Simulation

Simulation: Transactions of the Society for Modeling and Simulation International
2015, Vol. 91(4) 320–336
© 2015 The Author(s)
DOI: 10.1177/0037549715575197
sim.sagepub.com
SAGE SCS

Quantization-based simulation of switched mode power supplies

Gustavo Migoni, Ernesto Kofman, Federico Bergero and Joaquín Fernández

Abstract
In this article we study the performance of quantized state system algorithms in the simulation of switched mode power supplies. Under realistic modeling assumptions, these models are stiff and exhibit frequent discontinuities, making them difficult to simulate with classic solvers. However, there are linearly implicit quantized state system methods that can efficiently handle these types of systems, providing faster and more accurate results. In order to corroborate these features, we first built the models corresponding to the different topologies of switched mode power supplies, and then analyzed the resulting equation structures in order to establish whether they can be efficiently simulated by linearly implicit quantized state system algorithms. Finally, we compared the simulation performance of linearly implicit quantized state systems with the widely used DASSL solver. The results showed that the linearly implicit quantized state systems were 3–200 times faster and noticeably more accurate than the differential algebraic system solver.

Keywords
Hybrid systems simulation, quantized state systems, switched mode power supplies

1. Introduction

Switching mode power supplies (SMPSs) are electronic devices that incorporate switching regulators to convert electrical power efficiently. The output voltage of these sources is regulated by the electronic switches duty cycle and the switching components should work at high frequency to minimize the output ripple.

SMPSs are used in a wide area of applications where a regulated voltage is required. They can be found inside personal computers, battery chargers, vehicles, etc.

Simulation of SMPSs is known to be a tough issue. On the one hand, due to the high switching frequency, classic numerical integration methods become inefficient. To simulate discontinuous models using methods based on time discretization, the solver spends several computations at each simulation step to determine whether there are any changes in the model. Each time the model changes, the simulation must be reinitialized. On the other hand, the usage of realistic models of the discontinuous elements usually leads to stiff models. Thus, implicit methods must be used with their additional computational cost.

In recent years, a new family of ordinary differential equation (ODE) solvers called quantized state system (QSS) methods were developed.[1–5] These algorithms, that replace the classic time discretization by the quantization of the state variables, have shown some advantages.

1. They have strong stability and error bound theoretical properties.[1,2,6]
2. They are very efficient to simulate ODE models with frequent discontinuities.[7] Due to their dense output feature, their built-in root-solving method is explicit and does not require any iteration to detect discontinuities. Moreover, the simulation does not need to be reinitialized after their occurrence. Consequently, detecting and handling a discontinuity does not add more computational cost than that of a regular step.
3. They are very efficient in the simulation of large-scale sparse discontinuous models.[5,8] This is due to the fact that QSS methods intrinsically track the system activity,[9] performing calculations only where and when changes occur.
4. There are linearly implicit QSS (LIQSS) methods that can integrate stiff systems with a certain

CIFASIS, CONICET, Argentina FCEIA, Universidad Nacional de Rosario, Argentina

Corresponding author:
Gustavo Andres Migoni, CIFASIS, CONICET, 27 de febrero 210 bis, Rosario 2000, Argentina.
Email: migoni@cifasis-conicet.gov.ar

structure in a very efficient way, without performing iterations or matrix inversions.[5]

For these reasons, the QSS methods (and particularly LIQSS algorithms) seem to be a good option to efficiently simulate SMPSs. Moreover, LIQSS methods have shown important advantages over classic solvers on the simulation of buck converters[5] and interleaved buck converters.[10]

In this paper we analyze the performance and features of LIQSS methods in the simulation of SMPSs. Specifically:

1. an exhaustive analysis of the different SMPS topologies is performed to check whether their structure is appropriate to be efficiently integrated with LIQSS algorithms;
2. it is shown that in most cases the topologies are adequate and, when they are not, a simple change of variables can be applied to obtain an appropriate structure;
3. a comparative analysis of computational costs and simulation errors using LIQSS and the widely used DASSL is performed for the different SMPS topologies; we have used DASSL because SMPS models are stiff and discontinuous, conditions under which this solver offers the best performance among other classic algorithms usually implemented in simulation software tools.
4. a study about the growth of the computational cost with the circuit size is also performed on an interleaved buck topology.

The article is organized as follows: Section 2 provides the background concepts that are used in the rest of the article. Section 3 introduces the topologies and models of SMPSs, analyzing also whether they are appropriate for LIQSS simulation. Then, Section 4 presents the simulation results for the different topologies. Finally, Section 5 concludes the article, analyzing future lines of research.

2. Background

This section provides the background required for the rest of the article. We give a brief description of the problems suffered by classic numerical integration algorithms when dealing with discontinuous systems. Then, we present the family of QSS methods and the software tools that implement them, and finally we provide a brief introduction to SMPS principles.

2.1. Hybrid system simulation

Hybrid systems exhibit both continuous and discrete dynamic behavior. The interaction between the continuous and discrete sub-models may produce sudden changes (discontinuities) in the continuous parts that must be handled by the numerical integration algorithms. These discontinuities are called events, and two different cases can be distinguished according to the nature of their occurrence. The events that occur at a given time, independently of what happens in the continuous part, are called time events. On the other hand, the events triggered when some condition is reached by the continuous states are called state events.

It is well known that integration along discontinuities may lead to disastrous results on the global simulation solution because the theoretical assumptions on which solvers are founded are not met. To avoid this, the events must be detected and handled. When a discontinuity is detected, the simulation time must be advanced until the exact time of its occurrence and then, after processing the event, the simulation should be restarted in a new situation.[6]

Event detection is straightforward for time events, as it is known in advance when they occur. However, state events require the usage of iterative routines in order to find the time at which the event condition is met.

The whole process of event detection and handling adds extra computational cost to the simulation. In systems with frequent discontinuities, i.e. when events occur as fast as the system dynamics the problem becomes critical, as the algorithms spend more time with the event detection and handling routines than with the numerical integration itself.

2.2. QSS methods

Consider a time invariant ODE in its state equation system (SES) representation

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

where $\mathbf{x}(t) \in \mathbb{R}^n$ is the state vector and $\mathbf{u}(t) \in \mathbb{R}^m$ is an input vector, which is a known piecewise constant function.

The first-order QSS (QSS1) method[1] analytically solves an approximate ODE called a *Quantized State System* that results from replacing the state vector $\mathbf{x}(t)$ by its quantized version $\mathbf{q}(t)$

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

Each component of $\mathbf{q}(t)$ is related with the corresponding component of $\mathbf{x}(t)$ by the following hysteric quantization function

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

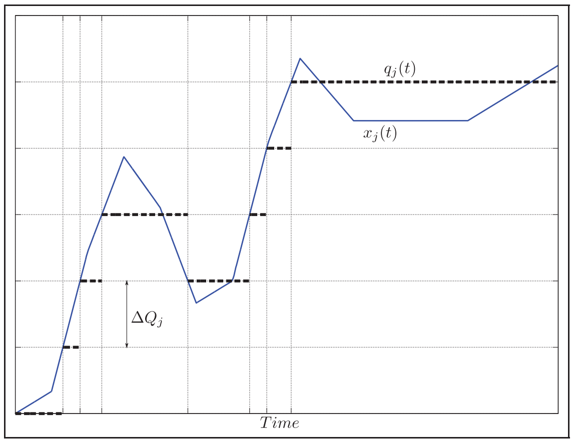

Figure 1. State and quantized trajectories in the QSS1 method.

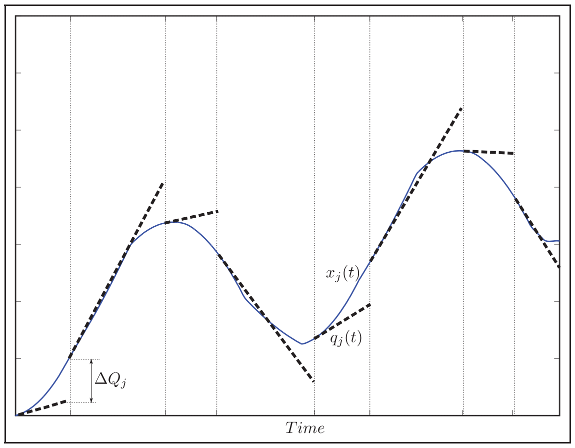

Figure 2. State and quantized state trajectories in QSS2.

That is, $q_j(t)$ only changes when it differs from $x_j(t)$ by a magnitude ΔQ_j defined as the *quantum*. After each change in the quantized variable, it takes the same value of the state variable, $q_j(t) = x_j(t)$.

Since the quantized state trajectories $q_j(t)$ are piecewise constant, the state derivatives $\dot{x}_j(t)$ also follow piecewise constant trajectories and, consequently, the states $x_j(t)$ follow piecewise linear trajectories. Figure 1 shows typical QSS1 trajectories.

Due to the particular form of the trajectories, the numerical solution of equation (2) is straightforward and can be easily translated into a simple simulation algorithm.

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

1. Advance the simulation time t to the minimum t_j.
2. Recompute $x_j(t) = x_j(t_j^-) + \dot{x}_j(t_j^-) \cdot (t - t_j^-)$, where t_j^- was the last update time of x_j, and $\dot{x}_j(t_j^-)$ was computed at time t_j^- from equation (2).
3. Take $q_j = x_j$ and recompute t_j (the next time at which $|q_j - x_j| = \Delta Q_j$).
4. For all i such that $\dot{x}_i$ explicitly depends on q_j, update $x_i(t) = x_i(t_i^-) + \dot{x}_i(t_i^-) \cdot (t - t_i^-)$, recompute $\dot{x}_i(t)$, and recalculate t_i (the next time at which $|q_i - x_i| = \Delta Q_i$).
5. Go back to step 1.

Note that the instant of time at which the piecewise linear state trajectory $x_j(t)$ crosses a given threshold can be computed without iterations. Thus, it is straightforward to detect state events. Moreover, when an event occurs, it will eventually change some state derivatives in the same way a change in a quantized variable does during a normal step. That way, the simulation does not need to be restarted.

In conclusion, the detection and handling of a discontinuity does not take more computational effort than that of a single step. Thus, the QSS1 method is very efficient in simulating discontinuous systems.[7]

In spite of this advantage and the fact that it has some favorable stability and error bound properties,[6,11] QSS1 performs only a first-order approximation and it cannot obtain accurate results without significantly increasing the number of steps.

This accuracy limitation was improved by defining the second- and third-order accurate QSS methods called QSS2[2] and QSS3,[3] respectively.

QSS2 and QSS3 have the same definition as QSS1 (equation (2)) except the components of $\mathbf{q}(t)$, which are calculated to follow piecewise linear and piecewise parabolic trajectories, respectively. Figure 2 shows a typical evolution of the state and quantized state trajectories in QSS2.

As seen previously, the analytical solution of equation (2) for QSS2 and QSS3 can be easily computed and the simulation algorithm is almost identical to that of QSS1. Consequently, QSS2 and QSS3 also share the advantages of QSS1 when simulating discontinuous systems.

Regardless of these advantages, QSS1, QSS2, and QSS3 methods are very inefficient for simulating stiff systems. In the presence of simultaneous slow and fast dynamics, these methods introduce spurious high-frequency oscillations that provoke a large number of steps with its consequent computational cost.

To overcome this problem, the family of QSS methods was completed with a set of algorithms called LIQSS, which are appropriate to simulate some stiff systems.[5]

LIQSS methods combine the principles of QSS methods with those of linearly implicit algorithms.[6] There are LIQSS algorithms that perform first-, second-, and third-order accurate approximations named LIQSS1, LIQSS2, and LIQSS3, respectively.

The main idea behind LIQSS methods is inspired by classic implicit methods that evaluate the state derivatives at future instants of time. In classic methods, these evaluations require iterations and/or matrix inversions to solve the resulting implicit equations. However, taking into account that QSS methods know the future value of the quantized state (it is $q_j(t) \pm \Delta Q_j$)), the implementation of LIQSS algorithms is explicit and does not require iterations or matrix inversions.

LIQSS methods share the definition of equation (2) with QSS methods, but the quantization functions that relate q_j with x_j are more involved. The resulting simulation algorithm is similar to that of the QSS methods, and they are also advantageous in the simulation of discontinuous systems.

Despite being explicit algorithms, LIQSS methods are able to integrate many stiff systems. In order to work efficiently, they require the stiffness to be caused by large entries in the main diagonal of the Jacobian matrix.

2.3. Implementation of QSS methods

The easiest way of implementing QSS methods is by building an equivalent discrete event system specification (DEVS) model, where the events represent changes in the quantized variables. Based on this idea, the whole family of QSS methods were implemented in PowerDEVS,[12] a DEVS-based simulation platform specially designed for and adapted to simulating hybrid systems based on QSS methods. In addition, the explicit QSS1, QSS2, and QSS3 methods were also implemented in a DEVS library of Modelica[13] and implementations of the first-order QSS1 method can also be found in CD + +[14] and VLE.[15]

DEVS-based implementations of QSS methods are simple but they are not efficient.

Recently, the complete family of QSS methods was implemented in a *stand-alone QSS solver*[10] that improves DEVS-based simulation times by more than one order of magnitude.

The stand-alone QSS solver requires that the models are described in a subset of the Modelica modeling language,[16] called μ-Modelica.[10]

2.4. Switched mode power supplies

Switched mode power supplies[17] are electronic devices that convert the available DC input voltage into a different DC or AC output voltage by switching commutation components at high frequency. These components are implemented in circuits by transistors or thyristors operating in cutoff and saturation states.

Due to their high efficiency, SMPSs are widely employed in a variety of applications, including power supplies for personal computers, battery chargers, telecommunications equipment, DC motor drives, etc.

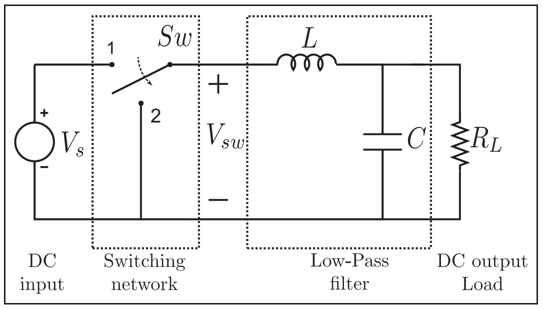

Figure 3. Ideal schematic of a buck converter.

The majority of the SMPS topologies used in today's power converters are all derived from the following three non-isolated circuits.

1. Buck converter: which reduces the input voltage ($V_{\text{out}} < V_{\text{in}}$).
2. Boost converter: which increases the input voltage ($V_{\text{out}} > V_{\text{in}}$).
3. Buck–boost converter: which can increase or decrease the input voltage. The Cuk circuit is a variant of this converter.

The output voltage in these topologies is regulated by controlling the relationship between the amount of time the switches are in *ON state* and *OFF state*, i.e. by controlling the *duty cycle*.

A simplified circuit scheme for the buck converter is illustrated in Figure 3.

The input voltage source V_s can be found on the left of the circuit. The input source is connected to the switching stage, which generates a high-frequency square signal V_{sw}, with a mean value proportional to the duty cycle. This frequency and duty cycle are determined by the commutation times of the switch.

The next stage is an LC low pass filter that produces an output voltage V_o. This stage preserves the mean value of V_{sw}, but reduces the high frequency components. Therefore, the voltage at the load has a continuous value with a small undesired high frequency component called *ripple*.

In order to reduce the ripple, the switching frequency should be very large in comparison with the dynamics of the low pass filter.

In real applications, the ideal switch depicted in Figure 3 is implemented by real components like transistors and diodes. Some limitations appear and, for instance, the current on the inductance L cannot become negative since it is blocked by a diode. In such case, the circuit is said to operate in *discontinuous mode*.

The remaining topologies (boost, buck–boost, etc.) work under similar principles.

A drawback of these topologies is that they require working at very high frequencies in order to obtain a low ripple. To overcome this problem, there are *interleaved* versions of the SMPS.

Interleaved converters are the result of a parallel connection of switching converters.[18] They offer several advantages over single power stage converters; a lower current ripple, faster transient response to load changes, and improved power handling capabilities. Thus, they are widely used in several applications requiring a high-quality input voltage, including power sources of personal computers, switching audio amplifiers, etc.

3. Models for switched mode power supplies

In this section, we develop simulation models for different topologies of SMPSs. We first introduce mathematical models for the commutation components (switches and diodes) and then we derive the circuit equations and translate them into μ-Modelica descriptions. Finally, we analyze the structure of the models in order to verify that the resulting stiff systems are suitable to be simulated by LIQSS methods.

3.1. Modeling switching components of SMPS

SMPSs commutation components can be represented following two basic approaches.

1. They can be represented by commuting from ideal short circuits to ideal open circuits according to their *ON* or *OFF* state. This approach leads to variable structure models, and different sets of state equations are obtained for the different situations. If a circuit has N commutation components, it can be configured in 2^N combinations according to the switches states, and the model must be described by 2^N sets of equations. In order to simulate these types of model, the simulation tools must be able to handle variable structure systems. In addition to these disadvantages, the ideal model of switching components has a lack of realism and may hide some features of the circuit behavior. This approach is still used by some circuit simulation software tools like PLECS,[19] and it has the advantage of avoiding stiffness which allows the usage of fast explicit numerical algorithms.
2. The second approach represents switching components as resistors with low or high value according to their *ON* or *OFF* state. In this way, the system equations are always the same and the only thing that changes after commutations are the values of certain parameters. Owing to its simplicity and realism, this is the approach followed by most simulation tools like PSPICE,[20] and its variants, and those based on Modelica.[21] However, this approach usually leads to stiff systems and it requires the usage of implicit stiff–stable numerical solvers.

In this article we focus on the second approach. It is the most realistic, it is used by most simulation tools, and it offers more difficulties from the numerical integration point of view.

SMPS circuits have two basic switching components: controlled switches (usually implemented by transistors or thyristors) and diodes. Their simplified models are developed below.

3.1.1. Controlled switch model.

A controlled key is an element that acts like an open circuit or short circuit according to the state of a control signal. As discussed above, this element can be modeled as a resistor R_s with a high or low value according to the control signal. This behavior is represented by the following equation

$$R_\mathrm{s} = \begin{cases} R_\mathrm{On} & \text{if } control = 1 \\ R_\mathrm{Off} & \text{if } control = 0 \end{cases} \tag{3}$$

where R_On and R_Off are very low and very large resistance values, respectively.

This behavior can be easily represented in terms of the μ-Modelica language. It corresponds to two event handlers that are triggered when the control signal changes.

```
when control > 0.5 then
  Rs := ROn;
end when;

when control < 0.5 then
  Rs := ROff;
end when;
```

3.1.2. Diode model.

Figure 4 shows the current–voltage characteristic of a real diode on the left side and a piecewise linear approximation on the right side. The latter is the result of representing the *OFF* state by a large resistance and the *ON* state by a small resistance.

According to this figure, the value of the diode resistance R_D obeys the law

$$R_\mathrm{s} = \begin{cases} R_\mathrm{On} & \text{if } u_D > 0 \\ R_\mathrm{Off} & \text{if } u_D \leqslant 0 \end{cases}$$

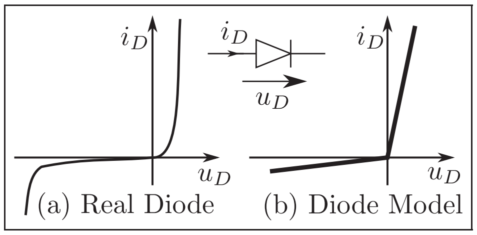

Figure 4. Real and approximate diode characteristics.

which leads to the following μ-Modelica representation

```
when uD > 0 then
  RD := ROn;
end when;

when uD < 0 then
  RD := ROff;
end when;
```

However, the detection of the crossing condition $u_D = 0$ when the diode is in *ON* state is very difficult due to numerical issues. The reason is that the voltage u_D is very small when R_D is very small which leads to large errors in the detection of the condition $u_D = 0$. Thus, when the diode is in *ON* state the event detection is performed using the current i_D, which leads to the following model:

```
when uD > 0 then
  RD := ROn;
end when;

when iD < 0 then
  RD := ROff;
end when;
```

3.2. Models for the different topologies

As we mentioned above, there are three basic topologies of SMPS. We shall obtain below their equations and μ-Modelica representations using the switching models obtained before.

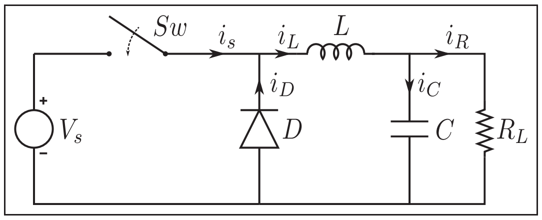

Figure 5. Buck converter circuit.

3.2.1. Buck converter. As was described before, the buck converter circuit is a switched converter that generates an output voltage lower than the input voltage.

The basic scheme of this converter was shown in Figure 3. The two point switch is implemented in real application by a transistor (controlled switch) and a diode as shown in Figure 5.

Representing the switch and diode by resistors R_s and R_D as discussed above, the following space state representation of the circuit can be obtained

$$\begin{aligned}\frac{\mathrm{d}i_L}{\mathrm{d}t} &= \frac{-i_D R_D - u_C}{L}\\ \frac{\mathrm{d}u_C}{\mathrm{d}t} &= \frac{i_L - u_C/R}{C}\end{aligned} \qquad (4)$$

where

$$i_D = \frac{i_L R_s - U}{R_s + R_D} \qquad (5)$$

Joining equations (4) and (5), the following μ-Modelica code represents the continuous equations of the model

```
equation
  der (iL) = (-iD*RD-uC)/L;  //ODE Equations
  der(uC) = (iL-uC/Rl)/C;
  iD=(iL*Rs-U)/(Rs + RD);   //Diode equations
  uD=iD*RD;
```

The μ-Modelica representation of the switch and diode commutation laws completes the model as follows

```
algorithm
  when time > nextT then              //Switch ON time event
    lastT:=nextT;
    nextT:=nextT + T;
    Rs := ROn;                        //control=1
  end when;

  when time - lastT-DC*T > 0 then     //Switch OFF time event
    Rs := ROff;                       //control=0
  end when;

  when iD < 0 then    //Diode OFF state event
    RD := ROff;
  end when;

  when uD > 0 then  //Diode ON state event
    RD := ROn;
  end when;
```

In this last model, we included the generation of the control signal for the controlled switch. This control signal takes the value 1 (*ON* state) every T units of time and switches to 0 after $D_C T$ units of time, where D_C is the duty cycle mentioned previously.

The usage of this control signal corresponds to an *open loop* output voltage regulation. In many applications, a *closed loop* strategy is preferred, where the control signal is computed by comparing a reference with the output voltage to obtain an automatically adjusted duty cycle.

The usage of open or closed loop strategies does not introduce any significant difference from a numerical point of view, so we work here with the simpler open-loop scheme.

3.2.2. Boost converter. The boost converter circuit, shown in Figure 6, is a switched converter that generates an output voltage higher than the input voltage.

Proceeding in the same manner as with the buck converter, the following state equations can be obtained

$$\begin{aligned}\frac{\mathrm{d}i_L}{\mathrm{d}t} &= \frac{-R_s i_L + R_s i_D + U}{L}\\ \frac{\mathrm{d}u_C}{\mathrm{d}t} &= \frac{i_D}{C} - \frac{u_C}{R_L C}\end{aligned} \tag{6}$$

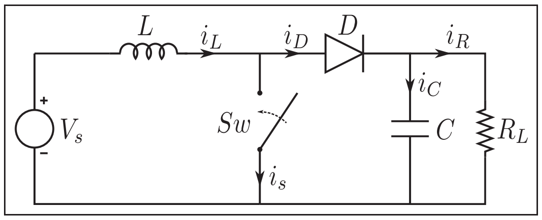

Figure 6. Boost converter circuit.

where

$$i_D = \frac{R_s i_L - u_C}{R_D + R_s} \tag{7}$$

These equations can be translated into the following μ-Modelica code

```
equation
  der(iL)=(U-Rs*(iL-iD))/L;  //ODE Equations
  der(uC)=(iD- uC/Rl)/C;
  iD=(Rs*iL-uC)/(RD + Rs);    //Diode equation
  uD=iD*RD;
```

The μ-Modelica representation of the switch and diode commutation laws is identical to that of the buck converter.

3.2.3. Buck–boost converter. Figure 7 shows the buck–boost converter circuit. In this converter, the output voltage magnitude can be higher or lower than the input voltage according to the duty cycle. The output voltage polarity is always opposite to that of the input.

Proceeding as before, the following state equations can be derived

$$\begin{aligned}\frac{\mathrm{d}i_L}{\mathrm{d}t} &= \frac{-u_C - R_D i_D}{L}\\ \frac{\mathrm{d}u_C}{\mathrm{d}t} &= \frac{i_D}{C} - \frac{u_C}{R_L C}\end{aligned} \tag{8}$$

where

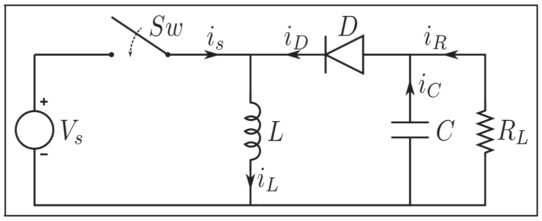

Figure 7. Buck–boost converter circuit.

$$i_D = \frac{R_s i_L - u_C - U}{R_D + R_s} \tag{9}$$

Then, the continuous equations of the model can be written in μ-Modelica language as:

```
equation
  der(iL) = (-uC-iD*RD)/L;    //ODE Equations
  der(uC) = (iD- uC/Rl)/C;
  iD=(Rs*iL-uC-U)/(RD + Rs); //Diode equation
  uD=iD*RD;
```

The μ-Modelica representation of the switch and diode commutation laws is identical to those of the buck and boost converters.

3.2.4. Cuk converter. The Cuk converter circuit is a variant of the buck–boost converter that also generates an output voltage magnitude that can be greater than or less than the input voltage magnitude, but with opposed polarity. The basic Cuk converter circuit is shown in Figure 8.

The state equations for this Cuk converter are

$$\begin{aligned}
\frac{\mathrm{d}i_{L_1}}{\mathrm{d}t} &= \frac{U - u_{C_1} - R_D i_D}{L_1}\\
\frac{\mathrm{d}u_{C_1}}{\mathrm{d}t} &= \frac{i_D - i_{L_2}}{C_1}\\
\frac{\mathrm{d}i_{L_2}}{\mathrm{d}t} &= \frac{-u_{C_2} - R_D i_D}{L_2}\\
\frac{\mathrm{d}u_{C_2}}{\mathrm{d}t} &= \frac{R_L i_{L_2} - u_{C_2}}{R_L C_2}
\end{aligned} \tag{10}$$

where

$$i_D = \frac{R_s(i_{L_2} + i_{L_1}) - u_{C_1}}{R_D + R_s} \tag{11}$$

The corresponding μ-Modelica code for these equations is

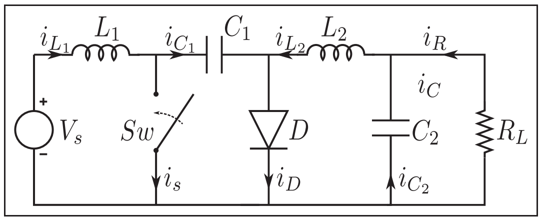

Figure 8. Cuk converter circuit.

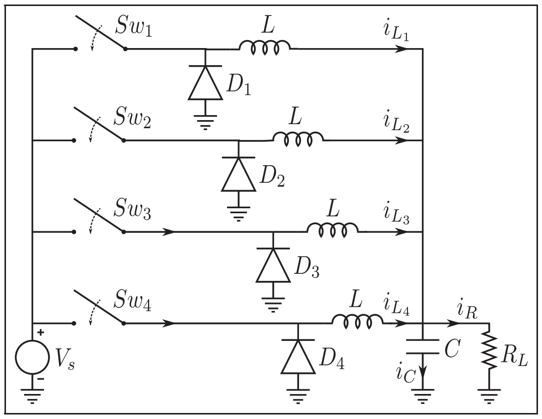

Figure 9. Four-stage buck interleaved converters.

```
equation
  der(iL1) = (U-uC1-iD*RD)/L1;    //ODE
  Equations
  der(uC1) = (iD - iL2)/C1;
  der(iL2) = (-uC2-iD*RD)/L2;
  der(uC2) = (iL2 - uC2/Rl)/C2;
  iD=(Rs*(iL2 + iL1)-uC1)/(RD + Rs);
    //Diode Equations
  uD=iD*RD;
```

The μ-Modelica representation of the switching laws is identical to that of the previous converters.

3.3. Interleaved converters

Figure 9 shows the circuit corresponding to a four–stage buck interleaved converter. In this circuit, each period is divided by four. During each sub-period only one stage is in charge of switching *ON* and *OFF* in order to feed the load while the other stages remain in *OFF* state. That way, the switching frequency off the whole circuit is four times faster than that of the individual stages.

The state equations for a N-stage buck converter circuit can be written as follows

$$\begin{aligned}\frac{\mathrm{d}i_{L_j}}{\mathrm{d}t} &= \frac{-i_{D_j}R_{D_j} - u_C}{L} \quad \text{for } j = 1, \ldots, N \\ \frac{\mathrm{d}u_C}{\mathrm{d}t} &= \frac{\sum_{j=1}^{N} i_{L_j}}{C} - \frac{u_C}{R_L C}\end{aligned} \tag{12}$$

with

$$i_{D_j} = \frac{i_{L_j}R_{s_j} - U}{R_{s_j} + R_{D_j}} \tag{13}$$

which can be written in μ-Modelica as

```
equation
  for i in 1:N loop
    der(iL[i]) = (-iD[i]*RD[i]-uC)/L;
    iD[i]=(iL[i]*Rs[i]-U)/
(Rs[i] + RD[i]);
    uD[i]=iD[i]*RD[i]s;
  end
  der(uC) = (sum(iL)-uC/Rl)/C;
```

The μ-Modelica representation of the switches and diodes commutation laws completes the model as follows

3.4. Stiffness analysis

Stiffness is related to the simultaneous presence of fast and slow dynamics in a system. In linear systems, this feature can be analyzed by observing the real part of the eigenvalues λ_i of the Jacobian matrix $J \triangleq \partial\mathbf{f}/\partial\mathbf{x}$.

Although the converters analyzed here are nonlinear systems, their models between switching times are linear. Moreover, taking into account that the switches and diodes are modeled as resistors with changing parameters, the Jacobian matrix of each converter has the same expression, independent of the condition on the switches.

The fact that switching components are modeled as very large or small resistors introduces large terms in the system Jacobian matrices, which in turn result in the presence of some very large eigenvalues.

When these large terms are only located in the main diagonal of the Jacobian matrix, the LIQSS methods can efficiently integrate the resulting stiff model. Otherwise, they introduce spurious oscillations which add a significant computational load.

Based on these observations, we can now analyze the Jacobian matrices resulting from the different topologies, in order to determine which models are suitable for integration with LIQSS algorithms.

```
algorithm
  when time > nextT then   //Start of a new period
    lastT:=nextT;
    nextT:=nextT + T;
  end when;

  for i in 1:N loop
    when time-lastT-T*(i-1)/N-T/100 > 0 then
      Rs[i] := ROn;          //Switch ON time event
    end when;
  end for;
  for i in 1:N loop
    when time - lastT-T*(i-1)/N-DC*T/N-T/100 > 0 then
      Rs[i] := ROff;          //Switch OFF time event
    end when;
  end for;
  for i in 1:N loop
    when iD[i] < 0 then
      RD[i] := ROff;          //Diode OFF state event
    end when;
  end for;
  for i in 1:N loop
    when uD[i] > 0 then
      RD[i] := ROn;           //Diode ON state event
    end when;
  end for;
```

3.4.1. Buck converter. By replacing i_D in equation (4), with the expression of equation (5), we obtain the following Jacobian matrix:

$$J_{\text{Buck}} = \begin{bmatrix} \frac{-R_s R_D}{L(R_s + R_D)} & \frac{-1}{L} \\ \frac{1}{C} & \frac{-1}{R_L C} \end{bmatrix} \tag{14}$$

In this case, the switch and diode resistances, R_s and R_D only appear on the main diagonal. Thus, when they take a large value (R_{OFF}) a large value will only appear on the main diagonal.

Hence, if stiffness appears as a cause of the switching components, it will be due to a large term in the main diagonal and it will be properly handled by LIQSS methods.

3.4.2. Boost converter. Proceeding as before, from equations (6) and (7), the Jacobian matrix is given by

$$J_{\text{Boost}} = \begin{bmatrix} \frac{-R_s R_D}{(R_D + R_s)L} & \frac{-R_s}{(R_D + R_s)L} \\ \frac{R_s}{C(R_D + R_s)} & -\frac{R_L + R_D + R_s}{C(R_D + R_s)R_L} \end{bmatrix} \tag{15}$$

Here, R_s and R_D also appear outside the main diagonal. However, in these matrix entries, the switch resistances only appear in the expression

$$\frac{R_s}{R_s + R_D} \tag{16}$$

which always has a value less than 1. Thus, terms with order of magnitude R_{OFF} or $1/R_{\text{ON}}$ cannot appear outside the main diagonal.

Consequently, the switch resistances cannot introduce stiffness that is not well handled by LIQSS methods.

3.4.3. Buck–boost converter. Using equations (8) and (9) we arrive at the same Jacobian matrix as that of the boost converter given by equation (15). Thus, the buck–boost converter can also be efficiently integrated using LIQSS algorithms.

3.4.4. Cuk converter. Proceeding as previously, from equations (10) and (11) the following Jacobian matrix is obtained

$$J_{\text{Cuk}} = \begin{bmatrix} \frac{-R_s R_D}{(R_D + R_s)L1} & \frac{-R_s}{(R_D + R_s)L1} & \frac{-R_s R_D}{(R_D + R_s)L1} & 0 \\ \frac{R_s}{(R_D + R_s)C_1} & \frac{-1}{(R_D + R_s)C_1} & \frac{-R_D}{(R_D + R_s)C_1} & 0 \\ \frac{-R_s R_D}{L_2(R_s + R_D)} & \frac{R_D}{L_2(R_s + R_D)} & \frac{-R_s R_D}{L_2(R_s + R_D)} & \frac{-1}{L_2} \\ 0 & 0 & \frac{1}{C_2} & \frac{-1}{R_L C_2} \end{bmatrix} \tag{17}$$

Here, we have several terms outside the main diagonal that depend on the switch resistances. In some of them, these resistance appear within the expression of equation (16). In others we have

$$\frac{R_D}{R_s + R_D}$$

which can be analyzed in a similar way.

However, there are two entries outside the main diagonal that have the expression

$$\frac{R_s R_D}{R_s + R_D}$$

which, when $R_s = R_D = R_{\text{OFF}}$, result in a very large value of (R_{OFF} /2).

Unfortunately, LIQSS methods are not ensured to work efficiently in the presence of stiffness, with large terms outside the main diagonal. Thus, the algorithms may introduce spurious oscillations when simulating this circuit.

However, a simple change of variables can be introduced to overcome this problem. By defining

$$i_{12} \overset{\Delta}{=} \frac{L_1 \cdot i_{L_1} - L_2 \cdot i_{L_2}}{L_2}$$

and removing i_{L_1} from equations (10) and (11), these equations become

$$\begin{aligned} \frac{\mathrm{d}i_{12}}{\mathrm{d}t} &= \frac{U + u_{C_2} - u_{C_1}}{L_2} \\ \frac{\mathrm{d}u_{C_1}}{\mathrm{d}t} &= \frac{i_D - i_{L_2}}{C_1} \\ \frac{\mathrm{d}i_{L_2}}{\mathrm{d}t} &= \frac{-u_{C_2} - i_D R_D}{L_2} \\ \frac{\mathrm{d}u_{C_2}}{\mathrm{d}t} &= \frac{i_{L_2} - u_{C_2}/R_L}{L_2} \end{aligned} \tag{18}$$

where

$$\begin{aligned} i_{L_1} &= \frac{L_2 i_{12} + L_2 i_{L_2}}{L_1} \\ i_D &= \frac{R_s(i_{L_2} + i_{L_1}) - u_{C_1}}{R_D + R_s} \end{aligned} \tag{19}$$

Then, the new Jacobian matrix is

$$J_{\text{Cuk}_2} = \begin{bmatrix} 0 & \frac{-1}{L_2} & 0 & \frac{1}{L_2} \\ \frac{L_2 R_s}{L_1 C_1(R_D + R_s)} & \frac{-1}{C_1(R_D + R_s)} & \frac{R_s L_2 - R_D L_1}{L_1 C_1(R_D + R_s)} & 0 \\ \frac{-R_D R_s}{L_1(R_D + R_s)} & \frac{R_D}{L_2(R_D + R_s)} & \frac{-R_D R_s(L_1 + L_2)}{L1L2(R_D + R_s)} & \frac{-1}{L_2} \\ 0 & 0 & \frac{1}{C_2} & \frac{-1}{C_2 R_L} \end{bmatrix} \tag{20}$$

which still has one term that can take values of the order of R_{OFF} outside the main diagonal. However, in the presence of large terms restricted to the lower or upper triangular sub-matrix, LIQSS methods can still integrate efficiently.

This fact will be corroborated later with the simulation results.

The μ-Modelica code for the new set of equations is as follows

```
equation
  iL1=(L2*i12 + L2*iL2)/L1;
  iD=(Rs*(iL2 + iL1)-uC1)/(Rd + Rs);
  der(uC1) = (iD - iL2)/C1;
  der(i12) = (U + uC2-uC1)/L2;
  der(uC2) = (iL2 - uC2/Rl)/C2;
  der(iL2) = (-uC2-iD*Rd)/L2;
```

3.4.5. Interleaved buck converter. From equations (12) and (13), the following Jacobian matrix is obtained for a N-stage interleaved buck converter

$$J_{\mathrm{Int}} = \begin{bmatrix} \frac{-R_{D_1}R_{s_1}}{L(R_{D_1}+R_{s_1})} & 0 & 0 & \cdots & 0 & \frac{-1}{L} \\ 0 & \frac{-R_{D_2}R_{s_2}}{L(R_{D_2}+R_{s_2})} & 0 & \cdots & 0 & \frac{-1}{L} \\ 0 & 0 & \frac{-R_{D_3}R_{s_3}}{L(R_{D_3}+R_{s_3})} & \cdots & 0 & \frac{-1}{L} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & \frac{-R_{D_N}R_{s_N}}{L(R_{D_N}+R_{s_N})} & \frac{-1}{L} \\ \frac{1}{C} & \frac{1}{C} & \frac{1}{C} & \cdots & \frac{1}{C} & \frac{-1}{R_L C} \end{bmatrix} \qquad (21)$$

It can be seen that the switch and diode resistances R_s and R_D only appear on the main diagonal, and we arrive at the same conclusions as with the buck converter.

Note also that the Jacobian is a sparse matrix.

4. Simulation results

This section shows the simulation results, comparing the performance of the LIQSS methods with the classic solver DASSL in the simulation of the five SMPS models presented before.

In order to perform this comparison, we run a set of experiments according to the conditions described below.

1. We simulated all sources under two different error tolerance settings: rel.tol. = abs.tol = 10^{-3} and rel.tol. = abs.tol = 10^{-5}.
2. All of the simulations were performed until a final time $t_f = 0.01$ s.
3. The simulations were performed on an Intel i7-3770@3.40 GHz PC under Ubuntu OS.
4. LIQSS results were obtained with the QSS stand-alone solver described previously.
5. DASSL results were obtained with DASSRT code, using an interface provided by the QSS stand-alone solver, so the models simulated by DASSL and LIQSS were exactly the same.
6. The systems were also simulated with OpenModelica and Dymola implementations of DASSL. However, the direct use of DASSRT reported faster results than those of the mentioned simulation tools. Thus, only DASSRT results are reported.
7. In all cases, we measured the CPU time, the number of scalar function evaluations, the number of Jacobian computations, and the relative error, computed as

$$e_{rr} = \sqrt{\frac{\sum (u_C[k] - u_{C_{REF}}[k])^2}{\sum u_{C_{REF}}[k]^2}} \qquad (22)$$

where the reference solution $u_{C_{REF}}[k]$ was obtained using DASSL with a very small error tolerance (10^{-9}).

8. The central processing unit (CPU) time was measured as the mean value of 10 simulation runs.

4.1. Buck converter

This SMPS, whose model is described in Section 3.2.1 was simulated with the following set of parameters:

- Input source voltage: $V_s = 24$ V;
- Output capacity: $C = 10^{-4}$ F;
- Inductance: $L = 10^{-4}$ H;
- Load Resistance: $R_L = 10\ \Omega$;
- Switch and diode on-state resistance: $R_{\mathrm{On}} = 10^{-5}\Omega$;
- Switch and diode off-state resistance: $R_{\mathrm{Off}} = 10^{5}\Omega$;
- Switch control signal period: $T = 10^{-4}$ s;
- Switch control signal duty cycle: $DC = 0.5$.

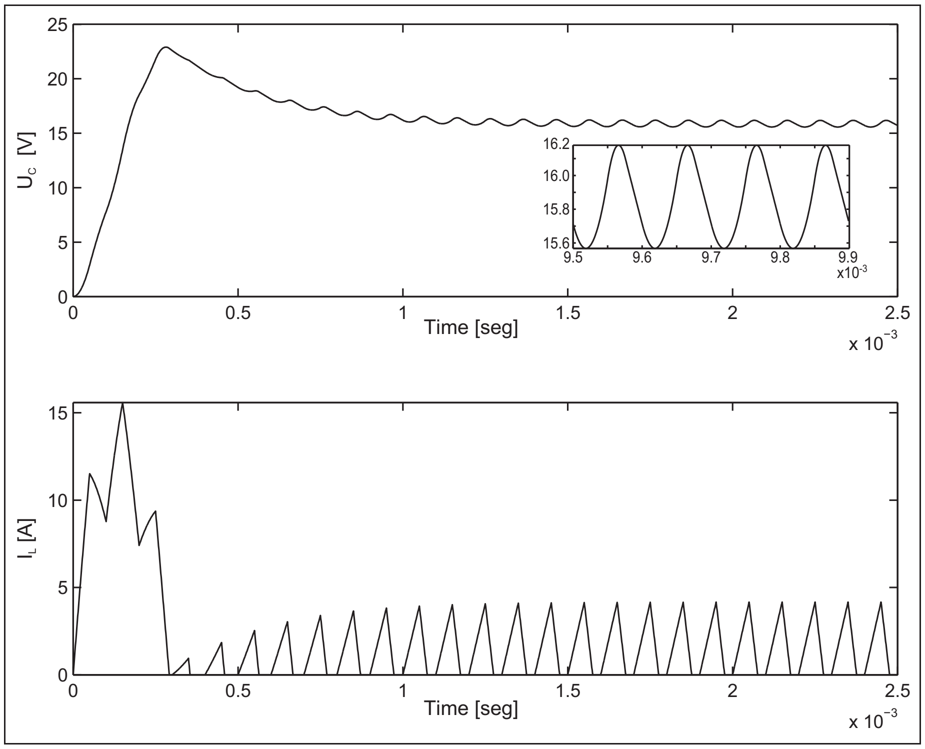

Figure 10. Transient trajectories for the buck converter

Table 1. Buck converter results.

Integration method		Relative error	Jacobian Eval.	Function f_i evaluations	CPU [mseg]
DASSL	err.tol= $1 \cdot 10^{-3}$	$2.28 \cdot 10^{-3}$	3079	26,670	6.58589
	err.tol.= $1 \cdot 10^{-5}$	$9.63 \cdot 10^{-6}$	4474	44,772	11.6278
LIQSS2	$\Delta Q_i = 1 \cdot 10^{-3}$	$1.31 \cdot 10^{-3}$	–	13,286	2.26316
	$\Delta Q_i = 1 \cdot 10^{-5}$	$1.06 \cdot 10^{-5}$	–	117,198	11.3644
LIQSS3	$\Delta Q_i = 1 \cdot 10^{-3}$	$1.09 \cdot 10^{-3}$	–	11,355	3.43807
	$\Delta Q_i = 1 \cdot 10^{-5}$	$1.04 \cdot 10^{-5}$	–	35,283	11.2723

The transient part of the results is shown in Figure 10. As expected for this topology, the output voltage $u_C(t)$ has a mean value lower than the input voltage V_s and it exhibits a small ripple at the switching frequency. The discontinuous behavior of this SMPS can be clearly observed in the current trajectory $i_L(t)$.

Table 1 compares the CPU time, the number of evaluations, and the errors obtained with the different solvers.

It can be seen that all of the solvers meet the error tolerance requirements. Regarding simulation time, when setting a small relative error tolerance (10^{-5}), the three algorithms require similar CPU times even when the number of function evaluations performed by LIQSS2 was approximately three times that corresponding to LIQSS3 and DASSL. This fact can be understood by observing that discontinuity detection in LIQSS2 is cheaper than in LIQSS3 and DASSL (to detect a discontinuity, LIQSS2 only solves a scalar linear equation) and that LIQSS2 does not require Jacobian computations. Also, LIQSS2 has continuous steps that are internally cheaper than those of LIQSS3 and DASSL.

For a larger tolerance (10^{-3}), which is the usual choice for these types of circuits, the simulation using LIQSS2 was 3 times faster than using DASSL and 1.5 times faster than using LIQSS3.

This fact is not surprising since lower order methods are usually more efficient for simulating systems under low accuracy requirements.

4.2. Boost converter

For the boost converter circuit, whose model was presented in Section 3.2.2, we used the same set of parameters as for

Table 2. Boost converter results.

Integration method		Relative error	Jacobian eval.	Function f_i evaluations	CPU [mseg]
DASSL	err.tol= $1 \cdot 10^{-3}$	$1.30 \cdot 10^{-3}$	2215	18,778	4.94262
	err.tol.= $1 \cdot 10^{-5}$	$5.34 \cdot 10^{-5}$	3192	28,834	7.2436
LIQSS2	$\Delta Q_i = 1 \cdot 10^{-3}$	$1.52 \cdot 10^{-3}$	–	10,476	1.54468
	$\Delta Q_i = 1 \cdot 10^{-5}$	$1.96 \cdot 10^{-5}$	–	70,628	8.25393
LIQSS3	$\Delta Q_i = 1 \cdot 10^{-3}$	$1.11 \cdot 10^{-3}$	–	9648	4.36562
	$\Delta Q_i = 1 \cdot 10^{-5}$	$1.26 \cdot 10^{-5}$	–	21,420	8.18659

Table 3. Buck–boost converter results.

Integration method		Relative error	Jacobian eval.	Function f_i evaluations	CPU [mseg]
DASSL	err.tol= $1 \cdot 10^{-3}$	$5.12 \cdot 10^{-3}$	3689	29,240	6.88186
	err.tol.= $1 \cdot 10^{-5}$	$3.04 \cdot 10^{-4}$	5138	46,532	11.8803
LIQSS2	$\Delta Q_i = 1 \cdot 10^{-3}$	$1.39 \cdot 10^{-3}$	–	14,632	2.74414
	$\Delta Q_i = 1 \cdot 10^{-5}$	$1.47 \cdot 10^{-5}$	–	84,476	10.1215
LIQSS3	$\Delta Q_i = 1 \cdot 10^{-3}$	$5.23 \cdot 10^{-4}$	–	12,618	5.28576
	$\Delta Q_i = 1 \cdot 10^{-5}$	$1.34 \cdot 10^{-5}$	–	27,912	8.23346

Table 4. Cuk converter results.

Integration method		Relative error	Jacobian eval.	Function f_i evaluations	CPU [mseg]
DASSL	err.tol= $1 \cdot 10^{-3}$	$1.75 \cdot 10^{-2}$	2858	77,016	10.7689
	err.tol.= $1 \cdot 10^{-5}$	$1.97 \cdot 10^{-4}$	4270	123,200	17.3262
LIQSS2	$\Delta Q_i = 1 \cdot 10^{-3}$	$7.40 \cdot 10^{-3}$	–	73,082	9.6474
	$\Delta Q_i = 1 \cdot 10^{-5}$	$8.71 \cdot 10^{-5}$	–	448,310	30.3638
LIQSS3	$\Delta Q_i = 1 \cdot 10^{-3}$	$6.07 \cdot 10^{-3}$	–	53,547	14.0629
	$\Delta Q_i = 1 \cdot 10^{-5}$	$5.02 \cdot 10^{-5}$	–	97,509	23.9215

the buck converter. Table 2 compares the performance exhibited by the different solvers.

The results are very similar to those of the buck converter and the same explanations can be applied. However, for a tolerance of 10^{-5}, DASSL exhibits a larger and more sensible error than that of the LIQSS methods.

In the LIQSS methods discontinuities are exactly detected, while in DASSL they can have certain error due to the iteration process which increases the global simulation error.

4.3. Buck–boost converter

For the buck–boost converter circuit, described in Section 3.2.3, we used the same set of parameters as for the buck converter except for the duty cycle, which was $DC = 0.25$. The performance comparison for the different solvers is reported in Table 3.

Regarding CPU times and function evaluations, the results are similar to those of the buck and boost converters. However, DASSL had much larger errors than were seen for boost converter. The relative error is between 5 and 30 times larger than the tolerance, while LIQSS methods meet the error requirements as expected.

4.4. Cuk converter

For the Cuk converter, using the model described in Section 3.2.4 with the change of variables described in Section 3.4.4, we repeated the set of parameters used for the buck–boost converter, taking also $C_1 = C_2 = 10^{-4}$ F and $L_1 = L_2 = 10^{-4}$ H.

Table 4 shows the performance comparison for the different algorithms.

Regarding simulation times, for a small error tolerance (10^{-5}) DASSL is now faster than both LIQSS methods, while for the larger tolerance (10^{-3}) the CPU times are similar for the three solvers. However, DASSL errors are almost 20 times larger than the error tolerance, while LIQSS methods are clearly more accurate.

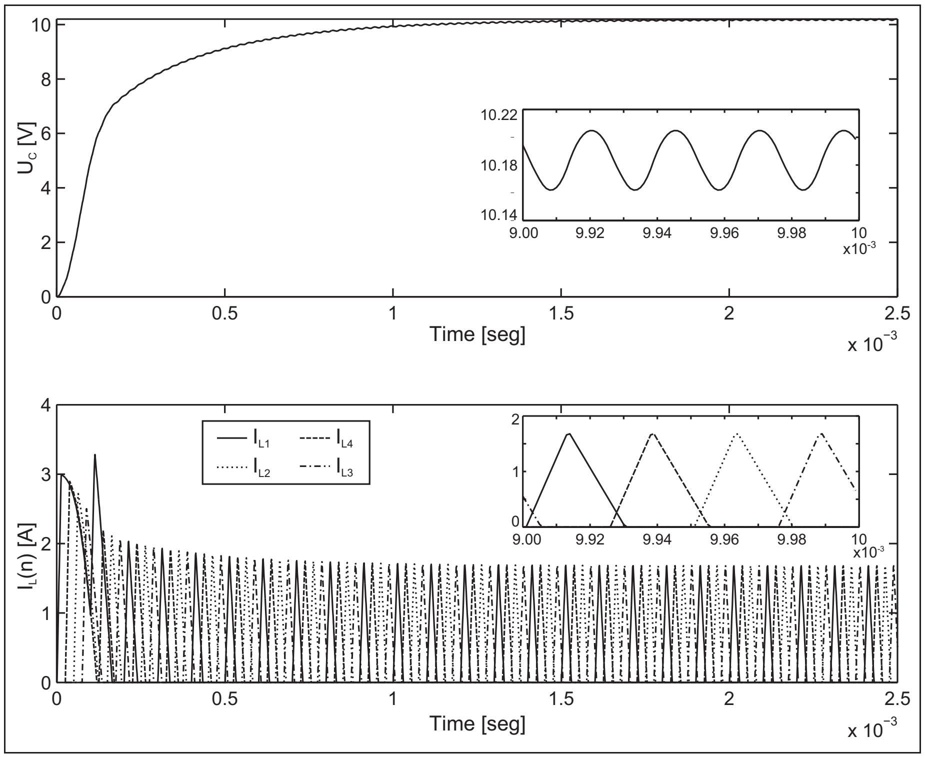

Figure 11. Four-stage interleaved buck converter trajectories.

Table 5. Four-stage interleaved buck converter results.

Integration method		Relative error	Jacobian eval.	Function f_i evaluations	CPU [mseg]
DASSL	err.tol= $1 \cdot 10^{-3}$	$2.50 \cdot 10^{-2}$	12,538	435,224	29.7604
	err.tol.= $1 \cdot 10^{-5}$	$1.40 \cdot 10^{-2}$	16,433	620,754	42.1447
LIQSS2	$\Delta Q_i = 1 \cdot 10^{-3}$	$1.26 \cdot 10^{-3}$	–	61,870	8.82444
	$\Delta Q_i = 1 \cdot 10^{-5}$	$1.62 \cdot 10^{-5}$	–	463,170	35.0696
LIQSS3	$\Delta Q_i = 1 \cdot 10^{-3}$	$8.04 \cdot 10^{-4}$	–	67,425	17.0736
	$\Delta Q_i = 1 \cdot 10^{-5}$	$9.89 \cdot 10^{-6}$	–	122,958	25.9182

4.5. Interleaved buck converter

For this circuit, described in Section 3.3, we used the same set of parameters as for the buck converter, taking also $L_1 = L_2 = \cdots = L_N = 10^{-4}$ H.

Figure 11 shows the output voltage $u_C(t)$ and the inductance currents $i_{L_k}(t)$ $(k = 1, \ldots, 4)$ for a four-stage interleaved buck model. Comparing these trajectories with those of the buck converter in Figure 10, we see that even when the control signal of both models was the same, the ripple amplitude at the output voltage is an appropriate amount smaller in the interleaved model. The current trajectories show the *interleaved* behavior of this circuit.

The performance comparison for the different solvers is reported in Table 5.

LIQSS methods were faster than DASSL in all cases, showing even more advantages than those observed in the buck, boost and buck–boost converters.

As always, LIQSS methods meet the error tolerance requirement in all cases. However, now DASSL exhibits unacceptable errors. They are 25 times larger than the tolerance of 10^{-3} and 14,000 times larger than the tolerance of 10^{-5}. Thus, the last results are in fact invalid for comparison purposes.

The reason for these large errors is that each switch is in *ON* state for a very short time. Thus, a small error in the discontinuity detection may result in a large error on the output voltage.

This model is also sparse as it can be observed in its Jacobian matrix (equation (21)). Thus, LIQSS methods

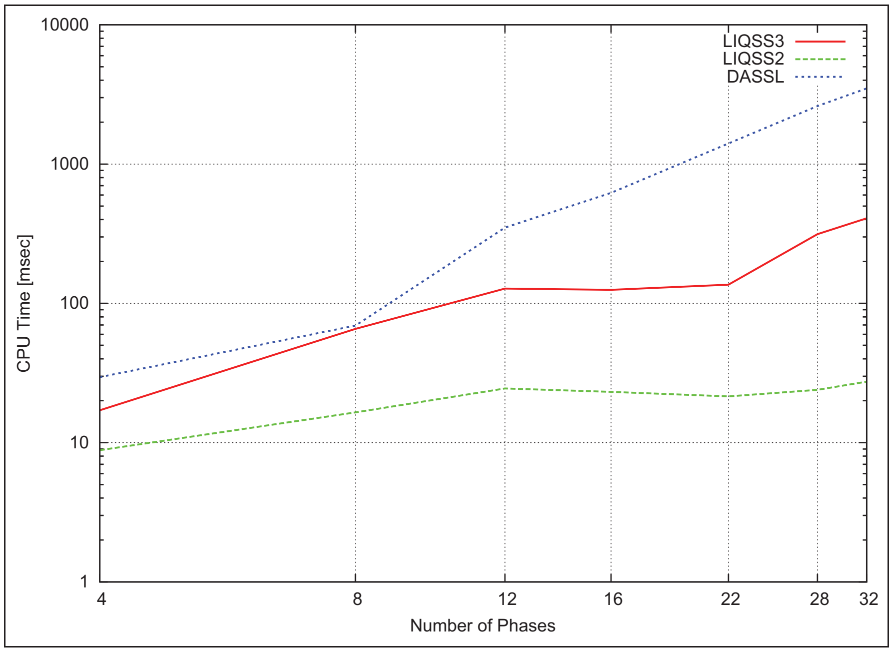

Figure 12. Interleaved buck converter: CPU time versus number of stages (err= $1 \cdot 10^{-3}$).

have the additional advantage of efficient sparsity exploitation which is reflected in a sensibly smaller number of function evaluations, with respect to DASSL.

In order to verify this fact, we also simulated the model varying the size from 4 to 32 stages. In each of these experiments, we set the tolerance of each solver so that the measured error results would be the same. That way, we compare the CPU time required by each solver to simulate the system obtaining identical errors.

The CPU time taken by each solver to simulate a N-stage interleaved buck converter is depicted in Figure 12 (for an error of 10^{-3}) and Figure 13 (for an error of 10^{-5}).

Figure 12 shows that, for simulating the system with a relative error of 10^{-3}, LIQSS2 shows the best performance, followed by LIQSS3, and then DASSL. LIQSS2 is 3 times faster than DASSL for 4 stages and almost 200 times faster than DASSL for 32 stages.

The rapid growth of the CPU time in DASSL can be easily explained. First, the rate of occurrence of discontinuities grows linearly with the number of stages and thus, the maximum step size is reduced accordingly. Secondly, the dimension of the ODE grows linearly with the number of stages and thus each full function evaluation performed by DASSL requires more calculations. Consequently, with smaller steps and a larger number of computations per step, the computational cost grows quadratically (approximately) with the number of stages.

However, in LIQSS methods, each step or discontinuity only provokes local calculations resulting in an almost linear growth of the computational cost with respect to the number of stages.

For the error of 10^{-5} the results are similar, except that now LIQSS3 is faster than LIQSS2 when there are few stages. For the accuracy settings, LIQSS3 can perform larger steps than LIQSS2 and it sensibly reduces the number of function evaluations. However, when the number of stages grow, discontinuities are so frequent that those larger steps are no longer possible, and thus LIQSS2 outperforms LIQSS3.

Anyway, both LIQSS methods are significantly faster than DASSL for a large number of stages.

5. Conclusion

In this article we analyzed the performance of the LIQSS algorithms in the simulation of SMPSs comparing results with those obtained by the classic solver DASSL.

From the analysis performed we conclude that the second order accurate LIQSS2 method results are about 3 times faster than DASSL for a *standard* relative error tolerance of 10^{-3} in single stage circuits. For obtaining more accurate results (relative error tolerance of 10^{-5}), LIQSS2, LIQSS3, and DASSL show similar CPU times. However, in all cases LIQSS methods meet the tolerance settings while DASSL errors may have results up to 20 times larger. Thus, LIQSS results are not only faster but also more robust.

The efficient and exact discontinuity detection and handling, and the fact that LIQSS methods do not need to compute and invert Jacobian matrices to integrate stiff systems, explains these advantages.

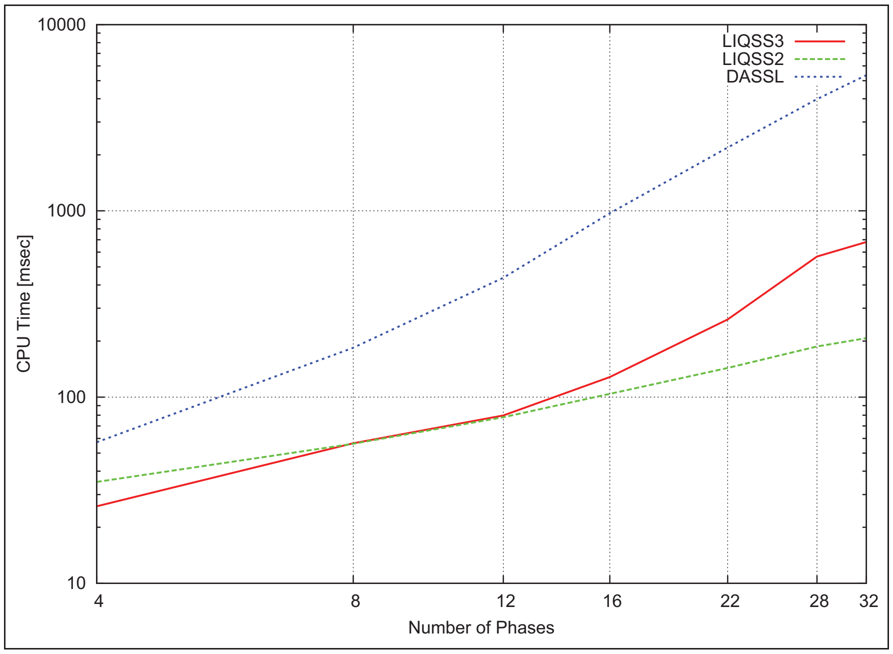

Figure 13. Interleaved buck converter: CPU time versus number of stages (err= $1 \cdot 10^{-5}$).

The analysis of the interleaved buck converter allows us to conclude that both advantages (speed and error) become even more noticeable as the size of the circuit grows. On a 32-stage interleaved converter, LIQSS2 is about 200 times faster than DASSL for obtaining results with similar accuracy.

Here, the intrinsic efficient sparsity exploitation of QSS methods provides an additional advantage to those mentioned for the single stage circuits.

In spite of these advantages, we also observed some drawbacks in the LIQSS methods. The most important limitation is that LIQSS requires that stiffness is due to the presence of large values on the Jacobian main diagonal (without large entries at both sides of the main diagonal). This problem appeared in the Cuk circuit but it was solved by introducing a simple change of variables.

Another limitation is related to the accuracy order. So far, LIQSS methods were implemented up to the third order. Thus, when the error tolerance is too small, the methods require too many steps. However, in the simulation of circuits where the parameter uncertainties are usually large, asking for a relative tolerance lower than 10^{-3} does not make much sense.

Regarding future lines of research, we are currently working on the following issues:

- analyzing the performance under different modeling hypotheses (ideal diodes and switches, or even more realistic models with presence of parasitic inductances and capacitances);
- analyzing the behavior of the SMPSs in a closed loop and with realistic loads;
- automatize the variable change procedure to obtain a system structure adequate for LIQSS;
- creating tools to automatically translate circuit topologies into the set of equations in μ-Modelica required by the QSS stand-alone solver so that the LIQSS algorithms can be easily available for end-users of circuit simulation tools.

Funding

This work was partially funded by FONCYT-ANPCYT (grant PICT 2012 number 0077) and CONICET (grant PIP 2012–2014 number 0216).

References

1. Kofman E and Junco S. Quantized state systems. A DEVS approach for continuous system simulation. *Trans SCS* 2001; 18(3): 123–132.
2. Kofman E. A second order approximation for DEVS simulation of continuous systems. *Simulation* 2002; 78(2): 76–89.
3. Kofman E. A third order discrete event simulation method for continuous system simulation. *Latin Amer Appl Res* 2006; 36(2): 101–108.
4. Migoni G, Kofman E and Cellier F. Quantization-based new integration methods for stiff ODEs. *Simulation* 2012; 88(4): 387–407.
5. Migoni G, Bortolotto M, Kofman E, et al. Linearly implicit quantization-based integration methods for stiff ordinary differential equations. *Sim Modell Practice Theory* 2013; 35: 118–136.

6. Cellier F and Kofman E. *Continuous System Simulation*. New York: Springer, 2006.
7. Kofman E. Discrete event simulation of hybrid systems. *SIAM J Scientific Comput* 2004; 25(5): 1771–1797.
8. Grinblat G, Ahumada H and Kofman E. Quantized state simulation of spiking neural networks. *Simulation* 2012; 88(3): 299–313.
9. Muzy A, Jammalamadaka R, Zeigler BP, et al. The activity-tracking paradigm in discrete-event modeling and simulation: The case of spatially continuous distributed systems. *Simulation* 2011; 87(5): 449–464.
10. Fernández J and Kofman E. A stand-alone quantized state system solver for continuous system simulation. *Simulation* 2014; in press.
11. Nutaro J and Zeigler B. On the stability and performance of discrete event methods for simulating continuous systems. *J Computat Phys* 2007; 227(1): 797–819.
12. Bergero F and Kofman E. PowerDEVS. a tool for hybrid system modeling and real time simulation. *Simulation* 2011; 87(1–2): 113–132.
13. Beltrame T and Cellier F. Quantised state system simulation in Dymola/Modelica using the DEVS formalism. In *Proceedings of the fifth international Modelica conference*, volume 1, Vienna, Austria, pp. 73–82.
14. D'Abreu M and Wainer G. M/CD ++ : Modeling continuous systems using Modelica and DEVS. In *Proceedings of MASCOTS* 2005, Atlanta, GA, pp. 229–236.
15. Quesnel G, Duboz R, Ramat E, et al. VLE: a multimodeling and simulation environment. In *Proceedings of the 2007 Summer Computer Simulation Conference*, San Diego, CA, pp. 367–374.
16. Tiller M. *Introduction to physical modeling with Modelica*. New York: Springer, 2001.
17. Basso C. *Switch-Mode Power Supplies. Spice Simulations and Practical Designs*. New York: McGraw-Hill, 2008.
18. Ang S and Oliva A. *Power-switching converters*. Boca Raton, FL: CRC Press, 2005.
19. Alimeling J and Hammer WP. PLECS- piece-wise linear electrical circuit simulation for Simulink. In *Proceedings of PEDS'99. IEEE international conference on power electronics and drive systems*, volume 1, IEEE, pp. 355–360.
20. Tuinenga PW. *SPICE: a guide to circuit simulation and analysis using PSpice*. Prentice Hall PTR, 1995.
21. Fritzson P. *Principles of Object-Oriented Modeling and Simulation with Modelica 2.1*. New York: Wiley-Interscience, 2004.

Author biographies

Federico Bergero received a computer science degree in 2008 and a PhD degree in informatics in 2012 both from the Universidad Nacional de Rosario, Argentina. He is currently a postdoc student at the French Argentine International Center for Information and Systems Sciences (CIFASIS) and holds a professorial position at the Universidad Nacional de Rosario in the Department of Computer Science, FCEIA–UNR, Rosario, Argentina. His research interests include discrete event systems, real time and parallel simulation, simulation tools, signal processing, and audio filtering methods.

Joaqíen Fernández received his BS degree in computer science in 2012 from the Universidad Nacional de Rosario, Argentina. He is currently a PhD student at the French Argentine International Center for Information and Systems Sciences (CIFASIS). His research interests include hybrid system simulation, real-time and parallel simulation, simulation tools, and modeling languages.

Ernesto Kofman received his BS degree in electronic engineering in 1999 and his PhD degree in automatic control in 2003, all from the National University of Rosario. He is an adjunct professor at the department of control of the National University of Rosario (FCEIA - UNR). He also holds a research position at the National Research Council of Argentina (CIFASIS-CONICET). His research interests include automatic control theory and numerical simulation of hybrid systems. He coauthored a textbook on continuous system simulation in 2006 with Springer, New York

Gustavo Migoni received his BS degree in electronic engineering in 2004 and a PhD degree in automatic control in 2010, both from the National University of Rosario. He is an adjunct professor at the Department of Control of the National University of Rosario (FCEIA–UNR) and holds a research position at the National Research Council of Argentina (CIFASIS-CONICET). His research interests include numerical simulation of hybrid systems and automatic control theory.