Quantization Based Simulation of Switched Mode Power Supplies.

Gustavo Migoni, Federico Bergero, Ernesto Kofman and Joaquín Fernández

June 18, 2014

Abstract

This article studies the performance of Quantized State System (QSS) algorithms in the simulation of Switched Mode Power Supplies (SMPS). Under realistic modeling assumptions, these models result stiff and exhibit frequent discontinuities making them difficult to simulate by classic solvers. However, there are Linearly Implicit QSS (LIQSS) 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 SMPS, and we analyzed the resulting equation structures in order to establish if they can be efficiently simulated by LIQSS algorithms. Then, we compared the simulation performance of LIQSS and the classic solver DASSL, showing that LIQSS results are from 3 to 200 times faster and noticeably more accurate than DASSL.

Keywords: Hybrid Systems Simulation, Quantized State Systems, Switched Mode Power Supplies.

1 Introduction

Switching mode power supplies (SMPS) 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 SMPS 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 if there are any changes in the model and, 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 was developed [14, 11,

13, 16, 15]. These algorithms, that replace the classic time discretization by the quantization of the state variables, have shown some advantages:

- They have strong stability and error bound theoretical properties [14, 11, 6].
- They are very efficient to simulate ODE models with frequent discontinuities [12]. 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.
- They are very efficient in the simulation of large scale sparse discontinuous models [10, 15]. This is due to the fact that QSS methods intrinsically track the system activity [17], performing calculations only where and when changes occur.
- There are Linearly Implicit QSS methods (LIQSS) that can integrate stiff systems with certain structure in a very efficient way, without performing iterations or matrix inversions [15].

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 [15] and interleaved buck converters [8].

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

- An exhaustive analysis of the different SMPS topologies is performed to check if their structure is appropriate to be efficiently integrated with LIQSS algorithms.
- 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.
- A comparative analysis of computational costs and simulation errors using LIQSS and the classic solver DASSL is performed for the different SMPS topologies. We 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.
- 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. Then, Section 3 introduces the topologies and models of SMPSs, analyzing also if they are appropriate for LIQSS simulation. Then, Section 4 presents the simulation results for the different topologies. Finally, Section 5 concludes the article analyzing also future lines of research.

2 Background

This section provides the background required by the rest of the article. It gives a brief description of the problems suffered by classic numerical integration algorithms when dealing with discontinuous systems. Then, it presents the family of QSS methods and the software tools that implement them, and finally it provides 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 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 obviously 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 Quantized State System 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 Quantized State System (QSS1) method [14] analytically solves an approximate ODE called *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:

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

This 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 results that $q_j(t) = x_j(t)$.

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

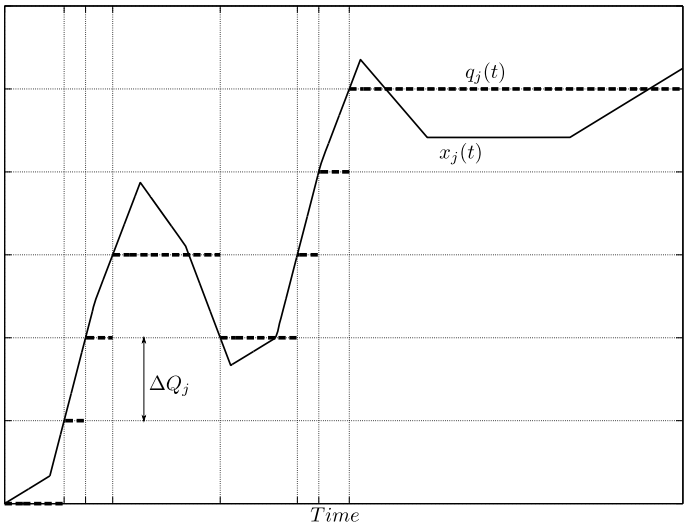


Figure 1: State and Quantized Trajectories in QSS1 Method

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

For $j = 1, \cdots, n$, 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 Eq.(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.

Notice 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, state

events can be straightforwardly detected. 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 that that of a single step. Thus, QSS1 method is very efficient to simulate discontinuous systems [12].

In spite of this advantage and the fact that it has some nice stability and error bound properties [6, 18], QSS1 performs only a first order approximation and it cannot obtain accurate result without significantly increasing the number of steps.

This accuracy limitation was improved with the definition of the second and third-order accurate QSS methods called QSS2 [11] and QSS3 [13], respectively.

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

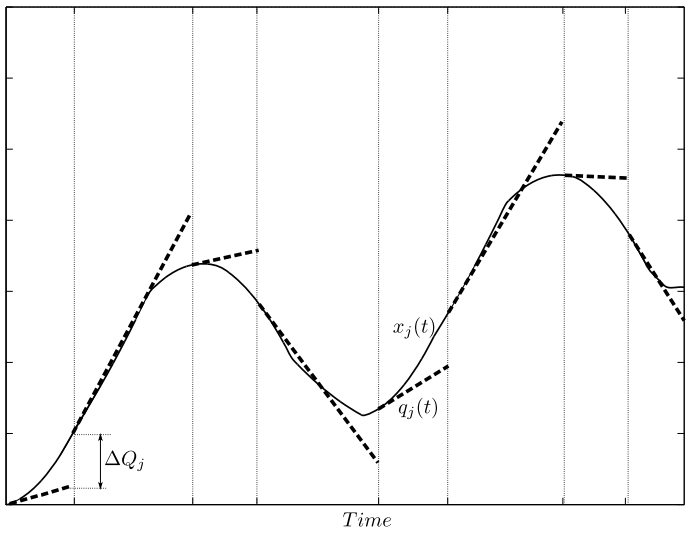


Figure 2: State and Quantized State Trajectories in QSS2

As before, the analytical solution of Eq.(2) for QSS2 and QSS3 can be easily computed and the simulation algorithm is almost identical to that of QSS1. In consequence, QSS2 and QSS3 also share the advantages to simulate discontinuous systems.

In spite of these advantages, QSS1, QSS2 and QSS3 methods are very inefficient to simulate stiff systems. In 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 *Linearly Implicit QSS* (LIQSS) which are appropriate to simulate some stiff systems [15].

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

In spite of being explicit algorithms, LIQSS methods are able to integrate many stiff systems. Anyway, 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 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 [5], a DEVS–based simulation platform specially designed for and adapted to simulating hybrid systems based on QSS methods. In addition, the explicit QSS methods of orders 1 to 3 were also implemented in a DEVS library of Modelica [4] and implementations of the first–order QSS1 method can also be found in CD++ [7] and VLE [19].

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* [8] that improves DEVS–based simulation times in more than one order the magnitude.

The stand–alone QSS solver requires that the models are described in a subset of the Modelica modeling language [20], called μ-Modelica [8].

2.4 Switched Mode Power Supplies

Switched Mode Power Supplies [3] are electronic devices that convert the available DC input voltage to 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.

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

- Buck converter: which reduces the input voltage ($V_{\text{out}} < V_{\text{in}}$).
- Boost converter: which increases the input voltage ($V_{\text{out}} > V_{\text{in}}$).

- Buck–Boost converter: which can increase or decrease the input voltage. The Ćuk circuit is a variant of this converter.

The output voltage in these topologies is regulated by controlling the relationship between the ammount 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 Fig.3.

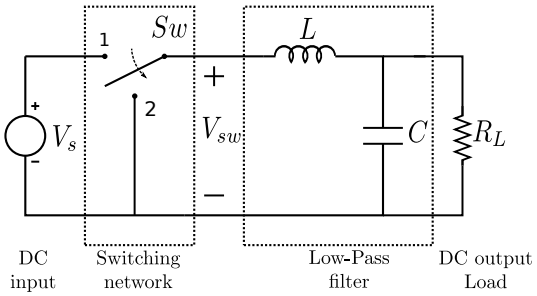


Figure 3: Ideal schematic of a Buck Converter

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 that preserves the mean value of V_{sw} but reduces the high frequency components. That way, the voltage at the load has a continuous value with small undesired high frequency components called *ripple*.

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

In real applications, the ideal switch depicted in Fig.3 is implemented by real components like transistors and diodes. Thus, 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 [2]. 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:

- 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 type of models, the simulation tools must be able to handle variable structure systems.

  Besides these disadvantages, the ideal models of switching components lack of realism and may hide some features of the circuit behavior.

  Anyway, this approach is used by some circuit simulation software tools like PLECS [1], and it has the advantage of avoiding stiffness which allow the usage of fast explicit numerical algorithms.

- 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.

  Due to its simplicity and realism, this is the approach followed by most simulation tools like PSPICE [21] and its variants and those based on Modelica [9].

  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, as it is more 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 it was discussed above, this element

can be modeled as a resistor R_s with a high or low value according to this 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.

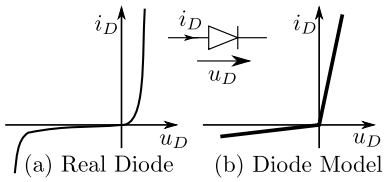
This behavior can be easily represented in terms of the μ–Modelica language. It corresponds to two event handlers that are triggered when the control signal changes.

```
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.



(a) Real Diode (b) Diode Model

Figure 4: Real and approximate diode characteristics

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

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

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.

3.2.1 Buck Converter

As it was described before, the Buck 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.

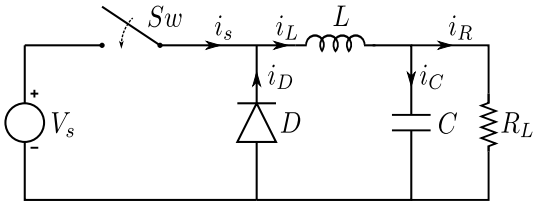


Figure 5: Buck converter circuit

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:

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

where

$$i_D = \frac{i_L R_s - U}{R_s + R_D} \tag{5}$$

Joining Eqs.(4)–(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 before.

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 in order to automatically adjust the 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 circuit, shown in Figure 6, is a switched converter that generates an output voltage higher than the input voltage.

Proceeding in the same way than with the Buck converter, the following state equations can be obtained:

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

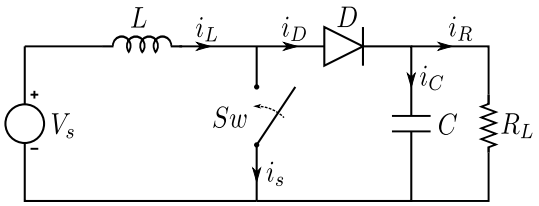


Figure 6: Boost converter circuit

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 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.

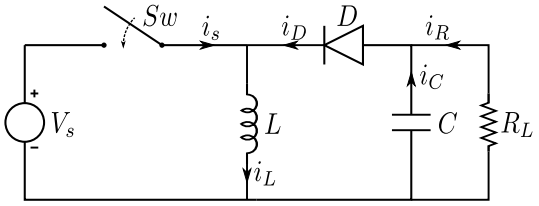


Figure 7: Buck-Boost converter circuit.

Proceedings 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

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

Then, the continuous equations of the model can be written in μ–Modelica as follows:

```
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 Ćuk Converter

The Ćuk 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 Ćuk converter circuit is shown in Figure 8.

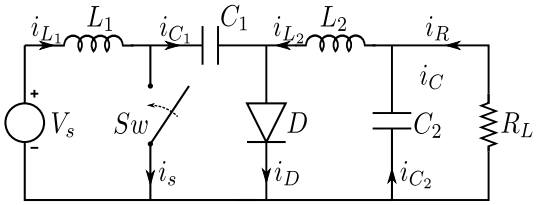


Figure 8: Ćuk converter circuit.

The state equations for this 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:

```
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.

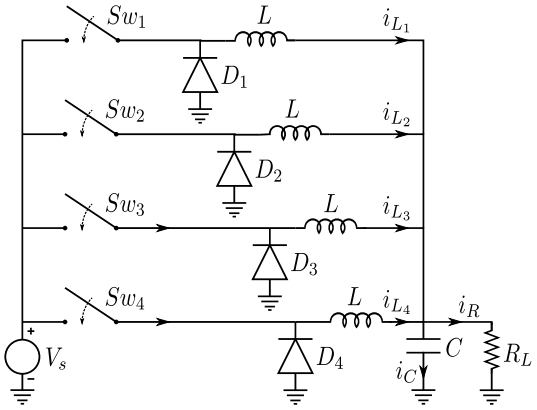


Figure 9: Four–stage Buck interleaved converters

The state equations for a N–stage Buck circuit can be written as follows

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

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];
  end
  der(uC) = (sum(iL)-uC/Rl)/C;
```

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

```
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 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 lineal. 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 independently on 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 can introduce spurious oscillations which add a significant computational load.

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

3.4.1 Buck Converter

Replacing i_D in Eq.(4) by the expression of Eq.(5), the following Jacobian matrix is obtained.

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

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}) they can only cause the appearance of a large value 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 Eqs.(6)–(7), the Jacobian matrix results

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

Here, R_s and R_D appear also 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 is always less than 1. Thus, terms of the order of magnitude of 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 Eqs.(8)–(9) we arrive to the same Jacobian matrix than that of the Boost converter given by Eq.(15). Thus, the Buck–Boost converter can also be efficiently integrated using LIQSS algorithms.

3.4.4 Ćuk Converter

Proceedings as before, from Eqs.(10)–(11) the following Jacobian matrix is obtained:

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

Here, we have several terms outside the main diagonal that depends on the switch resistances. In some of them, these resistance appear within the expression of Eq.(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 $R_s = R_D = R_{\mathrm{OFF}}$ results in a very large value ($R_{\mathrm{OFF}}/2$).

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

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

$$i_{12} \triangleq \frac{L_1 \cdot i_{L_1} - L_2 \cdot i_{L_2}}{L_2}$$

and removing i_{L_1} from Eq.(10)–(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

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

which still has one term that can take values of the order of R_{OFF} outside the main diagonal. However, in presence 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 the following:

```
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 Eqs.(12)–(13), the following Jacobian matrix is obtained for a N–stage Interleaved Buck:

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

It can be seen that the switch and diode resistances R_s and R_D only appear on the main diagonal, and we arrive to the same conclusions than in the Buck converter.

Notice 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:

- We simulated all sources under two different error tolerance settings: rel.tol. = abs.tol = 10^{-3} and rel.tol. = abs.tol = 10^{-5}.
- All the simulations were performed until a final time $t_f = 0.01sec.$.
- The simulations were performed on an Intel i7-3770@3.40GHz PC under Ubuntu OS.
- LIQSS results were obtained with the QSS stand alone solver described above.
- 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.
- 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.
- 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:

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

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

- The 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: $V_s = 24V$,
- 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: $T = 10^{-4}sec.$,
- Switch control signal duty–cycle: $DC = 0.5$.

The transient part of the results is shown in Figure 10. As it is 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 $i_L(t)$.

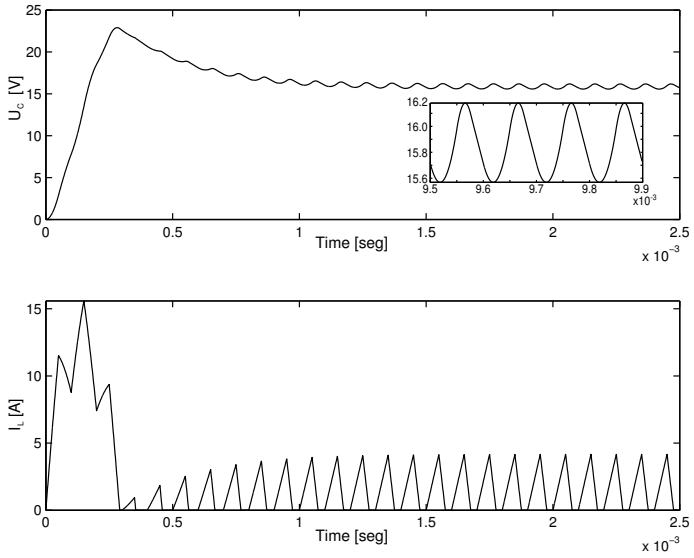


Figure 10: Transient Trajectories for the Buck converter

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

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	26670	6.58589
	err.tol.=$1 \cdot 10^{-5}$	$9.63 \cdot 10^{-}6$	4474	44772	11.6278
LIQSS2	$\Delta Q_i = 1 \cdot 10^{-3}$	$1.31 \cdot 10^{-}3$	–	13286	2.26316
	$\Delta Q_i = 1 \cdot 10^{-5}$	$1.06 \cdot 10^{-}5$	–	117198	11.3644
LIQSS3	$\Delta Q_i = 1 \cdot 10^{-3}$	$1.09 \cdot 10^{-}3$	–	11355	3.43807
	$\Delta Q_i = 1 \cdot 10^{-5}$	$1.04 \cdot 10^{-}5$	–	35283	11.2723

Table 1: Buck converter results comparison.

It can be seen that all the solvers meet the error tolerance requirements. Regarding simulation time, 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 3 times the corresponding to LIQSS3 and DASSL. This fact can be understood taking into account 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 continuous steps are internally cheaper than LIQSS3 and DASSL steps.

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 this circuit, whose model was presented in Section 3.2.2, we used the same set of parameters than for 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 sensibly larger error than the tolerance and than LIQSS methods.

In 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 this circuit, described in Section 3.2.3, we used the same set of parameters than for the Buck converter except for the duty cycle, which is now $DC = 0.25$. 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 the Boost converters. However, when it comes to errors,

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	18778	4.94262
	err.tol.=$1 \cdot 10^{-5}$	$5.34 \cdot 10^{-}5$	3192	28834	7.2436
LIQSS2	$\Delta Q_i = 1 \cdot 10^{-3}$	$1.52 \cdot 10^{-}3$	–	10476	1.54468
	$\Delta Q_i = 1 \cdot 10^{-5}$	$1.96 \cdot 10^{-}5$	–	70628	8.25393
LIQSS3	$\Delta Q_i = 1 \cdot 10^{-3}$	$1.11 \cdot 10^{-}3$	–	9648	4.36562
	$\Delta Q_i = 1 \cdot 10^{-5}$	$1.26 \cdot 10^{-}5$	–	21420	8.18659

Table 2: Boost converter results comparison.

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	29240	6.88186
	err.tol.=$1 \cdot 10^{-5}$	$3.04 \cdot 10^{-}4$	5138	46532	11.8803
LIQSS2	$\Delta Q_i = 1 \cdot 10^{-3}$	$1.39 \cdot 10^{-}3$	–	14632	2.74414
	$\Delta Q_i = 1 \cdot 10^{-5}$	$1.47 \cdot 10^{-}5$	–	84476	10.1215
LIQSS3	$\Delta Q_i = 1 \cdot 10^{-3}$	$5.23 \cdot 10^{-}4$	–	12618	5.28576
	$\Delta Q_i = 1 \cdot 10^{-5}$	$1.34 \cdot 10^{-}5$	–	27912	8.23346

Table 3: Buck-Boost converter results comparison.

now DASSL is even worse than in the Boost case. The relative error is between 5 and 30 times larger than the tolerance while LIQSS methods meet the error requirements as expected.

4.4 Ćuk Converter

For the Ćuk 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 of 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.

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	77016	10.7689
	err.tol.=$1 \cdot 10^{-5}$	$1.97 \cdot 10^{-}4$	4270	123200	17.3262
LIQSS2	$\Delta Q_i = 1 \cdot 10^{-3}$	$7.40 \cdot 10^{-}3$	–	73082	9.6474
	$\Delta Q_i = 1 \cdot 10^{-5}$	$8.71 \cdot 10^{-}5$	–	448310	30.3638
LIQSS3	$\Delta Q_i = 1 \cdot 10^{-3}$	$6.07 \cdot 10^{-}3$	–	53547	14.0629
	$\Delta Q_i = 1 \cdot 10^{-5}$	$5.02 \cdot 10^{-}5$	–	97509	23.9215

Table 4: Cuk converter simulation results comparison.

4.5 Interleaved Buck Converter

For this circuit, described in Section 3.3, we used the same set of parameters than 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 $i_{L_k}(t)$ $(k = 1 \cdots 4)$ for a 4-stage Interleaved Buck model. Comparing these trajectories to those of the Buck converter in Fig.10, it can be seen that, even when the control signal of both models was the same, the ripple amplitude at the output voltage is sensibly smaller in the interleaved model. The current trajectories show the *interleaved* behavior of this circuit.

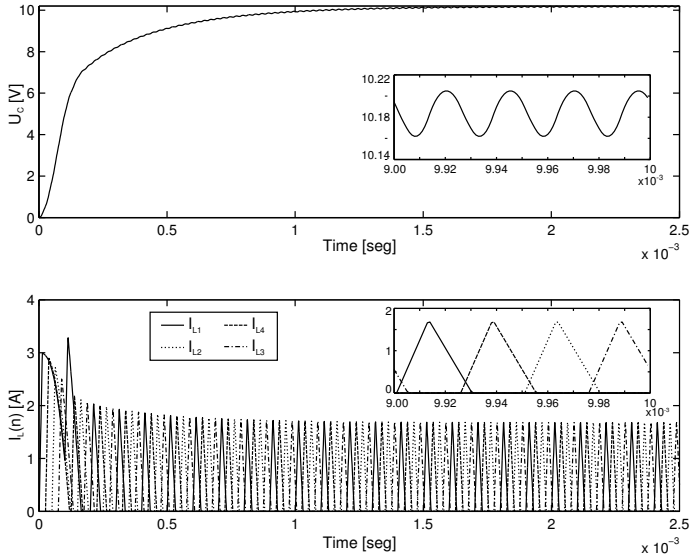


Figure 11: 4–Stage Interleaved Buck Converter Results

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

Now, LIQSS methods were faster than DASSL in all cases, showing even more advantages than those observed in the Buck, Boost and Buck–Boost con-

Integration Method		Relative Error	Jacobian Eval.	Function f_i Evaluations	CPU [mseg]
DASSL	err.tol=$1 \cdot 10^{-3}$	$2.50 \cdot 10^{-}2$	12538	435224	29.7604
	err.tol.=$1 \cdot 10^{-5}$	$1.40 \cdot 10^{-}2$	16433	620754	42.1447
LIQSS2	$\Delta Q_i = 1 \cdot 10^{-3}$	$1.26 \cdot 10^{-}3$	–	61870	8.82444
	$\Delta Q_i = 1 \cdot 10^{-5}$	$1.62 \cdot 10^{-}5$	–	463170	35.0696
LIQSS3	$\Delta Q_i = 1 \cdot 10^{-3}$	$8.04 \cdot 10^{-}4$	–	67425	17.0736
	$\Delta Q_i = 1 \cdot 10^{-5}$	$9.89 \cdot 10^{-}6$	–	122958	25.9182

Table 5: 4–Stage Interleaved Buck converter results comparison.

verters.

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 of Eq.(21). Thus, LIQSS methods have the additional advantage of its 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 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. Firstly, 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 growth 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 about quadratically 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.

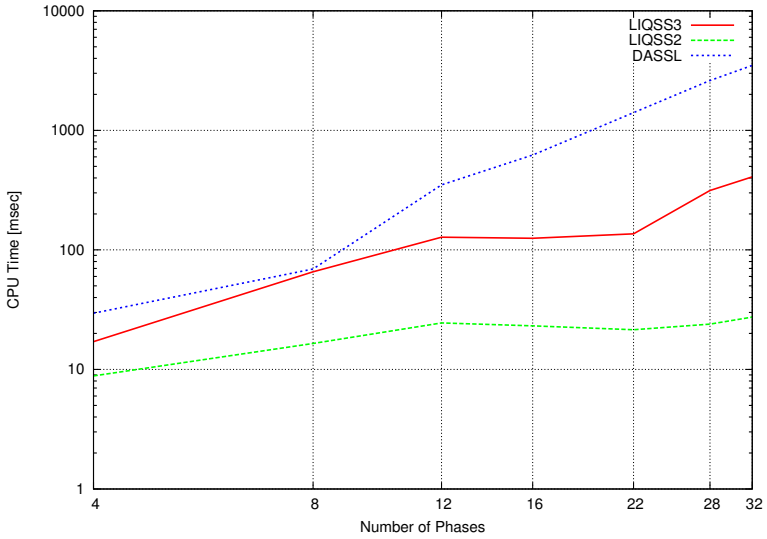


Figure 12: Simulation Time comparison (err=$1 \cdot 10^{-3}$)

For the error of 10^{-5} the results are similar, except that now LIQSS3 is faster than LIQSS2 when there are few stages. For this accuracy settings, LIQSS3 can perform larger steps than LIQSS2 and it sensibly reduces the number of function evaluations. However, when the number of stages grows, 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.

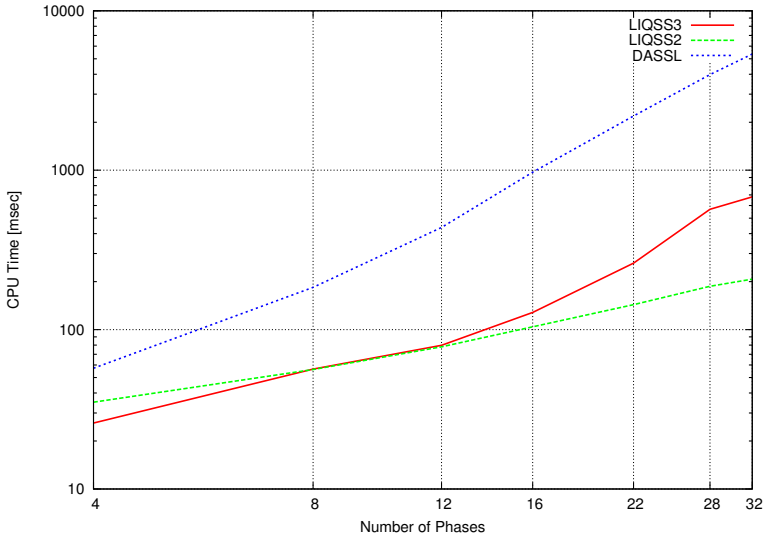


Figure 13: Simulation Time comparison (err=$1 \cdot 10^{-5}$)

5 Conclusions

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

From the analysis performed we conclude that the second order accurate LIQSS2 method results 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 result 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 explain these advantages.

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 LIQSS methods. The most important limitation is that LIQSS require that stiffness is due to the presence of large entries on the main diagonal (without large entries at both sides of the main diagonal). This problem appeared in the Ćuk 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 order 3. 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:

- Analyze the performance under different modeling hypothesis (ideal diodes and switches or even more realistic models with presence of parasitic inductances and capacitances).
- Analyze the behavior of the SMPS in closed loop and with realistic loads.
- Automatize the variable change procedure to obtain a system structure adequate for LIQSS.
- Create 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.

References

[1] JH Alimeling and Wolfgang P Hammer. 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, pages 355–360. IEEE, 1999.

[2] Simon Ang and Alejandro Oliva. *Power-switching converters*. CRC press, 2005.

[3] Christophe Basso. *Switch-Mode Power Supplies. Spice Simulations and Practical Designs*. Mc Graw Hill, 2008.

[4] T. Beltrame and F.E. Cellier. Quantised state system simulation in Dymola/Modelica using the DEVS formalism. In *Proceedings of the Fifth International Modelica Conference*, volume 1, pages 73–82, Vienna, Austria, 2006.

[5] F. Bergero and E. Kofman. PowerDEVS. A Tool for Hybrid System Modeling and Real Time Simulation. *Simulation: Transactions of the Society for Modeling and Simulation International*, 87(1–2):113–132, 2011.

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

[7] M. D'Abreu and G. Wainer. M/CD++: Modeling continuous systems using Modelica and DEVS. In *Proceedings of MASCOTS 2005*, pages 229 – 236, Atlanta, GA, 2005.

[8] Joaquín Fernández and Ernesto Kofman. A Stand-Alone Quantized State System Solver for Continuous System Simulation. *Simulation: Transactions of the Society for Modeling and Simulation International*, 2014. in Press.

[9] Peter Fritzson. *Principles of Object-Oriented Modeling and Simulation with Modelica 2.1.* Wiley-Interscience, New York, 2004.

[10] G. Grinblat, H. Ahumada, and E. Kofman. Quantized State Simulation of Spiking Neural Networks. *Simulation: Transactions of the Society for Modeling and Simulation International*, 88(3):299–313, 2012.

[11] E. Kofman. A Second Order Approximation for DEVS Simulation of Continuous Systems. *Simulation: Transactions of the Society for Modeling and Simulation International*, 78(2):76–89, 2002.

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

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

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

[15] G. Migoni, M. Bortolotto, E. Kofman, and F. Cellier. Linearly Implicit Quantization-Based Integration Methods for Stiff Ordinary Differential Equations. *Simulation Modelling Practice and Theory*, 35:118–136, 2013.

[16] G. Migoni, E. Kofman, and F. Cellier. Quantization-Based New Integration Methods for Stiff ODEs. *Simulation: Transactions of the Society for Modeling and Simulation International*, 88(4):387–407, 2012.

[17] Alexandre Muzy, Rajanikanth Jammalamadaka, Bernard P Zeigler, and James J Nutaro. The activity-tracking paradigm in discrete-event modeling and simulation: The case of spatially continuous distributed systems. *Simulation*, 87(5):449–464, 2011.

[18] J. Nutaro and B. Zeigler. On the stability and performance of discrete event methods for simulating continuous systems. *Journal of Computational Physics*, 227(1):797–819, 2007.

[19] G. Quesnel, R. Duboz, E. Ramat, and M. Traoré. Vle: a multimodeling and simulation environment. In *Proceedings of the 2007 Summer Computer Simulation Conference*, pages 367–374, San Diego, California, 2007.

[20] Michael Tiller. *Introduction to physical modeling with Modelica*. Springer, 2001.

[21] Paul W Tuinenga. *SPICE: a guide to circuit simulation and analysis using PSpice*. Prentice Hall PTR, 1995.