* that forms a circular network of incompressible fluid, where the water flows from the *Chilled Water Tank*, arriving then to the first zone, and passing through the successive zones until arriving back to the tank. The circuit is divided in sections, so that the i-th circuit section exchanges heat with the i-th zone.
- The *Zones*, that are modeled as rooms that exchange heat with the outside ambient and with the *Heat Exchangers*. They also receive heat from internal sources such as occupants and office equipment.
- The *Heat Exchangers*, that exchange heat between each section of the *Chilled Water Circuit* and the corresponding room.
- The *Zone Temperature Controllers*, that regulate the opening and closing of each *Heat Exchanger* valve in order to control the temperature of the corresponding zone.
- The *Chilled Water Temperature Controller*, that computes the *Cooling Plant* power that is necessary to regulate the temperature in the water tank at a specified set-point.

The mathematical representation of these components are described below:

The Water Chiller is simplified in this article, so it delivers exactly the power $Q_{\mathrm{CSP}}(t)$ required by the *Chilled Water Temperature Controller*.

The Chilled Water Tank contains water at the temperature $T_{\mathrm{TANK}}(t)$, that evolves according to:

$$C_{\mathrm{TANK}}\dot{T}_{\mathrm{TANK}}(t) = Q_{\mathrm{CW}}^{N}(t) - Q_{\mathrm{CSP}}(t) - Q_{\mathrm{CW}}^{0}(t) \tag{13}$$

where C_{TANK} is the tank heat capacitance, $Q_{\mathrm{CW}}^{N}(t)$ is the heat flow entering the tank from the last section of the *Chilled Water Circuit* and

$$Q_{\mathrm{CW}}^{0}(t) = T_{\mathrm{TANK}}(t)\cdot\rho\cdot C_{\mathrm{W}}\cdot q_{\mathrm{FLOW}} \tag{14}$$

is the heat flow going from the tank to the first section of the water circuit. Here, q_{FLOW} is the constant water flow impulsed by the water pump.

The Chilled Water Circuit dynamics at the i-th section obeys the equations

$$\begin{aligned} C_{\mathrm{CW}}^{i}\dot{T}_{\mathrm{CW}}^{i}(t) &= Q_{\mathrm{CW}}^{i-1}(t) - Q_{\mathrm{CW}}^{i}(t) - \rho\cdot C_{\mathrm{W}}\cdot q_{\mathrm{EXC}}^{i}(t)\cdot(T_{\mathrm{CW}}^{i}(t) \\ &\quad - T_{\mathrm{EXC}}^{i}(t)) \\ Q_{\mathrm{CW}}^{i}(t) &= T_{\mathrm{CW}}^{i}(t)\cdot\rho\cdot C_{\mathrm{W}}\cdot q_{\mathrm{FLOW}} \end{aligned} \tag{15}$$

where T_{CW}^{i} is the water temperature at i-th section, C_{CW}^{i} is

its thermal capacity, $Q^i_{CW}(t)$ is the heat flowing from the *i*-th circuit section to the next one, $T^i_{EXC}(t)$ is the temperature of the water in the *i*-th *Heat Exchangers*, and $q^i_{EXC}(t)$ is the controlled water flow entering the heat exchanger. Notice that the first circuit section receives the heat flow $Q^0_{CW}(t)$ coming from the chilled water tank.

The Zones are represented by the following equations:

$$\begin{aligned} C^i_{ZA}\dot{T}^i_{ZA}(t) &= -Q^i_{ZA}(t) + \frac{1}{R^i_{out}}\cdot(T_{OA}(t) - T^i_{ZA}(t)) \\ Q^i_{ZA}(t) &= \frac{1}{R^i_{cw}}\cdot(T^i_{ZA}(t) - T^i_{EXC}(t)) \end{aligned} \tag{16}$$

where $T^i_{ZA}(t)$ is the *i*-th room temperature, C^i_{ZA} is its thermal capacity, $T_{OA}(t)$ is the external temperature, R^i_{out} is a constant parameter representing the thermal resistance with the environment, and R^i_{cw} is the thermal resistance between the zone and the heat exchanger.

The Heat Exchangers are modeled as:

$$\begin{aligned} C^i_{EXC}\dot{T}^i_{EXC}(t) &= Q^i_{ZA}(t) + \rho\cdot C_W\cdot q^i_{EXC}(t)\cdot(T^i_{CW}(t) - T^i_{EXC}(t)) \\ q^i_{EXC}(t) &= X^i_c(t)\cdot q_{MAX} \end{aligned} \tag{17}$$

where $X^i_c(t)$ is the valve opening fraction (computed by the *Zone Temperature Controller*) and q_{MAX} is the maximum water flow through the heat exchanger.

The Chilled Water Temperature Controller is a PI controller with an anti-windup scheme that computes the cooling power set point $Q_{C\,SP}(t)$ that is necessary to regulate the chilled water tank temperature $T_{TANK}(t)$ around its set point $T_{CW\,SP}(t)$:

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

Here, $\Phi_{[a,b]}(\cdot)$ is a saturation function:

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

modeling the actuator limits. $e_{C,CW}(t)$ is the error in the regulated temperature, and $z_{C,CW}(t)$ is the integral state that, in a non saturated situation, is proportional to the integral of $e_{C,CW}(t)$. When the controller saturates, this anti-windup scheme clamps the integrator state to an equilibrium value instead of experiencing a linear growth with the error. Finally, $k_{P,CW}$ and $k_{I,CW}$ are the proportional and integral gains of the PI controller.

The Zone Temperature Controller of the *i*-th zone is also a PI controller with anti-windup, that attempts to control the zone temperature $T^i_{ZA}(t)$ around a set-point $T_{ZA\,SP}(t)$. The control variable is the heat exchanger valve opening $X^i_c(t)$, which spans the range [0,1]. The controller is modeled as:

$$\begin{aligned} e^i_{C,Z}(t) &= T^i_{ZA}(t) - T_{ZA\,SP}(t) \\ \dot{z}^i_{C,Z}(t) &= -\frac{k_{I,Z}}{k_{P,Z}}\cdot z^i_{C,Z}(t) + \frac{k_{I,Z}}{k_{P,Z}}\cdot X^i_c(t) \\ X^i_c(t) &= \Phi_{[0,1]}\left(k_{P,Z}\cdot e^i_{C,Z}(t) + z^i_{C,Z}(t)\right) \end{aligned}$$

with identical definitions to those of the *Chilled Water Temperature Controller*.

Model parameters

Table 2 shows the parameters for this case. In order to obtain a realistic behavior as the number of zones *N* changes, some parameters are proportional to the model size as it is expressed in the table.

Table 2 Model parameters—district cooling zone

Parameter	Value
Water tank and circuit	
Water temp. set point ($T_{CW\,SP}$)	10 °C
Water density (ρ)	998 kg/m^3
Water capacity (C_W)	4.18×10^3 J/(kg·°C)
Tank volume (Vol_{TANK})	$0.1 \times N$ (m^3)
Tank capacity (C_{TANK})	$C_W\,\rho Vol_{TANK}$ (J/(kg·°C))
Pipe volume (Vol_{PIPE})	$0.01 \times N$ (m^3)
Water flow (q_{FLOW})	$0.0001 \times N$ (m^3/s)
Water chiller and controller	
Chiller power ($Q_{C,max}$)	$3000 \times N$ (W)
Integral gain ($k_{I,CW}$)	$0.5 \times N$ (W/(°C·s))
Proportional gain ($k_{P,CW}$)	$6000 \times N$ (W/°C)
Heat exchanger	
Exchanger volume (Vol_{EXC})	0.005 m^3
Exchanger capacitance (C_{EXC})	1.25×10^5 J/(kg·°C)
Maximum flow (q_{MAX})	0.0001 m^3/s
Zone	
Temperature set point (T_{ZASP})	23 °C
Zone capacity (C_{ZA})	6092×10^3 J/W
Resistance with outside (R_{out})	2.16×10^{-3} °C/W
Resistance with exchanger (R_{cw})	3.03×10^{-4} °C/W
Internal gain (p_1,p_2,p_3)	0.2199J/(°C)^2, 5.0597J/°C, 84.9168J
Zone thermostat	
Integral gain ($k_{I,Z}$)	5×10^{-4} (°C·s) $^{-1}$
Proportional gain ($k_{P,Z}$)	1 (°C) $^{-1}$

Figure 2 plots the simulated trajectories for the ambient, the first zone and the last zone temperatures. Then, Fig. 3 shows the water temperature at the tank and at the first and last circuit segments. The simulations correspond to a model with 50 zones. As expected, the temperatures at first zone are cooler because it is closer to the chilled water tank.

Simulation issues

This model has a more complex structure than that of the previous cases. There are four state variables associated to each zone, and three additional states associated to the remaining components, completing a total of $4\times N+3$ states. The system has also some discontinuities corresponding to the saturation of the different actuators. However, discontinuities are not very frequent. The zone controllers also impose a relatively fast dynamics compared to that of the global system, so the model is also stiff. Moreover, the model becomes more stiff as N growths. In fact, taking N=50 zones, the real part of the Jacobian matrix eigenvalues are located in the interval $[-0.935, -5.2\times10^{-6}]$, while taking N=100 the eigenvalues are spread in the interval $[-1.927, -5.2\times10^{-6}]$.

3.4 Case study IV

This last case is an extension of the previous one, where the

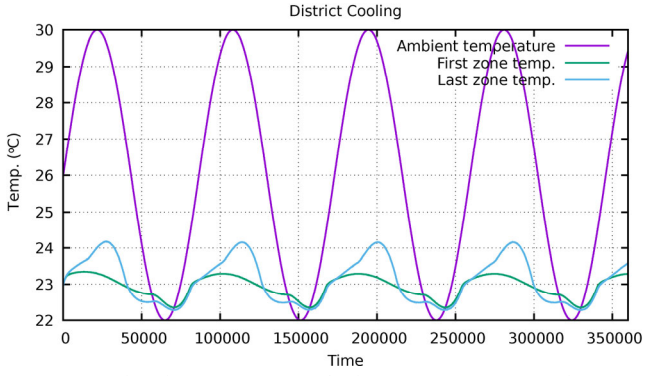

Fig. 2 Simulated trajectories of the environment and zones temperatures

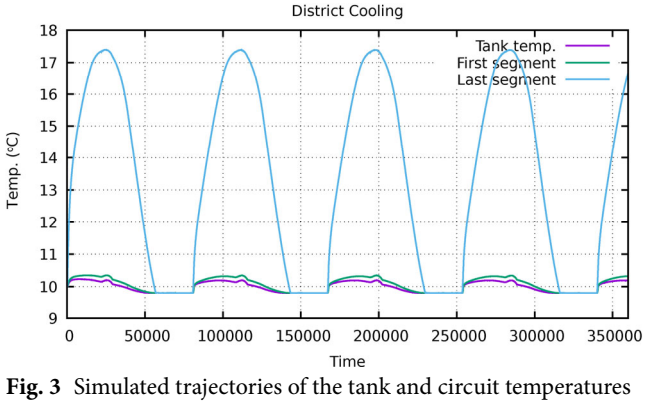

Fig. 3 Simulated trajectories of the tank and circuit temperatures

model of each zone also considers the internal heat flow produced by the occupants. The internal heat flow $Q^i_{\text{INT}}(t)$ is modeled according to the following non-linear function of the zone temperature:

$$Q^i_{\text{INT}}(t) = (p_1(T^i_{\text{ZA}})^2(t) + p_2 T^i_{\text{ZA}}(t) + p_3) n^i_{\text{people}}(t)$$

where n^i_{people} is the number occupants of the i-th zone. This model is proposed in CIBSE (2006), where suitable values for the coefficients p_1, p_2, and p_3 can be found (see Table 2).

With these considerations, Eq. (16) is modified as follows:

$$C_{\text{ZA}}\dot{T}^i_{\text{ZA}}(t) = -Q^i_{\text{ZA}}(t) + \frac{1}{R_{\text{out}}}(T_{\text{OA}}(t) - T^i_{\text{ZA}}(t)) + Q^i_{\text{INT}}(t) \tag{18}$$

The number of occupants $n^i_{\text{people}}(t)$ is generated by a simple stochastic model, where 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 enters or leaves the room. When there are less than 10 people, the probability of entering the room is 50%. Otherwise, it is 15%.

Although this model does not represent any specific and realistic pattern to model room occupancy, it serves for the purpose of generating a number of *uncorrelated* time events, whose density in time grows with the number of zones.

Simulation issues

This case is almost identical to the previous one, but it has the addition of time events caused by people entering and leaving the zones. As in the AC population example, the event density grows linearly with the number of zones which in turn limits the maximum step size. However, this time, the additional discontinuities caused by people entering and leaving the zones are *time events* instead of *state events*. Time events are easier to locate as their time of occurrence is known in advance, so there is no need of iterating on zero crossing functions depending on the state variables.

4 Simulation results

In this section we present the simulation results for the case studies introduced above using different numerical integration algorithms.

Simulation benchmark

The simulations were performed under the following settings:

- The simulation platform is a desktop computer with an Intel i7-3770 processor running at 3.40 GHz with 4 GB RAM under Ubuntu 16.04 operating system.

- All the simulations reported were performed using the QSS Stand Alone Solver (Fernández and Kofman 2014) as front-end.
- In all cases we compared QSS2, LIQSS2, DOPRI, DASSL, CVODE, and IDA results under equivalent tolerance settings: in classic solvers we used absolute and relative tolerances of *abstol*=*reltol*=10^{-3}, while in QSS we used absolute and relative quanta of $\Delta Q_{abs}=\Delta Q_{rel}=10^{-3}$.
- The implementation of DOPRI is `dopri5.c`, described in Hairer et al. (1993), with the addition of a zero crossing detection routine. DASSL results correspond to the code `ddaskr.f`, while CVODE and IDA are part of the SUNDIALS package (Hindmarsh et al. 2005). SUNDIALS solvers are always invoked making use of a sparse symbolical Jacobian evaluation.
- In all cases, we performed simulations for different values of the parameter N that represents the model size.
- We did not observe variability in the CPU time taken by repeating experiments. In consequence, we did not average multiple results.
- Simulation errors were computed by comparing the results against a reference trajectory obtained using CVODE with a tight tolerance (1×10^{-6}), using the formula:

$$\text{err} = \sum_{i=1}^{P} \sqrt{\frac{(y^{\text{sim}}[i] - y^{\text{ref}}[i])^2 \,/\, (y^{\text{ref}}[i])^2}{P^2}} \qquad (19)$$

- where P=5000 is the number of equidistant output points.
- Although some of the solvers can run on parallel exploiting the multi-core architecture, all the experiments were realized using a single core.

The reported results can be reproduced using the QSS Stand Alone Solver (https://sourceforge.net/projects/qssengine/). The models are included in a Modelica package that can be downloaded from http://www.fceia.unr.edu.ar/~fbergero/Examples.mo.

4.1 Case study I

The model was simulated varying N from 50 to 1000 AC units, using a final time t_f=3600 in all cases. The CPU time taken by each experiment is depicted in Fig. 4, while the errors are shown in Fig. 5.

Result analysis

Both QSS algorithms (LIQSS2 and QSS) outperform all classic solvers in all cases. What is more important, the difference becomes larger as the number of AC units grows. When the number of AC units is large (greater than 400) the CPU time of QSS algorithms shows a linear growth with the number of AC units, while the CPU times of DOPRI, CVODE and IDA exhibit a near quadratic growth and DASSL is almost cubic.

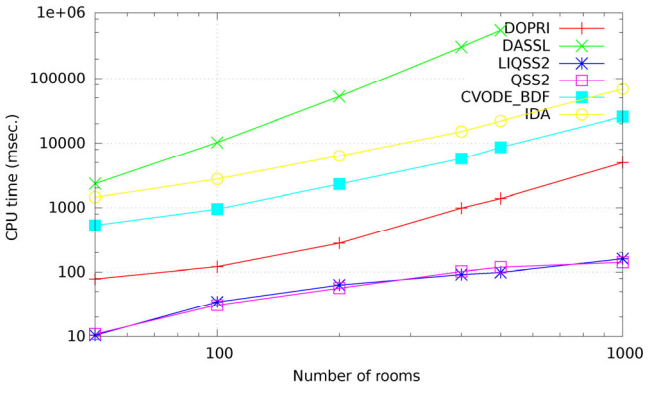

Fig. 4 CPU time vs. number of AC units for different solvers: Case study I

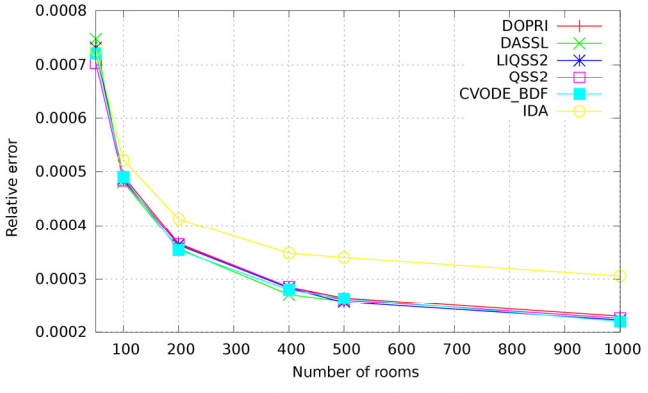

Fig. 5 Error vs. number of AC units for different solvers: Case study I

The approximately linear complexity of QSS was already known for large problems involving loosely coupled components including discontinuities (Grinblat et al. 2012). This is due to the fact that these algorithms do not need to reinitialize the whole simulation after the occurrence of a discontinuity.

Classic algorithms, in turn, perform steps at the discontinuity points: every one second (when the discrete time controller is updated) and whenever any AC unit switches on or switches off. Consequently, when the number of AC units increases, the number of switching events increases and the steps become shorter. Additionally, the evaluation of the right hand side of the ODE becomes more expensive as the model grows. Thus, when the number of AC units is large (so the number of switching events is much larger than those of the discrete time controller), the number of steps taken by the algorithm grows almost linearly with N. As the cost of evaluating the right hand side of the ODE is also linear with N, the CPU time grows quadratically with N.

Implicit algorithms have additional costs related to the evaluation of the Jacobian matrix and the Newton iterations. This is why DASSL CPU time is almost cubic. Implicit SUNDIAL solvers (CVODE and IDA) face the same issue

but they evaluate the Jacobian matrix using a sparse form and they perform the iterations exploiting that representation adding only a linear cost to the computation of each simulation step. Thus, these algorithms are more expensive than DOPRI but they scale in the same way.

The fact that the model is not stiff implies that QSS2 and LIQSS2 have almost identical CPU times. The lack of stiffness also implies that DOPRI is much faster than implicit algorithms, as the step size is limited by the occurrence of events instead of being limited by stability requirements.

Regarding the errors, all the algorithms show very similar figures. A noticeable feature is that the error becomes smaller as the number of AC units increases. The reason is that the simulation steps become smaller as N increases, reducing the simulation error.

4.2 Case study II

This model was run under the same conditions of the previous one. The CPU time taken by each experiment is depicted in Fig. 6, while the errors are shown in Fig. 7.

Result analysis

This time, LIQSS2 is significantly faster than any other solver and the CPU time grows linearly with the number of AC units. Due to the stiffness introduced by the actuator dynamics, QSS2 becomes almost 1000 times slower than LIQSS2.

Provided that N is not very large, CVODE solver shows the best performance among classic solvers. Although their simulation steps are more expensive than those of DOPRI, it does not have stability limitations and it can perform larger steps that are only limited by the occurrence of discontinuities. DASSL, in turn, also performs large steps but it is not optimized for sparse systems and the cost of each step is significantly larger.

As N increases, DOPRI becomes faster than CVODE. The reason is that DOPRI step size is limited by stability constraints (it is smaller than the time between consecutive events). Thus, the step size in DOPRI is almost constant and the CPU time only grows linearly with N. If N becomes even larger so that the time between consecutive events becomes smaller than the step size that ensures stability, then DOPRI should also exhibit quadratic growth.

Regarding errors, this time those of QSS are larger than those of classic algorithms and they do not fall as N grows. Anyway, they have the order of the prescribed tolerance.

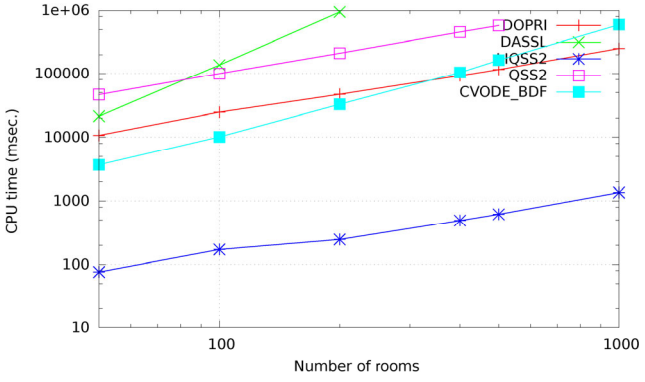

Fig. 6 CPU time vs. number of AC units for different solvers: Case study II

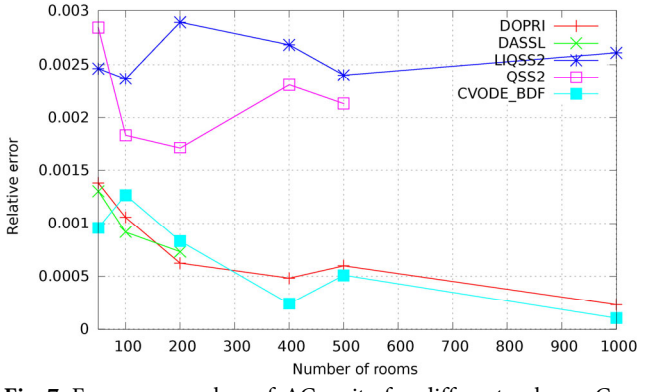

Fig. 7 Error vs. number of AC units for different solvers: Case study II

4.3 Case study III

The District Cooling model was simulated varying the number of zones N from 50 to 500, until a final time t_f=360000 (i.e., 4 days and 4 hours). In this case, in order to obtain qualitatively better results, the relative and absolute tolerance of the classic solvers was set as *abstol*=*reltol*=10^{-4}, while in QSS an absolute and relative quanta of ΔQ_{abs}= ΔQ_{rel}=10^{-4} was selected. The CPU times taken by the different experiments are depicted in Fig. 8, while the errors are shown in Fig. 9.

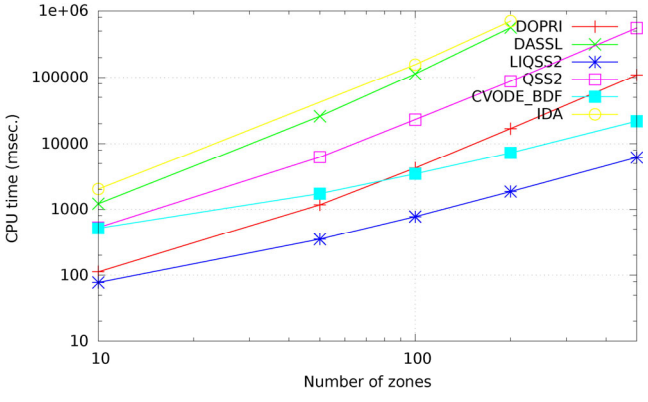

Fig. 8 CPU time vs. number of zones for different solvers: Case study III

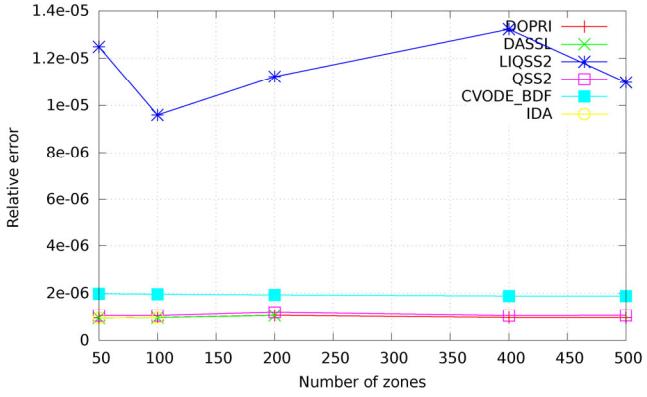

Fig. 9 Error vs. number of zones for different solvers: Case study III

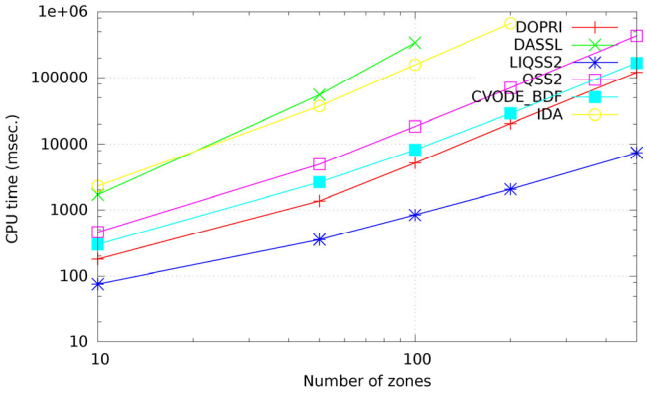

Fig. 10 CPU time vs. number of zones for different solvers: Case study IV

Result analysis

The results are similar to those of the last example. However, when N is small, DOPRI's performance is now closer to that of LIQSS2. This is mainly due to the fact that the model is not as stiff as the previous one and DOPRI can simulate it using reasonable large step sizes. However, when N grows the model becomes more stiff and CVODE results faster than DOPRI (but slower than LIQSS2).

This time, the model does not contain frequent discontinuities with an occurrence rate that grows with N. Thus, classic solvers do not exhibit the quadratic cost of the previous cases. Taking into account the absence of frequent events, the main advantage of LIQSS2 in this case is the efficient treatment of stiffness.

Regarding the errors, they are similar to those of the previous example. LIQSS2 errors are larger than those of the other models, but they are anyway bounded within the prescribed tolerance.

4.4 Case study IV

This last case was simulated under identical settings than the previous one. The CPU times taken by the different experiments are depicted in Fig. 10, while the errors are shown in Fig. 11.

In this case, the presence of frequent discontinuities (that are more frequent as the number of zones grows) reduces the performance of classic solvers, that exhibit back near quadratic growths with N. The CPU time of QSS solvers, however, is not affected by these events. That way, the advantages of LIQSS2 are more noticeable than in the previous example.

Errors are also within the prescribed tolerance, but it is now DOPRI the solver with the larger errors. These errors are mainly due to some inaccuracy related to the time event location.

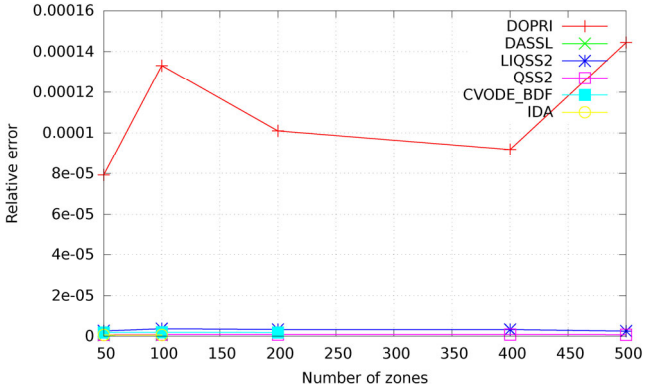

Fig. 11 Error vs. number of zones for different solvers: Case study IV

5 Conclusions and future work

This work studied the performance of QSS numerical integration methods in the field of building performance simulation, comparing it with that of classic solvers like DOPRI, DASSL and CVODE on different case studies. These case studies had some features that are common in building simulations and that impose some challenges to classic solvers: stiffness, frequent discontinuities and a large number of weakly connected components. In these cases, the results obtained suggest that QSS methods can be a good alternative to classic solvers. Particularly, the second order accurate LIQSS2 algorithm outperforms all classic numerical integration methods from one to three orders of magnitude in terms of simulation time, while still obtaining accurate results according to the prescribed error tolerance. Taking into account that building simulated trajectories can span from hours (like in the examples analyzed here) to years, the advantages observed projected on longer simulation runs may imply that a simulation with QSS takes a few minutes while that of the fastest classic solver takes several hours.

These results can be explained by three reasons: First, QSS methods are more efficient to handle discontinuities, as they only need to perform a localized re-initialization after their occurrence. Secondly, QSS methods also exploit localized activity by performing calculations only where significant changes occur, which is a key feature in large weakly connected systems like those presented in this article. Finally, LIQSS2 allows to simulate some stiff models in an explicit way without adding computational costs.

The last remark shows an additional advantage of LIQSS2. As the computational costs of their steps are not different from those of QSS2, it can be used as a default QSS solver even for non stiff cases, as it can be seen in Fig. 4. In classic solvers, however, implicit algorithms like DASSL have a computational cost per step much larger than that of explicit solvers like DOPRI. Thus, using DASSL as the default algorithm is safe but may be inefficient in non stiff or mildly stiff models.

Having said that, we do not believe that current QSS methods can actually replace classic solvers like DASSL (or their variants) in general building simulations. We already know that QSS algorithms are inefficient in presence of large non-sparse models since in those models it is convenient to advance the entire state vector in a single step rather than updating individual components (updating individual components is only better in presence of localized activity). Also, LIQSS algorithms are only efficient in presence of certain types of stiff structures, in particular, when the stiffness is due to the presence of large terms at the main diagonal of the Jacobian matrix (Migoni et al. 2015). Thus, there are several cases in which classic solvers like DASSL or DOPRI offer a better performance than QSS algorithms.

Regarding future work, it is very important to establish more clearly when it is convenient to use QSS algorithms and when it is better to use classic solvers. For that goal, the analysis performed in this article should be extended to a wider set of building applications. A first step in this direction would be to test QSS methods in models from the *Modelica Buildings Library* (Wetter and Haugstetter 2006).

Another important issue is that of facilitating the usage of QSS algorithms to the building simulation community. Current implementations of the algorithms are now limited to the Stand Alone QSS solver, which require that the models are defined in the μ-Modelica sub-language. Although there exist some tools that automatically translate Modelica models into μ-Modelica, they are not robust enough to be utilized in a transparent way by final users. Also, we know that Modelica is not the most used modeling language for building simulation. Thus, improving the existing Modelica translation tools and extending them to other modeling languages is an important task to effectively allow the usage of QSS algorithms by the community.

Finally, in spite of the advantages shown by QSS algorithms, several applications in building performance require that the simulations are run for final times corresponding to several days, months and even years. Thus, in spite of the advantages shown by QSS algorithms, the simulation speed they achieve may be insufficient to obtain results in reasonable CPU times. In those cases, some parallelization strategies for QSS algorithms as those recently reported by Fernández et al. (2017) could be explored in the context of building applications.

References

Åkesson J, Gäfvert M, Tummescheit H (2009). Modelica—An open source platform for optimization of Modelica models. In: Proceedings of the 6th Vienna International Conference on Mathematical Modelling.

Baetens R, De Coninck R, Jorissen F, Picard D, Helsen L, Saelens D (2015). OpenIDEAS—An open framework for integrated district energy assessments. In: Proceedings of the 14th International IBPSA Building Simulation Conference, Hyderabad, India, pp. 347–354.

Bergero F, Kofman E (2011). PowerDEVS: A tool for hybrid system modeling and real-time simulation. *Simulation*, 87: 113–132.

Bergero F, Floros X, Fernández J, Kofman E, Cellier FE (2012). Simulating Modelica models with a stand-alone quantized state systems solver. In: Proceedings of the 9th International Modelica Conference, Munich, Germany, pp. 237–246.

Bergero F, Botta M, Campostrini E, Kofman E (2015). Efficient compilation of large scale Modelica models. In: Proceedings of the 11th International Modelica Conference, Versaille, France, pp. 449–458.

Brück D, Elmqvist H, Mattsson SE, Olsson H (2002). Dymola for multi-engineering modeling and simulation. In: Proceedings of the 2nd Modelica Conference, Oberpfaffenhofen, Germany.

Cellier F, Floros XF, Kofman E (2013). The complexity crisis: Using modeling and simulation for system level analysis and design. In: Proceedings of the 3rd International Conference on Simulation and and Modeling Methodologies, Technologies and Applications (SimulTech), Reykjavik, Iceland.

Cellier F, Kofman E (2006). Continuous System Simulation. New York: Springer.

Ceriani NM, Vignali R, Piroddi L, Prandini M (2013). An approximate dynamic programming approach to the energy management of a building cooling system. In: Proceedings of European Control Conference, Zurich, Switzerland, pp. 2026–2031.

CIBSE (2006). CIBSE Guide A: Environmental Design. Norwich, UK: CIBSE Publications.

Crawley DB, Lawrie LK, Winkelmann FC, Buhl WF, Huang YJ, et al. (2001). EnergyPlus: Creating a new-generation building energy simulation program. *Energy and Buildings*, 33: 319–331.

Dormand JR, Prince PJ (1980). A family of embedded Runge-Kutta formulae. *Journal of Computational and Applied Mathematics*, 6: 19–26.

Fernández J, Kofman E (2014). A stand-alone quantized state system solver for continuous system simulation. *Simulation*, 90: 782–799.

Fernández J, Kofman E, Bergero F (2017). A parallel Quantized State System Solver for ODEs. *Journal of Parallel and Distributed Computing*, 106: 14–30.

Floros X, Bergero F, Ceriani N, Casella F, Kofman E, Cellier FE (2014). Simulation of smart-grid models using quantization-based integration methods. In: Proceedings of the 10th International Modelica Conference, Lund, Sweden, pp. 787–797.

Frances VMS, Escriva EJS, Ojer JMP (2014). Discrete event heat transfer simulation of a room. *International Journal of Thermal Sciences*, 75: 105–115.

Frances VMS, Escriva EJS, Ojer JMP (2015). Discrete event heat transfer simulation of a room using a Quantized State System of order two, QSS2 integrator. *International Journal of Thermal Sciences*, 97: 82–93.

Fritzson P (2015). Principles of Object-Oriented Modeling and Simulation with Modelica 3.3: A Cyber-Physical Approach, 2nd edn. Piscataway, NJ, USA: Wiley-IEEE Press.

Fritzson P, Aronsson P, Lundvall H, Nystrom K, Pop A, Saldamli L, Broman D (2005). The OpenModelica modeling, simulation, and development environment. In: Proceedings of the 46th Conference on Simulation and Modeling (SIMS'05), Trondheim, Norway, pp. 83–90.

Fuchs M, Constantin A, Lauster M, Remmen P, Streblow R, Müller E (2015). Structuring the building performance Modelica model library AixLib for open collaborative development. In: Proceedings of the 14th International IBPSA Building Simulation Conference, Hyderabad, India, pp. 331–338.

Grinblat GL, Ahumada H, Kofman E (2012). Quantized state simulation of spiking neural networks. *Simulation*, 88: 299–313.

Hairer E, Nørsett S, Wanner G (1993). Solving Ordinary Differential Equations I. Nostiff Problems, 2nd edn. Berlin: Springer.

Hairer E, Wanner G (1996). Solving Ordinary Differential Equations II. Stiff and Differential-Algebraic Problems. Berlin: Springer.

Hindmarsh AC, Brown PN, Grant KE, Lee SL, Serban R, Shumaker DE, Woodward CS (2005). SUNDIALS: Suite of nonlinear and differential/algebraic equation solvers. *ACM Transactions on Mathematical Software*, 31: 363–396.

Jorissen F, Helsen L, Wetter M (2015). Simulation speed analysis and improvements of Modelica models for building energy simulation. In: Proceedings of the 11th International Modelica Conference, Versaille, France, pp. 59–69.

Kofman E (2002). A second-order approximation for DEVS simulation of continuous systems. *Simulation*, 78: 76–89.

Kofman E (2004). Discrete event simulation of hybrid systems. *SIAM Journal on Scientific Computing*, 25: 1771–1797.

Kofman E (2006). A third order discrete event simulation method for continuous system simulation. *Latin American Applied Research*, 36: 101–108.

Kofman E, Junco S (2001). Quantized-state systems: A DEVS approach for continuous system simulation. *Transactions of the Society for Computer Simulation International,* 18: 123–132.

Mattsson SE, Elmqvist H, Otter M (1998). Physical system modeling with Modelica. *Control Engineering Practice*, 6: 501–510.

Migoni G, Kofman E, Bergero F, Fernández J (2015). Quantization-based simulation of switched mode power supplies. *Simulation*, 91: 320–336.

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

Nytsch-Geusen C, Huber J, Ljubijankic M, Rädler J (2013). Modelica BuildingSystems—eine Modellbibliothek zur Simulation komplexer energietechnischer Gebäudesysteme. *Bauphysik*, 35: 21–29.

Perfumo C, Kofman E, Braslavsky JH, Ward JK (2012). Load management: Model-based control of aggregate power for populations of thermostatically controlled loads. *Energy Conversion and Management*, 55: 36–48.

Petzold LR (1982). A description of DASSL: A differential/algebraic system solver. In: Proceedings of Scientific computing, Montreal, Canada, pp. 65–68.

Wetter M, Haugstetter C (2006). Modelica versus TRNSYS—A comparison between an equation-based and a procedural modeling language for building energy simulation. In: Proceedings of the 2nd National IBPSA-USA Conference, Cambridge, MA, USA.

Wetter M, Zuo W, Nouidui TS, Pang X (2014). Modelica Buildings library. *Journal of Building Performance Simulation*, 7: 253–270.

Wetter M, Nouidui TS, Lorenzetti D, Lee EA, Roth A (2015). Prototyping the next generation EnergyPlus simulation engine. In: Proceedings of the 14th International IBPSA Building Simulation Conference, Hyderabad, India.

Wetter M, Bonvini M, Nouidui TS (2016). Equation-based languages—A new paradigm for building energy modeling, simulation and optimization. *Energy and Buildings*, 117: 290–300.

Zeigler B, Praehofer H, Kim TG (2000). Theory of Modeling and Simulation, 2nd edn. San Diego, USA: Academic Press.