An introduction to Siconos

Vincent Acary, Franck Pérignon

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

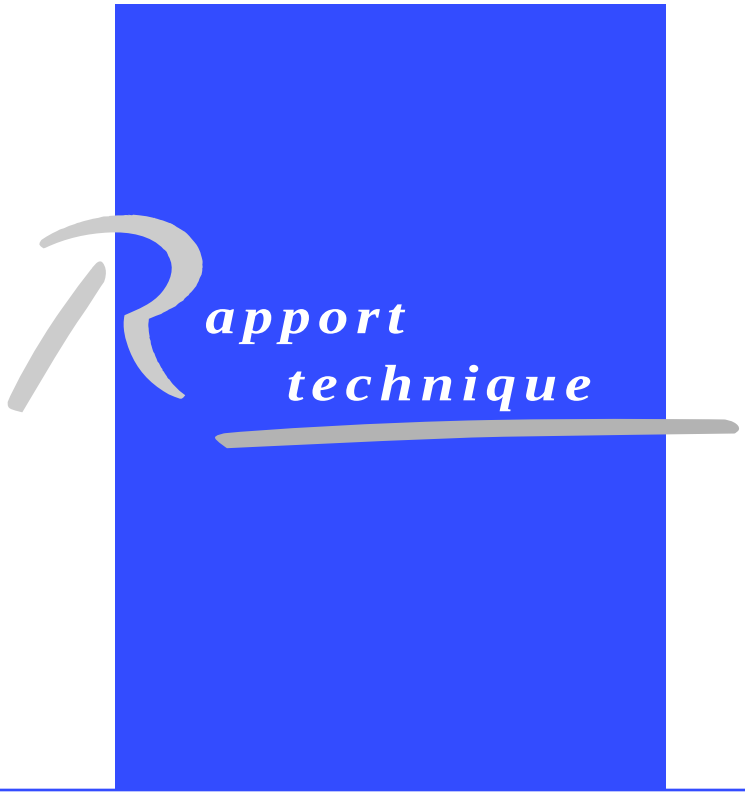
An introduction to Siconos

Vincent Acary — Franck Pérignon

N° 0340

July 2007

Thème NUM



ISSN 0249-0803 ISRN INRIA/RT--0340--FR+ENG

An introduction to Siconos

Vincent Acary, Franck Pérignon

Thème NUM — Systèmes numériques

Projet Bipop

Rapport technique n° 0340 — July 2007 — 45 pages

Abstract: In this document, a brief overview of the Siconos Platform is given. One of the goal is to give a flavor on a simple example of the ability of the platform to model and simulate the so-called non smooth dynamical systems (NSDS). In particular, some examples of Lagrangian mechanical systems with contact and friction or electrical circuits with ideal and piecewise linear components (diodes, MOS transistors, ...) are commented. Finally, the Siconos software is presented, starting from its architecture to a non exhaustive presentation of its components and functionalities. The aim of this document is not to serve as a reference guide but more as a illustrative introduction document to promote the use of the platform.

Key-words: Nonsmooth Dynamical Systems, Linear Complementarity Systems, Lagrangian Mechanical Systems with Contact and Friction, Electrical Circuits with Ideal and Piecewise Linear Components, Siconos, Time Stepping Scheme, Event Driven Scheme, Nonsmooth Law.

Unité de recherche INRIA Rhône-Alpes
655, avenue de l'Europe, 38334 Montbonnot Saint Ismier (France)
Téléphone : +33 4 76 61 52 00 — Télécopie +33 4 76 61 52 52

Une introduction à Siconos

Résumé : Dans cet article on présente sommairement la plate-forme Siconos. Dans un premier temps, la démarche de modélisation et de simulation des systèmes dynamiques dits "non réguliers" (NSDS) est illustrée grâce à un exemple simple. Ensuite, l'architecture du logiciel Siconos est détaillée et complétée par une description non-exhaustive de ses composants et fonctionnalités. Pour terminer, quelques exemples de calcul sont présentés, notamment des systèmes mécaniques lagrangiens avec contacts et frottements, ou encore des circuits électriques constitués de composants idéaux et linéaires par morceaux (diodes, transistors MOS ...). A noter que ce document n'est pas un guide de référence ou d'utilisation de la plate-forme mais simplement une illustration de ses capacités, et est plutôt dédié à la promotion du logiciel Siconos.

Mots-clés : Systèmes Dynamiques Non Réguliers, Systèmes à Complémentarité Linéaire, Systèmes Mécaniques Lagrangiens avec contacts et frottements, Circuits Electriques avec composants linéaires par morceaux, Siconos, Time-Stepping, Schéma à Evènements, Loi Non Régulière.

Contents

I An insight into Siconos 6

1 Step 1. Building a NonSmooth Dynamical System 6

2 Step 2. Simulation Strategy Definition 9

II Overview of NonSmooth Dynamical Systems (NSDS) 11

3 Standard Classes of NSDS 11

3.1 The Example of the Constrained First Order Dynamics 11
3.2 Nonsmooth Lagrangian Dynamical Systems or NonSmooth Multibody Systems. . . 12
3.3 Complementarity Systems. 13
3.4 Other Classes of NSDS 14
3.5 Comparison with the Hybrid Approach 14

4 Simulation of NSDS 15

4.1 The NonSmooth Approach *vs.* the Hybrid Approach 15
4.2 Event–driven and Time–stepping Schemes 15

III Siconos Software 17

5 General Principles of Modeling and Simulation 17

5.1 NSDS Modeling in Siconos Software 17
5.2 Simulation Strategies for the NSDS Behavior 19
5.3 Control Tools 19

6 NSDS Related Components 20

6.1 Dynamical Systems 20
6.2 Relations 21
6.3 Nonsmooth Laws 23

7 Simulation Related Components 23

7.1 Integration of the Dynamics 23
7.2 Formalization and Solving of the Nonsmooth Problems 24

8 Siconos Software Design 25

8.1 Overview 25

8.2 Siconos Kernel Components 26

IV Examples **28**

9 Mechanical examples **28**

9.1 The Bouncing Ball(s) 28

9.2 A Lagrangian Nonlinear System: Simulation of a Robotic Arm 30

9.3 The Woodpecker Toy 32

9.4 The Cam Follower System 32

10 Electrical Examples **36**

10.1 MOS Transistors and Inverters 36

10.2 Power Converters 43

Introduction

SICONOS was a European Project, starting in September 1, 2002, ending in December 31, 2006, gathering scientists from various communities like Mechanics, Applied Mathematics, Systems and Control, and Numerical Analysis. More details are available at `http://siconos.inrialpes.fr`.

The project has resulted in the Siconos Platform, a scientific computing software dedicated to the modeling, simulation, control and analysis of NonSmooth Dynamical Systems (NSDS), mainly developed in the BipOp team (`http://bipop.inrialpes.fr`) at INRIA Rhônes-Alpes in Grenoble and distributed under GPL GNU license.

The software can be downloaded at `http://siconos.gforge.inria.fr/`, where one can also find an installation guide, a tutorial, the full doxygen documentation of the code, support, mailing lists and all that sort of utilities.

Siconos aims at providing a general and common tool for nonsmooth problems in various scientific fields like applied Mathematics, Mechanics, Robotics, Electrical circuits and so on. However, the platform is not supposed to re-implement the existing dedicated tools already used for the modeling of specific systems, but to possibly integrate them. For instance, strong collaborations exist with HuMAns (humanoid motion modeling and control[1]) or LMGC90 (multi-body contact mechanics[2]) software package. Some interfaces with the Scilab/Scicos environment, the Matlab/Simulink environment, the Modelica language and the SPICE software either already exist or are planned.

In the present document, we begin in Part I with the presentation of a characteristic example of the ability of the Siconos Platform: A 4-diode bridge. The goal of this first section is to provide the reader with an insight into the Siconos platform without entering into further details. After this part, some recalls are given in Part II on what can be a NonSmooth Dynamical system (NSDS) and what are the possible strategies to simulate it. In Part III, The general modeling and simulation principles in Siconos are explained. This part ends with a description of the main basic components, which help the user to build a NSDS in Siconos. Finally, in Part IV, Some illustrative examples are presented.

It is noteworthy that this document is nor a user guide neither a tutorial; for a practical use of Siconos please refer to the documents on `http://siconos.gforge.inria.fr/`. The only goal of this document is to give a flavor of the Siconos platform and to promote its use.

[1] `http://bipop.inrialpes.fr/software/humans/index.html`
[2] `http://www.lmgc.univ-montp2.fr/~dubois/LMGC90/`

Part I

An insight into Siconos

The present part is dedicated to a short presentation of the general writing process for a problem treated with Siconos, through a simple example. The point is to introduce the main functionalities, the main steps required to model and simulate the systems behavior, before going more into details in parts II and III, where the NSDS will be described.

The chosen example is a four diodes bridge wave rectifier as shown on Figure 1.

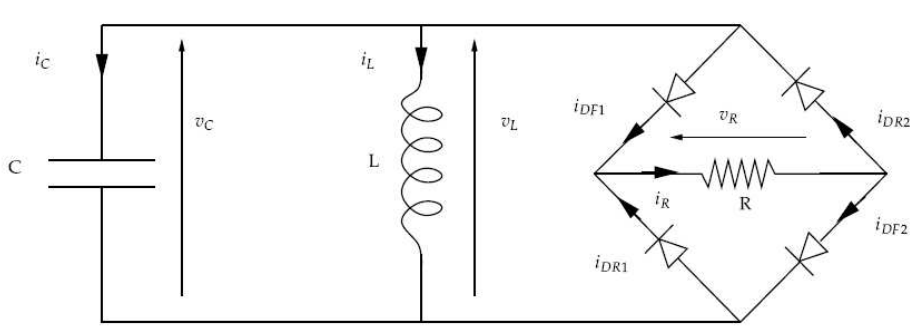


Figure 1: A 4-diodes bridge wave rectifier

A LC oscillator, initialized with a given voltage across the capacitor and a null current through the inductor, provides the energy to a load resistance through a full-wave rectifier consisting of a 4 ideal diodes bridge. Both waves of the oscillating voltage across the LC are provided to the resistor with current flowing always in the same direction. The energy is dissipated into the resistor and results in a damped oscillation.

One of the ways to define a problem with Siconos consists in writing a C++ file. In the following, for the diode-bridge example, only snippets of the C++ commands will be given, just to enlighten the main steps. It is noteworthy that one can also use an XML description as shown in the BouncingBall example 9.1, or the Python interface.

1 Step 1. Building a NonSmooth Dynamical System

In the present case, the oscillator is a time-invariant linear dynamical system, and using the Kirchhoff current and voltage laws and branch constitutive equations, its dynamics is written as (see Figure 1 for notation)

$$\begin{bmatrix} \dot{v}_L \\ \dot{i}_L \end{bmatrix} = \begin{bmatrix} 0 & -\frac{1}{C} \\ \frac{1}{L} & 0 \end{bmatrix} . \begin{bmatrix} v_L \\ i_L \end{bmatrix} + \begin{bmatrix} 0 & 0 & -\frac{1}{C} & \frac{1}{C} \\ 0 & 0 & 0 & 0 \end{bmatrix} . \begin{bmatrix} -v_{DR1} \\ -v_{DF2} \\ i_{DF1} \\ i_{DR2} \end{bmatrix} . \tag{1}$$

If we denote

$$x = \begin{bmatrix} \dot{v}_L \\ \dot{i}_L \end{bmatrix}, \quad \lambda = \begin{bmatrix} -v_{DR1} \\ -v_{DF2} \\ i_{DF1} \\ i_{DR2} \end{bmatrix}, \quad A = \begin{bmatrix} 0 & \frac{-1}{C} \\ \frac{1}{L} & 0 \end{bmatrix}, \quad r = \begin{bmatrix} 0 & 0 & \frac{-1}{C} & \frac{1}{C} \\ 0 & 0 & 0 & 0 \end{bmatrix} . \lambda \tag{2}$$

the dynamical system (1) results in:

$$\dot{x} = A.x + r \tag{3}$$

which fits with the formalism of first order linear dynamical systems proposed in (24).

The first step of any Siconos problem is to define and build some `DynamicalSystems`. The corresponding command lines to build a `FirstOrderLinearTIDS` object are:

```
// User-defined parameters
unsigned int ndof = 2;  // number of degrees of freedom of your system
double Lvalue = 1e-2;   // inductance
double Cvalue = 1e-6;   // capacitance
double Rvalue = 1e3;    // resistance
double Vinit = 10.0;    // initial voltage
// DynamicalSystem(s)
SimpleMatrix A(ndof,ndof); // All components of A are automatically set to 0.
A(0,1) = -1.0/Cvalue;
A(1,0) = 1.0/Lvalue;
// initial conditions vector
SimpleVector x0(ndof);
x0(0) = Vinit;
// Build a First Order Linear and Time Invariant Dynamical System
//           using A matrix and x0 as initial state.
FirstOrderLinearTIDS * oscillator = new FirstOrderLinearTIDS(1,x0,A);
```

The suffix `DS` to the name of a class such as the `FirstOrderLinearTIDS` object means that this class inherits from the general class of `DynamicalSystem`.

Thereafter, it is necessary to define the way the previous defined dynamical systems will interact together. This is the role of the `Interaction` object composed of a `Relation` object, something like algebraic constraints, and of a `NonSmoothLaw` object.

The linear relations between voltage and current inside the circuit are given by

$$\begin{bmatrix} i_{DR1} \\ i_{DF2} \\ -v_{DF1} \\ -v_{DR2} \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ -1 & 0 \\ 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} v_L \\ i_L \end{bmatrix} + \begin{bmatrix} \frac{1}{R} & \frac{1}{R} & -1 & 0 \\ \frac{1}{R} & \frac{1}{R} & 0 & -1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} -v_{DR1} \\ -v_{DF2} \\ i_{DF1} \\ i_{DR2} \end{bmatrix} \tag{4}$$

which can be stated as in (34) by the linear equation

$$y = Cx + D\lambda, \tag{5}$$

with:

$$y = \begin{bmatrix} i_{DR1} \\ i_{DF2} \\ -v_{DF1} \\ -v_{DR2} \end{bmatrix}, C = \begin{bmatrix} \frac{1}{R} & \frac{1}{R} & -1 & 0 \\ \frac{1}{R} & \frac{1}{R} & 0 & -1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}, D = \begin{bmatrix} -v_{DR1} \\ -v_{DF2} \\ i_{DF1} \\ i_{DR2} \end{bmatrix} \tag{6}$$

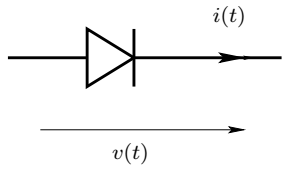
Completed with the relation between r and λ (see (2)) it results in a linear equation as in (34)

$$r = B\lambda. \tag{7}$$

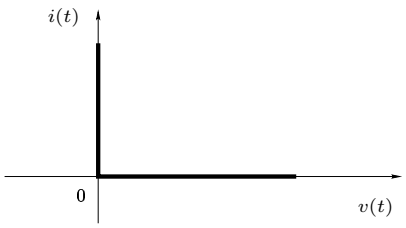
This corresponds to a Siconos `FirstOrderLinearTIR` object i.e. a linear and time invariant coefficients relation, (34). The corresponding code is as follows:

```
// -- Interaction --
// - Relations -
unsigned int ninter = 4; // dimension of your Interaction = size of y and lambda vectors
SimpleMatrix B(ndof,ninter);
B(0,2) = -1.0/Cvalue ;
B(0,3) = 1.0/Cvalue;
SimpleMatrix C(ninter,ndof);
C(2,0) = -1.0;
C(3,0) = 1.0;
// the Relation:
FirstOrderLinearTIR * myRelation = new FirstOrderLinearTIR(C,B);
SimpleMatrix D(ninter,ninter); D(0,0) = 1.0/Rvalue;
D(0,1) = 1.0/Rvalue; D(0,2) = -1.0; D(1,0) = 1.0/Rvalue;
D(1,1) = 1.0/Rvalue; D(1,3) = -1.0; (2,0) = 1.0; D(3,1) = 1.0;
myRelation->setD(D);
```

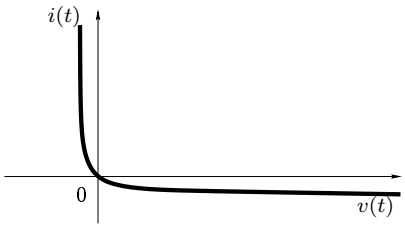
To complete the `Interaction` object, a nonsmooth law is needed to define what the behavior will be when a nonsmooth event occurs. On Figure 2 below, the left-hand sketch displays the ideal diode characteristic and the right-hand sketch displays the usual exponential characteristic as stated by Shockley's law. Thus the behavior of each diode of the bridge, supposed to be ideal,



(a) The diode component.



(b) Characteristics of an ideal diode. A complementarity condition



(c) The graph of the Shockley's law.

Figure 2: Diodes characteristics

can be described with a complementarity condition between current and reverse voltage (variables (y, λ)). Depending on the diode position in the bridge, y stands for the reverse voltage across the diode or for the diode current. Then, the complementarity conditions, results of the ideal diodes characteristics, are given by:

$$\begin{array}{l} 0 \leqslant -v_{DR1} \perp i_{DR1} \geqslant 0 \\ 0 \leqslant -v_{DF2} \perp i_{DF2} \geqslant 0 \\ 0 \leqslant i_{DF1} \perp -v_{DF1} \geqslant 0 \\ 0 \leqslant i_{DR2} \perp -v_{DR2} \geqslant 0 \end{array} \quad \Leftrightarrow \quad 0 \leqslant y \perp \lambda \geqslant 0 \tag{8}$$

which corresponds to a `ComplementarityConditionNSL` object which is an inherited class form the `NonSmoothLaw` class. The Siconos code is as follows.

```
// NonSmoothLaw definition
unsigned int nslawSize = 4;
NonSmoothLaw * myNslaw = new ComplementarityConditionNSL(nslawSize);
```

The `Interaction` is built using the concerned `DynamicalSystem`, the `Relation` and the `NonSmoothLaw` defined above:

```
// A name and a id-number for the Interaction
string nameInter = "InterDiodeBridge";
unsigned int numInter = 1;
unsigned int ninter = 4; // ninter is the size of y
Interaction* myInteraction = new Interaction(nameInter,allDS,numInter,ninter, myNslaw, myRelation);
```

When the `DynamicalSystem` and `Interaction` have been clearly defined, they are gathered into a NSDS,

```
// NonSmoothDynamicalSystem construction
NonSmoothDynamicalSystem* myNSDS = new NonSmoothDynamicalSystem(oscillator,myInteraction);
```

Finally the NSDS is inserted into a `Model`, an object that will link the NSDS to the strategy of simulation. It also defines the time boundaries of the simulation.

```
// Model construction
double t0 = 0; // Initial time
double T = 10; // Total simulation time
Model * DiodeBridge = new Model(t0,T);
// The pre-built NSDS is linked to the DiodeBridge Model.
DiodeBridge->setNonSmoothDynamicalSystemPtr(myNSDS);
```

From this point, the diodes bridge system is completely defined by the `NonSmoothDynamicalSystem` object named `myNSDS` and handled by the `Model` object `DiodeBridge`. In the next section, a strategy of simulation will be defined and applied to this model.

2 Step 2. Simulation Strategy Definition

It is now necessary to define the way the dynamical behavior of the `NonSmoothDynamicalSystem` will be computed. This is the role of `Simulation` class. In Siconos, two different strategies of simulation are available at this time: the time-stepping schemes or the event-driven algorithms. They will be detailed in Section 4.

To be complete, a `Simulation` object requires:

- a discretization of the considered time interval of study,
- a time–integration method for the dynamics,
- a way to formalize and solve the possibly nonsmooth problems.

For the diode bridge example, the Moreau's time–stepping scheme is used, where the integration of the equations over the time steps is based on a θ–method. The nonsmooth problem is written as a Linear Complementarity Problem (LCP) and solved thanks to a projected Gauss-Seidel algorithm. The resulting code in Siconos is:

```
double h =  1.0e-6;  // Time step
// The time discretisation, linked to the Model.
TimeDiscretisation * td = new TimeDiscretisation(h, DiodeBridge);
Simulation * s = new TimeStepping(td);
// Moreau Integrator for the dynamics:
double theta = 0.5;
Moreau* myIntegrator = new Moreau(oscillator,theta,s);
// One Step nonsmooth problem:
string solverName = "PGS"; // nonsmooth problem solver name.
OneStepNSProblem* myLCP = new LCP(s, "LCP", solverName, 101, 0.0001, "max", 0.6 );
```

Note that the `Simulation` is connected to the `Model` thanks to the `TimeDiscretisation`.

The last step is the simulation process with first the initialization and then the time-loop:

```
// Simulation process
s->initialize();
s->run()
```

The final results of the previous simulation are presented on Figure 3.

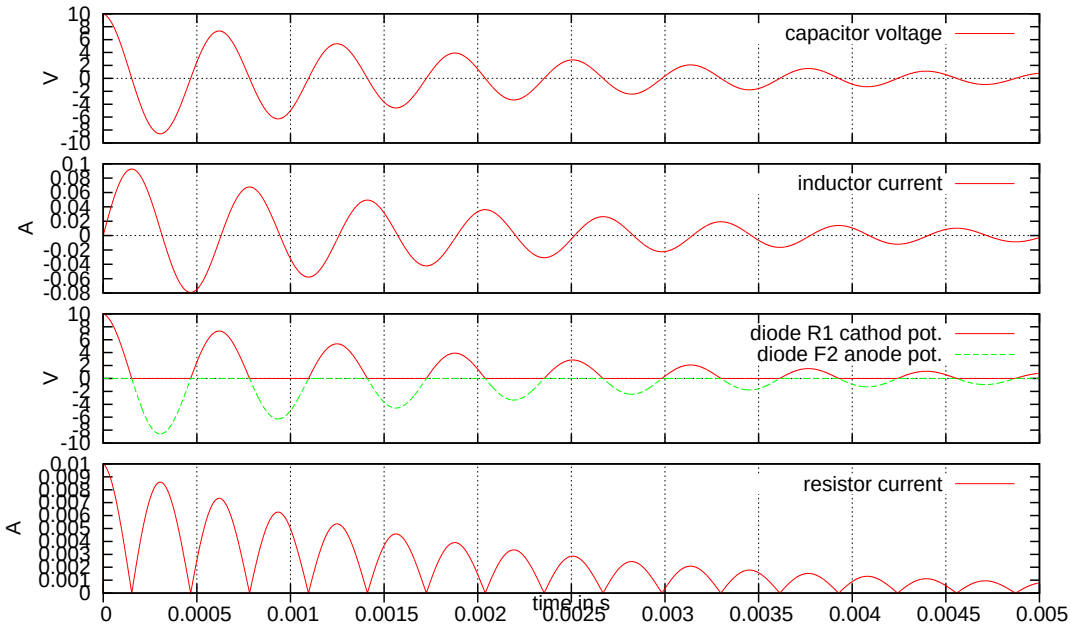


Figure 3: Diodes bridge wave rectifier simulation results.

Part II

Overview of NonSmooth Dynamical Systems (NSDS)

The NonSmooth Dynamical Systems (NSDS) are a special kind of dynamical systems characterized by the nonsmoothness of their time-evolution and of their constitutive equations. The instants of discontinuity of the state or its derivatives can be viewed as events, or transitions, where the system modifies its structure. In this way, a NSDS combines features of continuous dynamical systems with the characteristics of finite automata. Thus, as a mixture of time-continuous dynamics and discrete systems, the NSDS can be viewed as a subclass of hybrid systems.

Like the term "hybrid", the "nonsmooth" one is not a precise definition, in the sense that it is not restrictive enough. A better way to define a coherent class of dynamical systems is to consider the mathematical nature of their solutions depending on the chosen formulation. This introduction aims at giving a flavor of the mathematical properties and of the numerical consequences, that are shared by NSDS. The term "nonsmooth" is partly inherited from the extensive use of a well-recognized mathematical theory: the Nonsmooth Analysis [3]. Since they are specializations of hybrid dynamical systems, new mathematical results can be derived and more efficient simulation tools can be designed. The role of the Siconos platform is then to take advantage of these properties in order to provide general modeling and simulation tools for NSDS.

3 Standard Classes of NSDS

3.1 The Example of the Constrained First Order Dynamics

A standard example of NSDS is a first order dynamical system of the form:

$$\dot{x} = f(x,t), x \in \mathbb{R}^n, t \in [0,T], \tag{9}$$

subjected to a set of constraints on its state[3]:

$$y = h(x) = [h_\alpha(x), \alpha = 1 \dots m]^T \geqslant 0. \tag{10}$$

The constraints (10) are usually enforced by an external input, let us say a multiplier $\lambda \in \mathbb{R}^m$, through an input function g such that

$$\begin{cases} g : \lambda \in \mathbb{R}^m \mapsto g(\lambda) \in \mathbb{R}^n, \\ \dot{x} = f(x,t) + g(\lambda). \end{cases} \tag{11}$$

Finally, in order to complete the system, additional modeling information are needed. Particularly, two laws are of utmost importance

a) A generalized equation [16] between the output y and the multiplier λ, denoted by the following inclusion:

$$0 \in F(y,\lambda) + Q(y,\lambda) \tag{12}$$

where $F : \mathbb{R}^{m\times m} \mapsto \mathbb{R}^{m\times m}$ is assumed to be continuously differentiable and $Q : \mathbb{R}^{m\times m} \rightsquigarrow \mathbb{R}^{m\times m}$ is a multivalued mapping with a closed graph. In simple cases, a complementarity condition is usually introduced for (12):

$$0 \leqslant y \perp \lambda \geqslant 0 \tag{13}$$

[3]The inequality is to be understood component-wise.

b) A reinitialization mapping or an impact law defining the state of the system after a possible nonsmooth event:

$$x(t^+) = \mathcal{F}(x(t^-), t) \tag{14}$$

3.2 Nonsmooth Lagrangian Dynamical Systems or NonSmooth Multibody Systems.

Lagrangian dynamical systems or NonSmooth MultiBody Systems (NSMBS) can be defined by the following set of relations

$$\begin{cases} M(q)\dot{v} = F(t,q,v) + G(q)\lambda & \text{(15a)} \\ \dot{q} = v & \text{(15b)} \\ y = g(t,q,v) & \text{(15c)} \\ 0 \in S(y,\lambda) + T(y,\lambda) & \text{(15d)} \\ v^+ = \mathcal{F}(v^-,q,t) & \text{(15e)} \end{cases}$$

The two first equations (15a) and (15b) define in terms of generalized coordinates, $q \in I\!R^n$ and velocities, $v \in I\!R^n$ the standard dynamics of a multibody systems in which some inputs are modeled by some Lagrange multipliers $\lambda \in I\!R^m$. The matrix $M(q)$ is the mass matrix and the vector $F(t,q,v)$ collects the external forces, internal forces and the gyroscopic accelerations. The equation (15c) defines an output of the state $y \in I\!R^m$.

The inclusion (15d) defines the behavior of the multiplier λ with respect to the output y. As previously mentioned, the function $S : I\!R^{m\times m} \mapsto I\!R^{m\times m}$ is assumed to be continuously differentiable and $T : I\!R^{m\times m} \rightsquigarrow I\!R^{m\times m}$ is a multivalued mapping with a closed graph. Finally, if the evolution is nonsmooth, a reinitialization rule has to be added to specify the behavior of the velocities. This relation is also known as an impact law.

3.2.1 Examples of NonSmooth Lagrangian Dynamical Systems

Most well-known instances of Lagrangian dynamical systems are the multibody systems subjected to nonsmooth bilateral constraints, the multibody systems subjected to perfect unilateral constraints or the multibody systems subjected to unilateral contact with Coulomb's friction.

- *Multibody systems subjected to nonsmooth bilateral constraints.* Choosing $S(y,\lambda) = y = g(t,q)$ and $T(y,\lambda) = 0$, a Differential Algebraic Equation (DAE) is retrieved:

$$\begin{cases} M(q)\dot{v} = F(t,q,v) + G(q)\lambda \\ \dot{q} = v \\ g(t,q) = 0 \end{cases} \tag{16}$$

  If the constraint is not sufficiently smooth, let us say, g is only continuous in q, some nonsmooth solutions have to expected (see for more details Chap 11 in [7]) and then an impact law has to be added $v^+ = \mathcal{F}(v^-,q,t)$.

- *Multibody systems subjected to perfect unilateral constraints.* Choosing $y = g(t,q)$, $S(y,\lambda) = \partial\psi_{I\!R_+}(\lambda)$ and $T(y,\lambda) = \lambda$, where ψ_K is the indicator function of the set K and the symbol ∂ denotes the subdifferential in the sense of the Convex Analysis, we obtain,

$$\begin{cases} M(q)\dot{v} = F(t,q,v) + G(q)\lambda \\ \dot{q} = v \\ y = g(t,q) \\ 0 \leqslant y \perp \lambda \geqslant 0 \\ v^+ = \mathcal{F}(v^-,q,t) \end{cases} \tag{17}$$

- *Multibody systems subjected to unilateral contact with Coulomb's friction.* The output is defined as $y = [g_n, v_t]^T$ where g_n is the normal gap between two bodies and v_t is the tangential velocity at contact. We assume in the same that λ is decomposed in normal and tangential component as $\lambda = [\lambda_n, \lambda_t]^T$. Choosing $S(y,\lambda) = \lambda$ and $T(y,\lambda) = [\partial\psi_{I\!R_+}(g_n), \mu\lambda_n\partial\|v_t\|]^T$, where μ is the coefficient of friction.

More details on nonsmooth Lagrangian dynamical systems can be found in the following references [15, 2].

On one side the Lagrangian dynamical system can be understood as an hybrid system with 2^m modes, a mode being defined by the fact that each constraint $\alpha = 1 \dots m$, is active or not. In each mode, the solution is assumed to be smooth with discontinuities arising at transitions between two modes. On the other side, it can be considered as a single NSDS. Its solution is then defined as a global one, possibly nonsmooth, containing the instants of discontinuity. Usually, for Lagrangian systems, the coordinates are considered as absolutely continuous functions of time, the velocities as functions of bounded variations and accelerations as measures [17, 13, 12].

3.3 Complementarity Systems.

A generalized dynamical complementarity system in a semi-explicit form is defined by

$$\begin{cases} \dot{x}(t) = f(x(t), t, \lambda(t)) \\ \\ y(t) = h(x(t), \lambda(t)) \\ \\ C^* \ni y(t) \perp \lambda(t) \in C \end{cases} \tag{18}$$

where $C \subset I\!R^n$ and $C^* = \{x \in I\!R^n \mid x^T y \geqslant 0 \text{ for all } y \in C\}$ is its dual set. The term generalized is used when the set C is a general cone of $I\!R^n$. If this cone is a nonnegative orthant like $I\!R^N_+$, we speak of DCS or shortly CS. A dynamical complementarity system (DCS) in an explicit form is defined by

$$\begin{cases} \dot{x}(t) = f(x(t), t, \lambda(t)) \\ \\ y(t) = h(x(t), \lambda(t)) \\ \\ 0 \leqslant y(t) \perp \lambda(t) \geqslant 0 \end{cases} \tag{19}$$

In [8], the authors study the so-called Linear Complementarity Systems (LCS) defined by

$$\begin{cases} \dot{x}(t) = Ax(t) + B\lambda(t) \\ \\ y(t) = Cx(t) + D\lambda(t) \\ \\ 0 \leqslant y(t) \perp \lambda(t) \geqslant 0 \end{cases} \tag{20}$$

In this type of systems, a linear output y of a linear dynamical system is defined as the complementary variable of an input λ. The linear complementarity condition can model for instance the behavior of an ideal diode. If the output y represents the current through the diode, this condition states that this current must be positive. The multiplier λ is then the opposite of the voltage across the diode. The graph of this relation is illustrated on Figure 4.

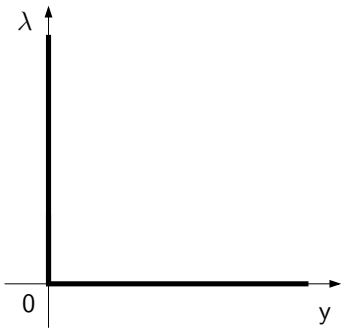


Figure 4: Complementarity condition. $0 \leqslant y \perp \lambda \geqslant 0$.

3.4 Other Classes of NSDS

Without entering into more details, the following system are usually considered to belong to the class of NSDS:

- Ordinary Differential Equations (ODE) with only Lipschitz continuous right-hand sides.
- Differential inclusions with compact and convex set valued mappings (such as Filippov's inclusions).
- Differential inclusions in normal cones.
- Measure differential inclusions.
- Evolution variational inequalities.
- Projected dynamical systems.

The Siconos platform aims at providing in the future release a scientific simulation software for all classes of systems for which some specific algorithms have been designed. For a comprehensive review of NSDS, we refer to the following monographs [2] and [18].

3.5 Comparison with the Hybrid Approach

For simple dynamics (constant or linear) and small systems (up to 100 degrees of freedom), the hybrid approach allows us to exploit the widespread analysis techniques for finite-state systems, such as the verification techniques to check some fundamental properties. For larger systems with fully nonlinear dynamics, it seems that these discrete techniques hardly apply.

In the case of NSDS, the mathematical properties of the solution and the associated numerical techniques allow the simulation and the analysis of large systems to be performed. We claim that the continuous approach is more efficient from the mathematical and the numerical point of view, rather than an event-driven or a discrete approach.

On the mathematical point of view, a NSDS can be considered as a unique time-continuous dynamical system which can encounter discontinuities and reinitializations. That leads to the definition of global solution of such systems in a class of appropriate functions or measures. The case of the constrained Lagrangian dynamics leads for example to the formulation of the dynamics in terms of measure differential inclusions [17, 13, 12] valid on the continuous part of the evolution as well as at the events. Such types of solutions and formulations can comprised complex sequences of events, like concurrent events or accumulation (Zeno), together with mathematical results such as global existence and uniqueness.

4 Simulation of NSDS

4.1 The NonSmooth Approach *vs.* the Hybrid Approach

From the numerical point of view, a nonsmooth approach of hybrid systems exploits the previous notion of solution defined everywhere in time. This fact leads to the design of powerful time–stepping schemes without explicit event-handling procedure. For the case of the Lagrangian dynamics, the measure differential inclusion is evaluated on a fixed time interval and the structural changes of the dynamics are taken into account in a weak sense. Contrary to event-driven schemes, such time-stepping ones are proved to be convergent even in the presence of events accumulations.

Another advantage of the nonsmooth approach of NSDS is the algebraic formulation of certain classes of state transitions at events. To shed more light on this aspect, the simple example of an ideal diode might be taken. This behavior can be modeled as a pure logical component thanks to an "if" statement as in Modelica [6]. Indeed recognizing that the curve of the graph in Figure 4 can be parametrized by a parameter s, we can defined the following Modelica script :

$$\begin{aligned} \mathsf{off} &= \mathsf{s} < 0 \\ \lambda &= \textbf{if}\ \ \mathsf{off}\ \ \textbf{then}\ \ -\mathsf{s}\ \ \textbf{else}\ \ 0 \\ y &= \textbf{if}\ \ \mathsf{off}\ \ \textbf{then}\ \ 0\ \ \textbf{else}\ \ \mathsf{s} \end{aligned}$$

The same representations can be performed with ideal switches, piecewise linear model of MOS transistors. The main difficulties to view systems with ideal components this way is that for each new Boolean variable like off, two modes of the hybrid dynamical system are possible. If we introduce n-Boolean variables, in the worst case, 2^n modes have to be checked. Therefore the problem complexity is exponential.

On the contrary, in the nonsmooth approach the discretized problem at each step can be reformulated as a Linear Complementarity Problem (LCP)[4] of the form:

$$\begin{cases} w = Mz + q \\ 0 \leqslant w \perp z \geqslant 0 \end{cases} \tag{21}$$

Under some usual assumptions on the matrix M, (positiveness, n-step property), and on the vector q, numerical algorithms can be used with polynomial complexity, avoiding an exhaustive enumerative verification of each modes in an exponential time algorithm [14].

To conclude, from the mathematical point of view, the nonsmooth framework yields precise definitions of solutions together with uniqueness and existence results under appropriate assumptions. From the numerical point of view, the use of specific algorithms (time–stepping schemes, LCP solvers with polynomial complexity) leads to an efficient simulation environment. Therefore, the Siconos platform is based on these two features.

4.2 Event–driven and Time–stepping Schemes

Two types of methods are available in Siconos to numerically integrate in time NSDS.

- The first one is known as the *Event-driven method* where the time of discontinuities in the state or in its derivative, also called a nonsmooth event, is detected and located. This method can also be referred to *nonsmooth event tracking method.* Between two events, the system is integrated with any standard ODE or Differential Algebraic Equation (DAE) solver with a suitable order according to the regularity of the system. This method for any standard ODE can be very efficient and of any order but suffers for several drawbacks. If the number of events is large, or worse infinite in a finite time interval, the time integration cannot efficiently

advance in time. This is particularly the case when a finite accumulation of impacts is encountered. Secondly, in practice the method is very sensitive to numerical tolerances and accuracy used for the detection of events. Finally, such a method needs a reformulation of the generalized equation at different kinematic levels. This index-like reduction contains some new conditional statements on the unilateral conditions, which leads to new numerical tolerances with theirs associated difficulties. Event-driven approaches are well-suited when the nonsmooth events are rare and well-separated in time.

- The other method is known as *time–stepping method* or *nonsmooth event capturing method.* In such a method, the time-integration is performed with a time step, which do not depend on the exact location of nonsmooth events. The advantages of this class of methods are the convergence proofs and the efficiency, even in the case of finite accumulation of impacts and the ability to work without an accurate event detection. Finally, another practical interest of this method is that it needs lower kinematic reformulation, and even better no reformulation at all for the constraint. The major drawback of this method is its order. It is at best of first order at impact but also on smooth solutions.

As we said the two possibilities are available on the Siconos software. There is also a possibility to take into account exogenous events in a time–stepping approach by controlling the integration through an event manager. The feature allows one to exploit the time–stepping abilities for the nonsmooth events and to drive the simulation when external events (Control, switch, user output, ...) are encountered.

Part III
Siconos Software

5 General Principles of Modeling and Simulation

Siconos software is mostly written in C++ and thus integrally relies on the object oriented paradigm. In this first section we will not get into details on how to build these objects[4], but rather on what they are and what they are used for.

As explained in Part I, the central object is the `Model`. Roughly speaking, a `Model` is a NonSmooth Dynamical System and some instructions on how to simulate its behavior during a given time period. The compulsory process to handle a problem with Siconos is first build a NonSmooth Dynamical System, as explained in Section 5.1 and then describe a simulation strategy, see Section 5.2. Additionally, a control of the `Model` object can possibly be defined, see Section 5.3.

The way the Software is written relies also on this "cutting-out" with clearly separated modeling and simulation components as explained in the last Section 8.

5.1 NSDS Modeling in Siconos Software

A NSDS can be viewed as a set of dynamical systems that may interact in a nonsmooth way trough interactions. The modeling approach in the Siconos platform consists in considering the NSDS as a graph with dynamical systems as nodes and nonsmooth interactions as branches. Thus, to describe each elements of this graph in Siconos, one needs to define a `NonSmoothDynamicalSystem` object composed of a set of `DynamicalSystem` objects and a set of `Interaction` objects.

A `DynamicalSystem` object is no more than a set of equations to describe the behavior of a single dynamical system, with some specific operators, initial conditions and so on. A complete review of the dynamical systems available in Siconos is given in Section 6.1.

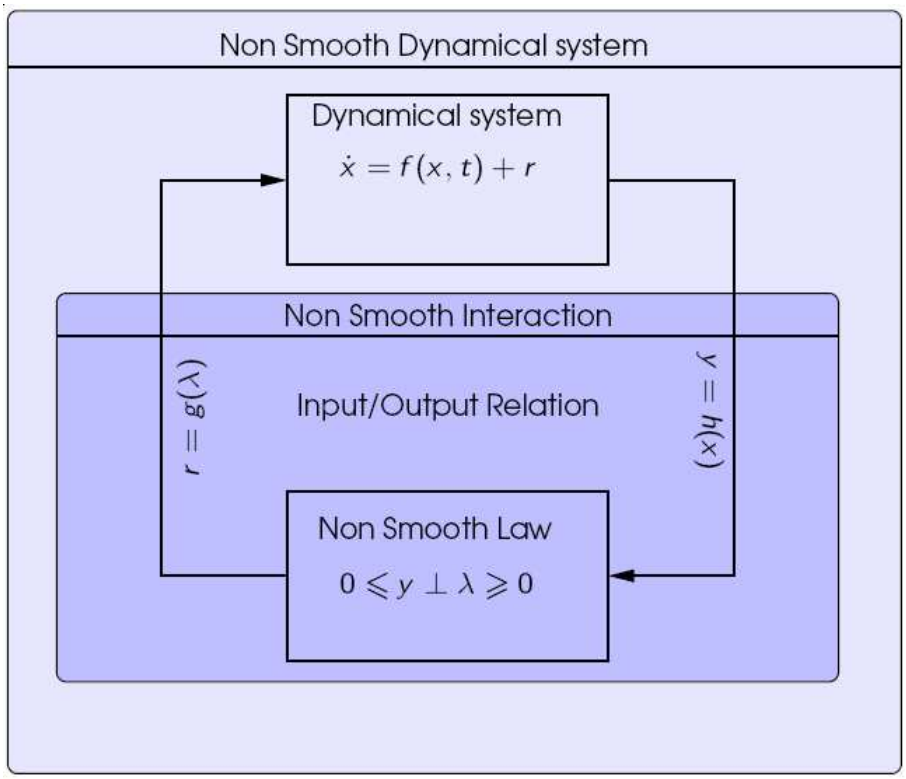
An `Interaction` object describes the way one or more dynamical systems are linked or may interact. For instance, if you consider a set of rigid bodies, the `Interaction` objects define and describe what happens at contact. The `Interaction` object is characterized by some "local" variables, y (also called output) and λ (input) and is composed of:

- a `NonSmoothLaw` object that describes the mapping between y and λ,
- a `Relation` object that describes the equations between the local variables (y, λ) and the global ones (those of the `DynamicalSystem` object).

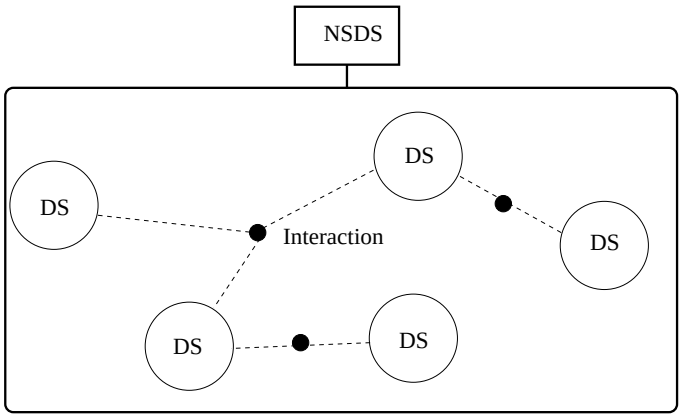
You can find a review of the various possibilities for the `Relation` and the `NonSmoothLaw` objects in Sections 6.2 and 6.3.

As summarized in Figure 5, building a problem in Siconos relies on the proper identification and construction of some `DynamicalSystems` and of all the potential interactions.

[4]This is the role of the Tutorial, Users Guide or the others manuals you can find on `http://siconos.gforge.inria.fr/`



(a) A simple `NonSmoothDynamicalSystem` with one `DynamicalSystem` object and one `Interaction`



(b) The graph structure of a complex NSDS with `DynamicalSystem` objects as nodes and nonsmooth `Interaction` object as branches

Figure 5: Siconos NonSmooth Dynamical System Modeling Principle.

5.2 Simulation Strategies for the NSDS Behavior

Once a NSDS has been fully designed and described thanks to the objects detailed above, it is necessary to build a `Simulation` object, namely to define the way the nonsmooth response of the NSDS will be computed.

First of all, let us introduce the `Event` object, which is characterized by a type and a time of occurrence. Each event has also a `process` method which defines a list of actions that are executed when this event occurs. These actions depend on the object type. For the objects related to nonsmooth time events, namely `NonSmoothEvent`, an action is performed only if an event-driven strategy is chosen. For the `SensorsEvents` and `ActuatorEvent` related to control tools (see Section 5.3), an action is performed for both time–stepping and event–driven strategy at the times defined by the Control law. Finally, thanks to a registration mechanism, user–defined events can be added.

To build the `Simulation` object, we first define a discretisation, using a `TimeDiscretisation` object, to set the number of time steps and their respective size. Note that the initial and final time values are part of the `Model`. The time instants of this discretisation define `TimeDiscretisationEvent` objects used to initialize an `EventsManager` object, which contains the list of `Event` objects and their related methods. The `EventsManager` object belongs to the simulation and will lead the simulation process: the systems integration is always done between a "current" and a "next" event. Then, during simulation, Events of different types may be added or removed, for example when the user creates a Sensor or when an impact is detected.
Thereafter, to complete the `Simulation` object, we need:

- some instructions on how to integrate the smooth dynamics over a time-step, that is the role of the `OneStepIntegrator` objects.
- some details on how to formalize and solve the nonsmooth problems when they occur, this is done with the `OneStepNSProblem` objects.

To summarize, a `Simulation` object is composed of a `TimeDiscretisation`, a set of `OneStepIntegrator` plus a set of `OneStepNSProblem` and belongs to a `Model` object. The whole Simulation process is led by the chosen type of strategy, either time–stepping or event–driven. To proceed, one need to instantiate one of the classes that inherits from `Simulation` object: `TimeStepping` or `EventDriven`.

5.3 Control Tools

In Siconos, some control can be applied on a NSDS. The principle is to get information from the systems thanks to some `Sensor` objects, used by some `Actuator` objects to act on the NSDS components. Each `Sensor` or `Actuator` object has its own `TimeDiscretisation` object, a list of time instants where data are to be captured for sensors or where action occurs for actuators. Those instants are scheduled as events into the simulation's `EventsManager` object and thus processed when necessary.
The whole control process is handled thanks to a `ControlManager` object, which is composed of a set of `Sensor` objects and another set of `Actuator` objects. The `ControlManager` object "knows" the `Model` object and thus all its components.

Each `DynamicalSystem` object has a specific variable, named z, which is a vector of discrete parameters (see section 6.1). To control the systems with a sampled control law, the `Actuator` object sets the values of z components according to the user instructions.

6 NSDS Related Components

In the following paragraphs, we turn our attention to the specific types of systems, relations and laws available in the platform.

6.1 Dynamical Systems

The most general way to write dynamical systems in Siconos is:

$$g(\dot{x}, x, t, z) = 0$$

which is a n-dimensional set of equations where

- t is the time,
- $x \in I\!R^n$ is the state[5]
- the vector of algebraic variables $z \in I\!R^s$ is a set of discrete states, which evolves only at user specified events. The vector z may be used to set some perturbation parameters or to stabilize the system with a sampled control law.

Under some specific conditions, we can rewrite this as:

$$\dot{x} = rhs(x, t, z)$$

where "rhs" means right-hand side. Note that in that case $\nabla_{\dot{x}} g$ must be invertible. From this generic interface, some specific dynamical systems are derived, to fit with different application fields. They are separated in two categories: first and second order (Lagrangian) systems, and then specialized according to the type of their operators (linear or not, time invariant ...).

The following list reviews the dynamical system implemented in Siconos:

- `FirstOrderNonLinearDS` class, which describes the nonlinear dynamical systems of first order in the form,

$$M\dot{x}(t) = f(t, x(t), z) + r \tag{22}$$
$$x(t_0) = x_0 \tag{23}$$

  with M a $n \times n$ matrix, $f(x, t, z)$ the vector field and r the input due to the nonsmooth behavior.

- `FirstOrderLinearDS` class, which describes the linear dynamical systems of first order in the form (coefficients may be time-invariant or not)

$$\dot{x}(t) = A(t, z)x(t) + b(t, z) + r \tag{24}$$
$$x(t_0) = x_0. \tag{25}$$

  Electrical circuits for instance fit to this formalism, as shown in the Diode Bridge example in part I.

[5] The typical dimension of the state vector can range between a few degrees of freedom and more than several hundred thousand, for example for mechanical or electrical systems. The implementation of the software has been done to deal either with small or large scale problems.

- `LagrangianDS` class, which describes the Lagrangian nonlinear dynamical systems in the form,

$$M(q,z)\ddot{q} + \mathrm{NNL}(\dot{q},q,z) + F_{int}(t,\dot{q},q,z) = F_{ext}(t,z) + p \quad (26)$$
$$q(t_0) = q_0, \quad \dot{q}(t_0) = v_0 \quad (27)$$

  where q denotes the generalized coordinates, NNL the nonlinear inertia operator, F_{int} the internal, nonlinear, forces and F_{ext}, the external forces, depending only on time. This formalism corresponds to Mechanics, and can be written in a simpler manner as:

$$M(q,z)\ddot{q} = f_L(t,\dot{q},q,z) + p \quad (28)$$
$$q(t_0) = q_0 \quad , \quad \dot{q}(t_0) = v_0 \quad (29)$$

  The full-form (26) with several operator has been designed to fit with different users habits, depending on the application field (Multibody mechanics, Robotics, Solid and structures Mechanics through Finite Element Method (FEM)).

- `LagrangianLinearTIDS` class, which describes the Lagrangian linear and time invariant coefficients systems:

$$M\ddot{q} + C\dot{q} + Kq = F_{ext}(t,z) + p \quad (30)$$
$$q(t_0) = q_0, \quad \dot{q}(t_0) = v_0 \quad (31)$$

  where C and K are respectively the classical viscosity and stiffness matrices.

As illustrated on the Figure 6, all the classes inherit from the `DynamicalSystem` class.

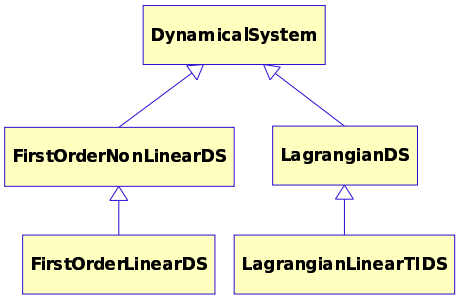


Figure 6: DynamicalSystem-type classes

6.2 Relations

As explained above, and according to Part II notation, some relations between local, (y,λ), and global variables (x,r), have to be set to describe the interactions between systems. The general form of these algebraic equations is:

$$\begin{aligned} y &= \mathrm{output}(x,t,z,\ldots) \\ r &= \mathrm{input}(\lambda,t,z,\ldots) \end{aligned} \quad (32)$$

and is contained in the abstract `Relation` class. Any other `Relation` objects are derived from this one.

As for `DynamicalSystems` they are separated in first and second order relations, and specified according to the type and number of the variables, the linearity of the operators, *etc.* The possible cases are:

- `FirstOrderR` class, which describes the nonlinear relations of first order as:

$$\begin{aligned} y &= h(X,t,Z) \\ R &= g(\lambda,t,Z) \end{aligned} \tag{33}$$

  Note that we use upper case for all variables related to `DynamicalSystem` objects. Remember that a `Relation` object applies trough the `Interaction` object to a set of dynamical systems, and thus, X, Z,... are concatenation of x, z,... of the `DynamicalSystem` objects concerned by the relation.

- FirstOrderLinearTIR class, which describes the first order linear and time invariant relations:

$$y = CX + FZ + D\lambda + e \tag{34}$$
$$R = B\lambda \tag{35}$$

  Once again, see for instance the Diode Bridge example in part I.

- `LagrangianScleronoumousR` class: the scleronomic constraints case, where the relation depends only on the global coordinates of the Dynamical Systems.

$$y = h(Q,Z) \tag{36}$$
$$\dot{y} = G_0(Q,Z)\dot{Q} \tag{37}$$
$$P = \nabla^t h(Q,Z)\lambda = G_0^t(Q,Z)\lambda \tag{38}$$

  with

$$G_0(Q,Z) = \nabla_Q h(Q,Z) \tag{39}$$

- `LagrangianRheonomousR`: in that case, the relation depends also on time.

$$y = h(Q,t,Z) \tag{40}$$
$$\dot{y} = G_0(Q,t,Z)\dot{Q} + \frac{\partial h}{\partial t}(Q,t,Z) \tag{41}$$
$$P = G_0^t(Q,t,Z)\lambda \tag{42}$$

  with

$$G_0(Q,t,Z) = \nabla_Q h(Q,t,Z) \tag{43}$$

- `LagrangianCompliantR` class: there, the relation depends on λ. For instance in the mechanical case, this may corresponds to a spring, since it links more or less a force to a displacement.

$$y = h(Q,\lambda_0,Z) \tag{44}$$
$$\dot{y} = G_0(Q,\lambda_0,Z)\dot{Q} + G_1((Q,\lambda_0,Z)\lambda_1 \tag{45}$$
$$P = G_0^t(Q,\lambda_0,Z)\lambda_0 \tag{46}$$

  with

$$G_0(Q,\lambda_0,Z) = \nabla_Q h(Q,\lambda_0,Z) \tag{47}$$
$$G_1(Q,\lambda_0,Z) = \nabla_{\lambda_0} h(Q,\lambda_0,Z) \tag{48}$$

  and λ_0 the multiplier corresponding to y, while λ_1 corresponds to $\dot{y}$.

- `LagrangianLinearR` class: the simplest one, with linear and time invariant relations between local and global variables.

$$y = HQ + D\lambda + FZ + b \tag{49}$$
$$P = H^t\lambda \tag{50}$$

As shown on the Figure 7, all the classes inherit from the `Relation` class.

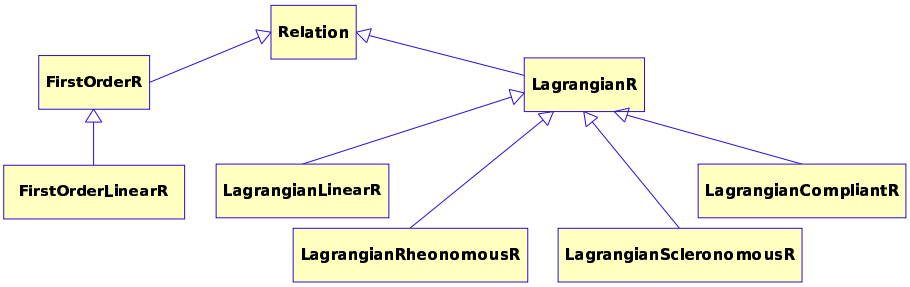


Figure 7: Relation-type classes.

6.3 Nonsmooth Laws

The `NonSmoothLaw` object is the last required object to complete the `Interaction` object. We present here a list of the existing laws in Siconos.

- `ComplementarityConditionNSL` class which models a complementarity condition as :

$$0 \leqslant y \perp \lambda \geqslant 0 \tag{51}$$

- `NewtonImpactNSL` class which models the unilateral contact with the Newton's impact law, known also as the Moreau's impacting rule:

$$\text{if } y(t) = 0, \quad 0 \leqslant \dot{y}(t^+) + e\dot{y}(t^-) \perp \lambda \geqslant 0 \tag{52}$$

- `RelayNSL` class which models the simple relay mapping as

$$\begin{cases} \dot{y} = 0, |\lambda| \leqslant 1 \\ \dot{y} \neq 0, \lambda = \text{sign}(y) \end{cases} \tag{53}$$

- `NewtonImpactFrictionNSL` class which models the unilateral contact with Coulomb's friction in 2D and 3D as

$$y = [y_{\boldsymbol{n}}, y_{\boldsymbol{t}}]^T, \lambda = [\lambda_{\boldsymbol{n}}, \lambda_{\boldsymbol{t}}]^T \tag{54}$$

$$\text{if} \quad y_{\boldsymbol{n}} = 0, \begin{cases} 0 \leqslant \dot{y}_{\boldsymbol{n}} \perp \lambda_{\boldsymbol{n}} \geqslant 0 \\ \dot{y}_{\boldsymbol{t}} = 0, \|\lambda_{\boldsymbol{t}}\| \leqslant \mu\lambda_{\boldsymbol{n}} \\ \dot{y}_{\boldsymbol{t}} \neq 0, \lambda_{\boldsymbol{t}} = -\mu\lambda_{\boldsymbol{n}}\text{sign}(\dot{y}_{\boldsymbol{t}}) \end{cases} \tag{55}$$

- `PiecewiseLinearNSL` class which models one-dimensional piecewise linear set-valued mapping with fill in graphs as depicted on the Figure 8

7 Simulation Related Components

7.1 Integration of the Dynamics

To integrate the dynamics over a time-step or between two events, `OneStepIntegrator` objects have to be defined. Two types of integrators are available at the time in the platform, listed below and represented on Figure 10.a:

- `Moreau` class for Moreau's time stepping scheme, based on a θ-method.
- `Lsodar` class for the Event Driven strategy; this class is an interface for LSODAR, odepack integrator (see `http://www.netlib.org/alliant/ode/doc`).

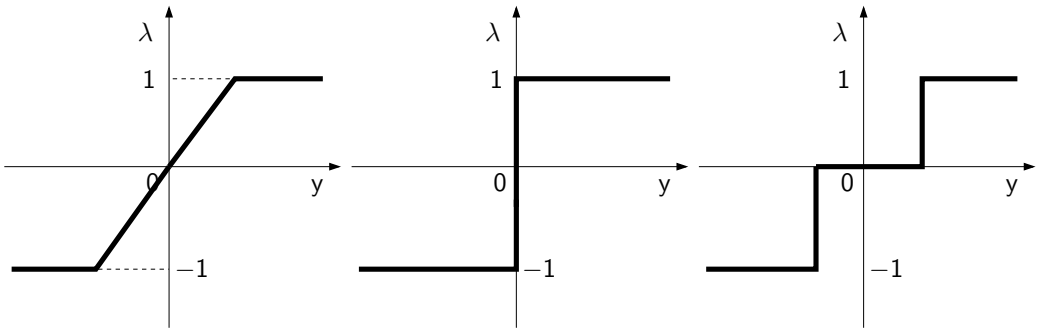


Figure 8: some multivalued piecewise linear laws: saturation, relay, relay with dead zone

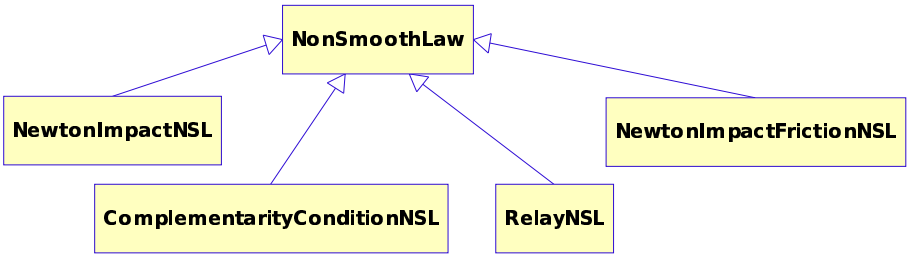


Figure 9: NonSmoothLaw-type classes.

7.2 Formalization and Solving of the Nonsmooth Problems

Depending on the encountered situation, various formalizations for the nonsmooth problem are available.

- `LCP` class which describes the Linear Complementarity Problem,

$$\begin{cases} w = Mz + q \\ 0 \leqslant w \perp z \geqslant 0 \end{cases}$$

- `FrictionContact2D(3D)` class, for two(three)-dimensional contact and friction problems
- `QP` class for the Quadratic Programming problem

$$\begin{cases} \min \dfrac{1}{2} z^T Q z + z^T p \\ z \geqslant 0 \end{cases}$$

- `Relay` class for the relay problem

From a practical point of view, the solving of nonsmooth problems relies on low level algorithms (from the Siconos/Numerics package). The available nonsmooth solvers for LCP are LexicoLemke, PGS (Projected Gauss Seidel), QP (Quadratic Programming), CPG (Conjugate Projected Gradient).

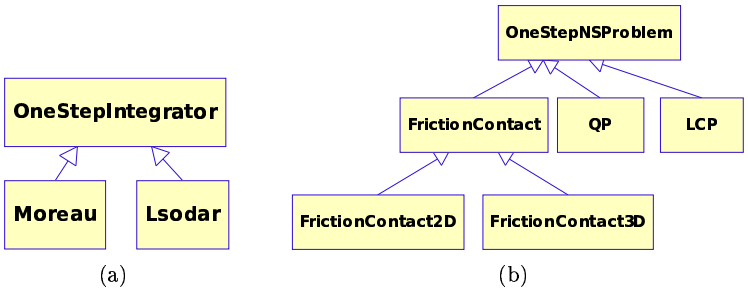


Figure 10: (a) One-Step Integrators classes - (b) One-Step Nonsmooth Problem classes.

8 Siconos Software Design

8.1 Overview

Siconos is composed of three main parts: Numerics, Kernel and Front-End, as represented on Figure 11 below.

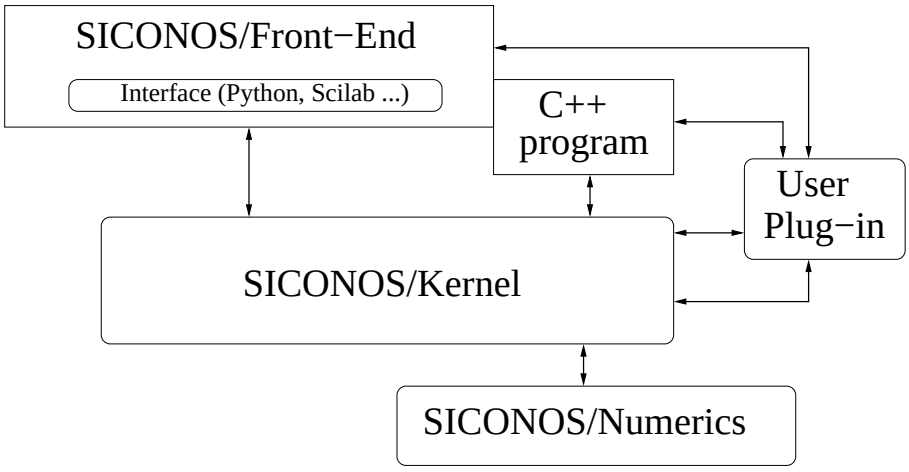


Figure 11: General design of Siconos software

The **Siconos/Kernel** is the core of the software, providing high level description of the studied systems and numerical solving strategies. It is fully written in C++, using extensively the STL utilities. A complete description of the Kernel is given in Section 8.2.

The **Siconos/Numerics** part holds all low-level algorithms, to compute basic well-identified problems (ordinary differential equations, LCP, QP ...) and is based on BLAS/LAPACK linear algebra routines and ODEPACK. It is written in C and FORTRAN.

The last component, **Siconos/Front-End**, provides interfaces with some specific command-languages such as Python or Scilab. This to supply more pleasant and easy-access tools for users, during pre/post–treatment.

Note that while the Kernel cannot work without Numerics, Front-End is only an optional pack.

8.2 Siconos Kernel Components

As previously said, Kernel is the central and main part of the Software. The whole dependencies among Kernel parts are fully depicted on Figure 12 below.

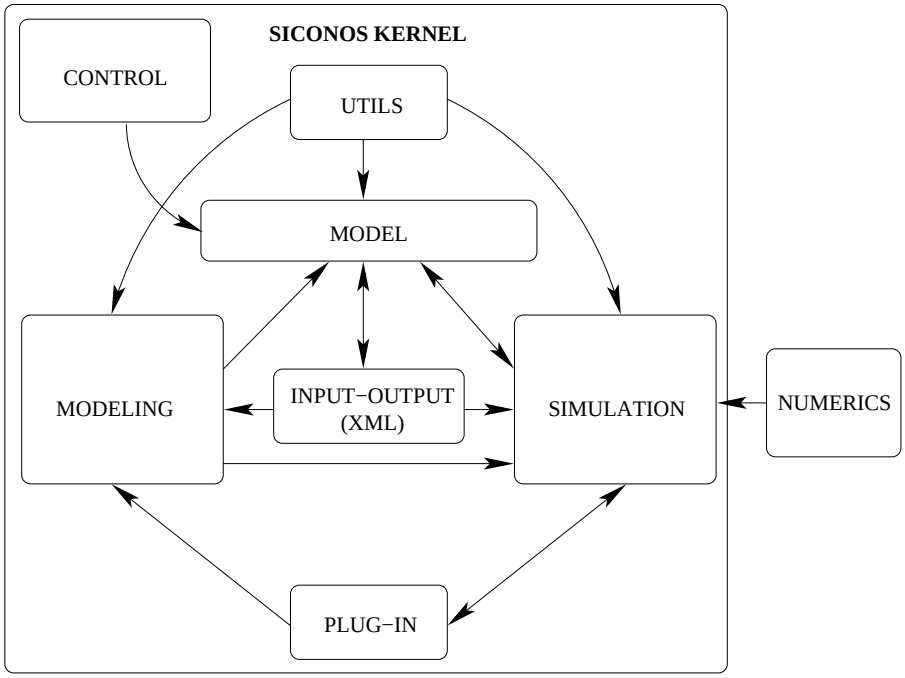


Figure 12: Kernel components dependencies.

All the Kernel implementation is based on the logic we gave in Section 5. It is mainly composed of two rather distinct parts, modeling and simulation, that hold all the objects used respectively in the NSDS modeling (see Section 5.1) and the Simulation description (see Section 5.2).

The Utils module contains tools, mainly to handle classical objects such as matrices or vectors and is based on the Boost library[6], especially, uBLAS[7], a C++ library that provides BLAS functionalities for vectors, dense and sparse matrices.

The Input-output module concerns objects for data management in XML format, thanks to the libxml2 library. More precisely, all the description of the Model, NSDS and Simulation, can be done thanks to an XML input file. A example of such a file is given in Section 9.1.

Control package provides objects like `Sensor` and `Actuator`, to add control of the dynamical systems through the `Model` object, as explained in Section 5.3.

A plug-in system is available, mainly to allow the user to provide his own computation methods for some specific functions (vector field of a dynamical system, mass ...), this without having to re-compile the whole platform. Moreover, the platform is designed in a way that allows user to add dedicated modules through object registration and object factories mechanisms (for example to add a specific nonsmooth law, a user-defined sensor ...).

To conclude, class diagrams for modeling and simulation components are given in Figures 13 and 14, which make clearer the various links between all the objects presented before.

[6] `http://www.boost.org`
[7] `http://www.boost.org/libs/numeric/ublas/doc/index.htm`

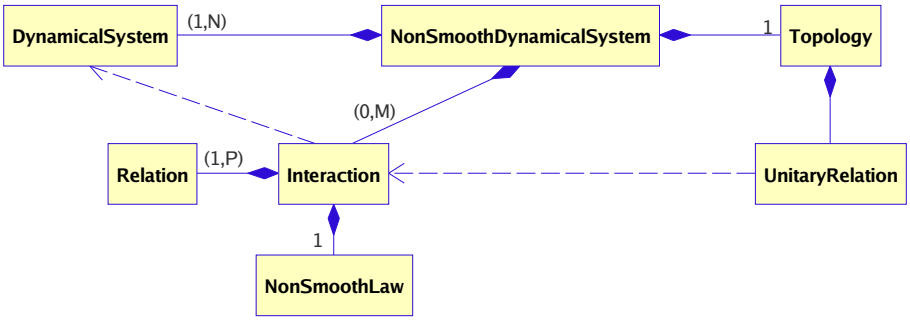


Figure 13: Simplified class diagram for Kernel modeling part.

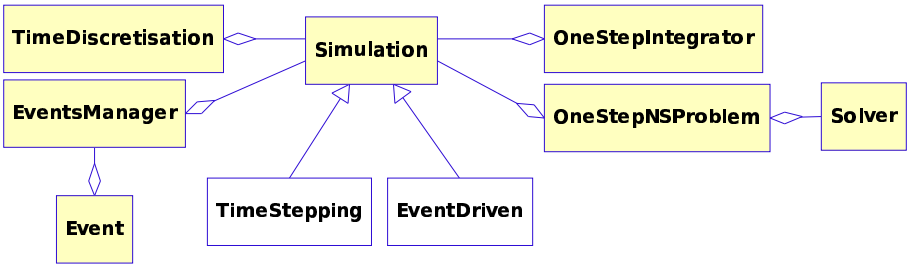


Figure 14: Simplified class diagram for Kernel simulation part.

Note that the above presentation is only an overall view which is moreover likely to change, so if you are interested in, we encourage you to check on the `http://siconos.gforge.inria.fr/` pages for the last updates.

Part IV
Examples

9 Mechanical examples

9.1 The Bouncing Ball(s)

As shown in the figure below, we consider a ball of mass m and radius R, described by 3 generalized coordinates $q = (z, x, \theta)$. The ball is subjected to the gravity g. The system is also constituted by a rigid plane, defined by its position h with respect to the axis Oz. We assume that the position of the plane is fixed.

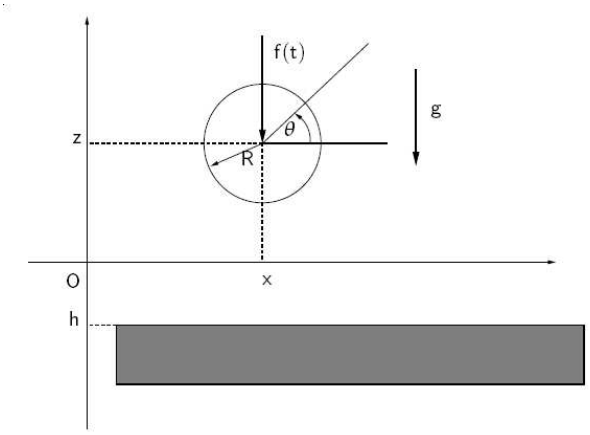


Figure 15: A ball bouncing on the ground.

The equation of motion of the ball is given by:

$$M\ddot{q} = F_{ext}(t) + p \tag{56}$$

with M the inertia term, p the force due to the non-smooth law, i.e., the reaction at the impact times and $F_{ext}(t) : \mathcal{R} \mapsto \mathcal{R}^n$ the given external force.

$$M = \begin{bmatrix} m & 0 & 0 \\ 0 & m & 0 \\ 0 & 0 & I \end{bmatrix}, \quad I = \tfrac{3}{5}mR^2, \quad F_{ext} = \begin{bmatrix} -mg \\ 0 \\ 0 \end{bmatrix} \tag{57}$$

The ball bounces on the rigid plane, introducing a constraint on its vertical position, given by:

$$z - R - h \geqslant 0 \tag{58}$$

We introduce y as the distance between the ball and the floor and λ as the multiplier that corresponds to the reaction at contact. Then from (58), we get:

$$y = Hq + b = [1\ 0\ 0]q - R - h \tag{59}$$

completed by:

$$p = H^t\lambda \tag{60}$$

Finally we need to introduce a non-smooth law to define the behavior of the ball at impact. The unilateral constraint is such that:

$$0 \leqslant y \perp \lambda \geqslant 0 \tag{61}$$

completed with a Newton Impact law, for which we set the restitution coefficient e to 0.9:

$$\text{if } y = 0,\ \dot{y}(t^+) = -e\dot{y}(t^-) \tag{62}$$

t^+ and t^- being post and pre-impact times.

Then, (56) fits with `LagrangianLinearTIDS` (30) formalism, (59) and (60) with `LagrangianLinearR` (49) and (61) and (62) with `NewtonImpactNSL` (52). If we use XML for the Model description, the part corresponding to the NSDS will look like:

```
<NSDS bvp='false'>
   <DS_Definition>
      <LagrangianLinearTIDS number='1'>
         <Id> Ball </Id>
         <q0 vectorSize='3'>1.0 0.0 0.0</q0>
         <Velocity0  vectorSize='3'>0.0 0.0 0.0</Velocity0>
         <FExt vectorPlugin="BallPlugin:ballFExt"/>
         <Mass matrixRowSize='3' matrixColSize='3'>
            <row>1.0 0.0 0.0</row>
            <row>0.0 1.0 0.0</row>
            <row>0.0 0.0 1.0</row>
            </Mass>
      </LagrangianLinearTIDS>
   </DS_Definition>
   <Interaction_Definition>
      <Interaction number='1' Id='Ball-Ground'>
         <size> 1 </size>
         <DS_Concerned all='true'></DS_Concerned>
         <Interaction_Content>
            <LagrangianLinearRelation>
               <H matrixRowSize='1' matrixColSize='3'>
                  <row> 1.0  0.0  0.0</row>
               </H>
            </LagrangianLinearRelation>
            <NewtonImpactLaw>
               <e>0.9</e>
            </NewtonImpactLaw>
         </Interaction_Content>
      </Interaction>
   </Interaction_Definition>
</NSDS>
```

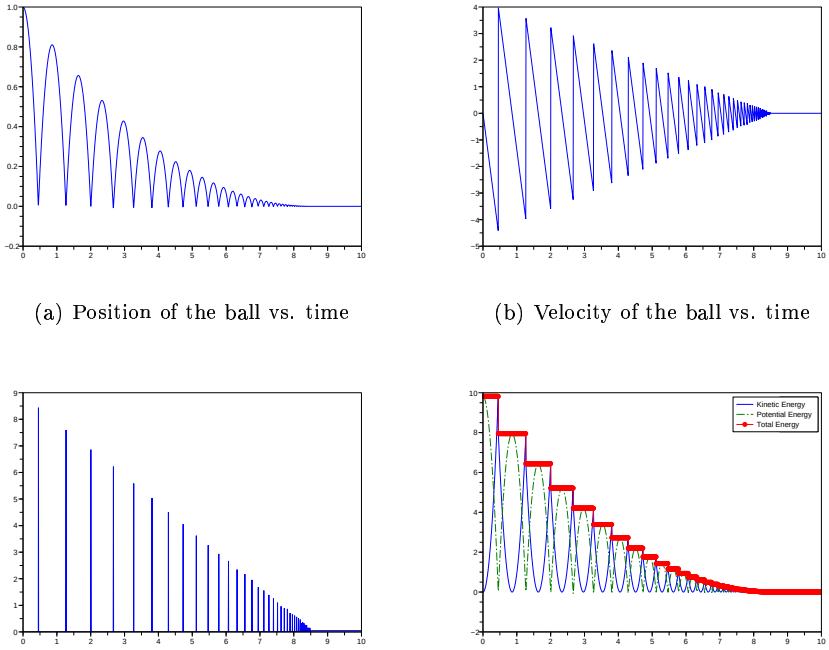
For the simulation, we use a Moreau's time-stepping scheme and an LCP formalization with a Lemke solver:

```
<Simulation type='TimeStepping'>
   <TimeDiscretisation>
      <h>0.005</h>
   </TimeDiscretisation>
   <OneStepIntegrator_Definition>
      <Moreau>
         <DS_Concerned vectorSize='1'>1</DS_Concerned>
         <Theta all="0.5"></Theta>
      </Moreau>
   </OneStepIntegrator_Definition>
   <OneStepNSProblem>
      <LCP>
         <Solver type="Lemke" maxIter="101" />
      </LCP>
   </OneStepNSProblem>
</Simulation>
```

And then, in the C++ input file we have:

```
// Loading of the XML input file that describes the model:
Model * bouncingBall = new Model("./BallTS.xml");
// Get the simulation object
Simulation* s = bouncingBall->getSimulationPtr();
// ...
// Initialize and run ...
s->initialize();
s->run();
```

The results are presented on Figure 16.



(a) Position of the ball vs. time

(b) Velocity of the ball vs. time

(c) Reaction due to the contact force vs. time

(d) Energy balance vs. time

Figure 16: Simulation results (time-stepping) for a ball bouncing on the ground.

We consider now a column of 1000 spherical beads, in contact or not, falling down to the ground. The modeling is quite the same as for the single ball, it is just necessary to define more dynamical systems, one for each bead, and more interactions, one for each potential contact between two beads. The interest of this example lies in the important number of degrees of freedom (i.e.size of vector q) and of relations (size of y and λ) equal to 1000. Figure 17 displays vertical displacements of the 8 lowest beads according to time.

9.2 A Lagrangian Nonlinear System: Simulation of a Robotic Arm

As a second example, we consider now the Mitsubishi PA-10 Robot, a seven degree–of–freedom anthropomorphic robot, presented on Figure 18.

In Siconos, this robotic arm is modeled using a Lagrangian (nonlinear) dynamical system, that interacts with the ground. The arm falls down due to its own weight and bounces on the ground. Moreover, articular stops have been included to limit angular rotations. The degrees of freedom are the rotation angles, represented by the vector q. Neither external forces nor control terms are present. Finally, the robot dynamics is cast into the framework of Lagrangian dynamical systems

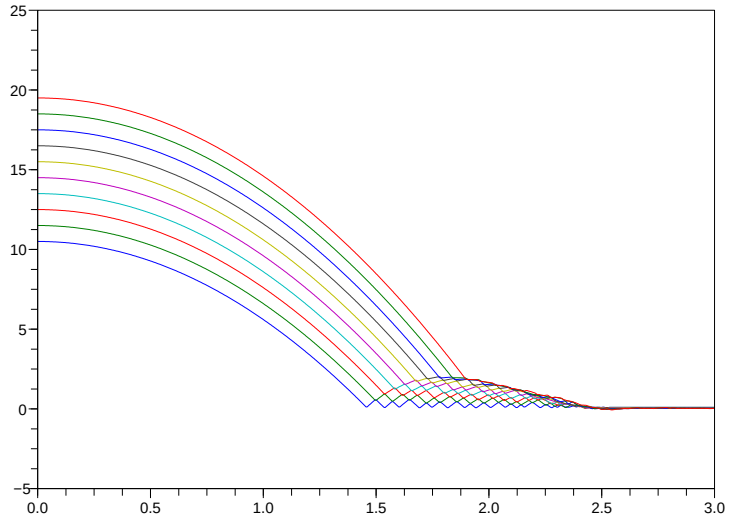


Figure 17: Vertical displacement of the ten lowest beads according to time.

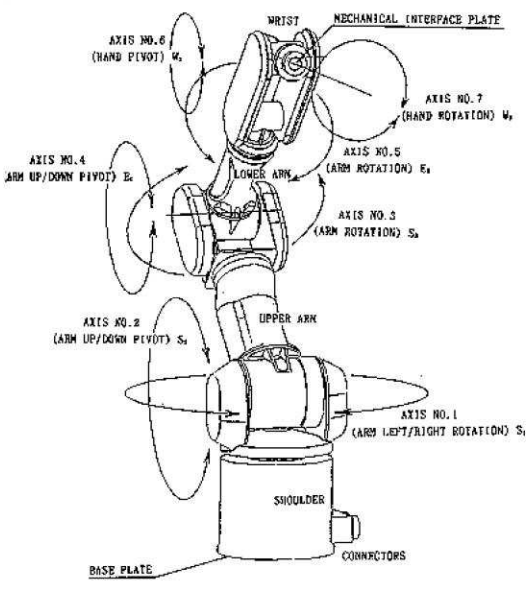


Figure 18: Mitsubishi PA-10 robotic arm

as in (26). As a relation, we set that the distance between the arm and the ground must remain positive, which means that contacts can occur but without penetration. The contact is supposed to be frictionless with a restitution coefficient denoted as e. This leads to nonlinear links between the angular position (i.e. components of q), that can be written as:

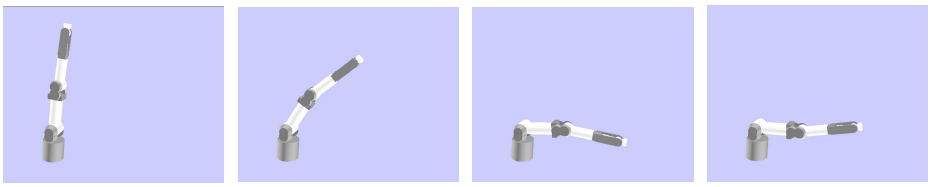
$$\dot{y} = \nabla_q h(q)\dot{q} \quad r = g(q)\lambda = \nabla_q^t h(q)\lambda \tag{63}$$

A Newton-Impact non smooth law complete this set of equations.

$$\dot{y}(t^+) = -e\dot{y}(t^-) \tag{64}$$

with a restitution coefficient $e = 0.9$ for contact with the ground and $e = 0$ for angular stops.

The results of Siconos simulation, using Moreau time-stepping, Newton algorithm and a non



(a) (b) (c) (d)

Figure 19: robotic arm fall-down (a) initial position - (b) before contact with ground - (c) contact - (d) post-contact

smooth quadratic programming solver, are presented on Figures 19 and 20. We denote A and B respectively the first (the one clamped on the ground) and second parts of the arm; θ_1 is the angle between A and the vertical position and θ_2 the angle between A and B. First, θ_1 and θ_2 increase, until the last one reaches its maximum angular position. Then θ_1, goes on increasing and θ_2 remains constant until the extremity of the arm touches the ground and bounces, here with a restitution coefficient e equal to 0.9. Finally A touches the ground and bounces also, together with B, until the complete arm lies on the ground. Note on 20.c and d that the angular velocity cancels when the angular stops are reached, and change their signs when contact is established.

9.3 The Woodpecker Toy

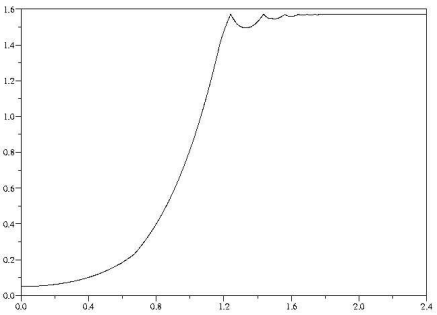
This system has been implemented in Siconos by M. Moeller from ETH Zurich, following the examples proposed in [10]. The Woodpecker toy is presented on Figure 21.a and b and consists in a sleeve, a spring and the woodpecker. The hole in the sleeve is slightly larger than the diameter of the pole, thus allowing a kind of pitching motion interrupted by impacts with friction. Its dynamical behavior shows both impact and friction phenomena.

The Woodpecker Toy is a system which can only operate in the presence of friction as it relies on combined impacts and jamming. Among other things, an animation of the toy can be found there: `http://www.zfm.ethz.ch/ leine/toys.htm`.

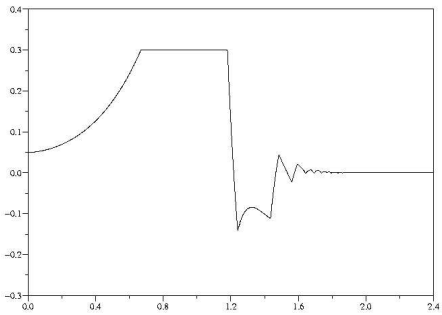
Some results obtained with Siconos are presented on Figure 22.

9.4 The Cam Follower System

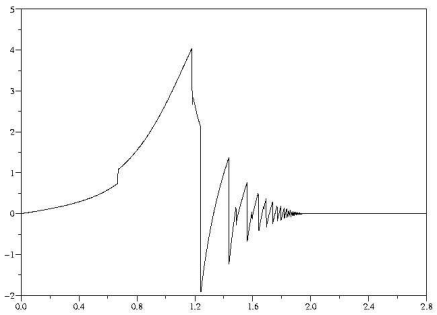
The cam-follower system has been proposed and implemented by G. Osorio, Mario di Bernardo and Stefania Santini from University of Naples Federico II, Italy.



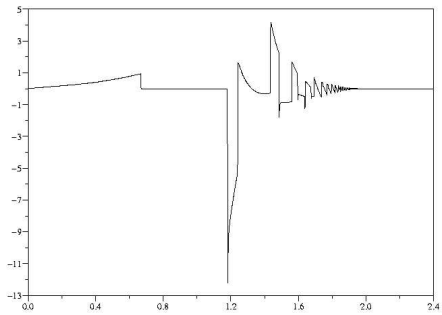
(a) $\theta_1(t)$



(b) $\theta_2(t)$

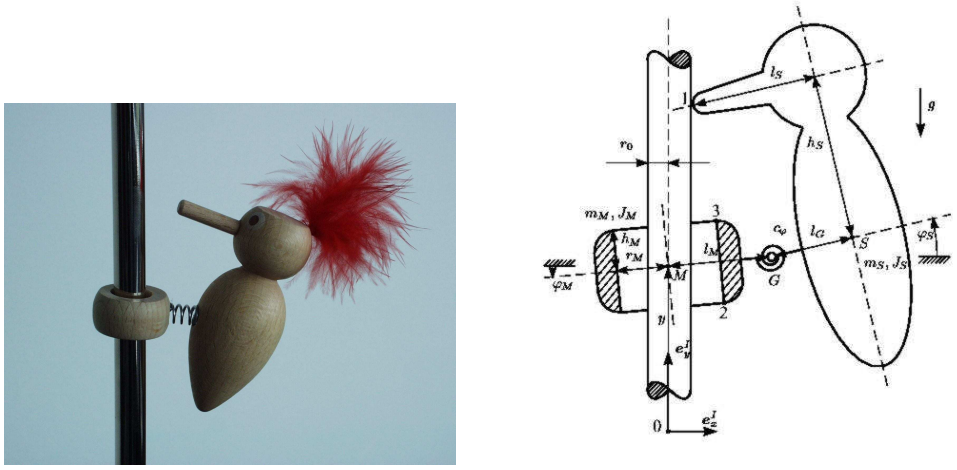


(c) $\dot{\theta}_1(t)$



(d) $\dot{\theta}_2(t)$

Figure 20: Siconos simulation of the robotic arm



(a) (b)

Figure 21: The woodpecker toy.

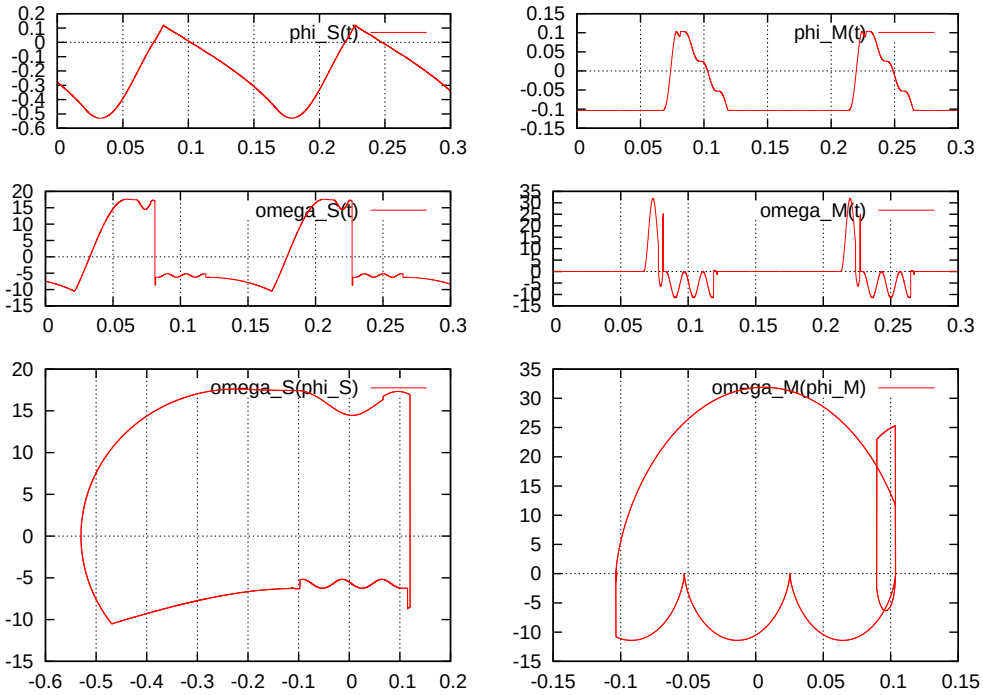


Figure 22: Simulation results for the woodpecker toy using Siconos platform.

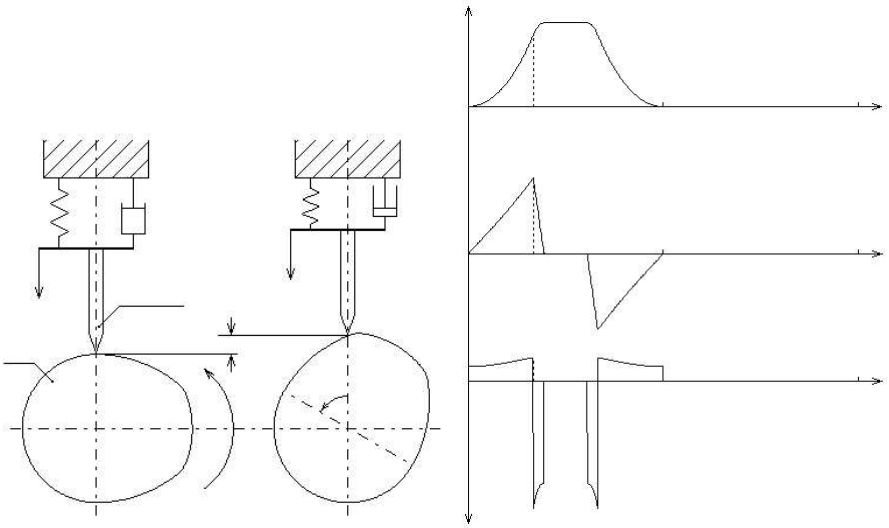
The cam-follower system is represented on Figure 23. The free body dynamics can be described by a linear second order system. An external input is considered acting directly on the follower. This input is a nonlinear forcing component coming from the valve. The follower motion is constrained to a phase space region bounded by the cam position. The non conservative Newton restitution law is used for the computation of the post impact velocity. The cam is assumed to be massive, therefore only rotational displacement is allowed. Under these assumptions, the free body dynamics of the follower can be described by:

$$\mu \frac{d^2u(t)}{dt^2} + \zeta \frac{du(t)}{dt} + \kappa u(t) = f_v(t),\ if\ u(t) > c(t). \tag{65}$$

where μ, ζ and κ are constant parameters for the follower mass, friction viscous damping and spring stiffness respectively. The state of the follower is given by the position $u(t)$ and velocity $v(t) = \frac{du}{dt}$. The external forcing is given by $f_v(t)$. The cam angular position determines $c(t)$ that defines the holonomic (i.e. constraint only on the position) rheonomic (i.e. time varying) constraint. The dynamic behavior when impacts occurs (i.e. $u(t) = c(t)$) is modelled via Newton's impact law that in this case is given by this case is given by:

$$v(t^+) = \frac{dc}{dt} - r\left(v(t^-) - \frac{dc}{dt}\right) = (1+r)\frac{dc}{dt} - rv(t^-),\ if\ u(t) = c(t). \tag{66}$$

where $v(t^+)$ and $v(t^-)$ are the post and pre impact velocities respectively, $\frac{dc}{dt}$ is the velocity vector of the cam at the contact point with the follower, and $r \in [0,1]$ is the restitution coefficient to model from plastic to elastic impacts. In Figure 23 is presented the schematic diagram of the physical cam-follower system. In Figure 23.a for $t = 0$, 1.b for $t = \beta$, and 23.c the profile of the constraint position $\delta c(t)$, velocity $\frac{dc}{dt}(t)$ and acceleration $\frac{d^2c}{dt^2}(t)$. It is possible to visualize the follower displacement as a function of the cam position. It is also important to notice that different types of cams and followers profiles are used in practical applications.



(a) (b)

Figure 23: (a) Schematic diagram of the cam-follower system - (b) profiles of the constraint position.

It is possible to completely describe the cam-follower system as a driven impact oscillator into the framework of Lagrangian NSDS using a translation in space. Setting $\hat{u}(t) = u(t) - c(t)$ and

$\hat{v}(t) = v(t) - dc/dt$, then equations (65) and (66) can be expressed as (the argument t will not be explicitly written)

$$\mu\frac{d^2\hat{u}}{dt^2} + \zeta\frac{d\hat{u}}{dt} + \kappa\hat{u} = f_v - \left(\mu\frac{d^2c}{dt^2} + \zeta\frac{dc}{dt} + \kappa c\right) \equiv \quad \hat{f}, \quad if\ \hat{u} > 0 \tag{67}$$

$$\hat{v}^+ = \quad -r\hat{v}^-, \quad if\ \hat{u} = 0. \tag{68}$$

Using the framework presented in (26) we can derive all of the terms which define a Lagrangian NSDS. In our case the model is completely linear:

$$q = \left[\ \hat{u}\ \right], \quad M(q) = \left[\ \mu\ \right], \quad NNL(q,\dot{q}) = \left[\ 0\ \right] \tag{69}$$

$$F_{int}(t,q,\dot{q}) = \left[\ \zeta\ \right]\dot{q} + \left[\ \kappa\ \right]q, \quad F_{ext} = \left[\ \hat{f}\ \right] \tag{70}$$

The unilateral constraint requires that $\hat{u} \geqslant 0$ so we can obtain

$$y = H^T q + b, \quad H^T = \left[\ 1\ \right], \quad b = 0 \tag{71}$$

In the same way, the reaction force due to the constraint is written as follows:

$$R = H\lambda$$

The unilateral contact law may be formulated as in (52) with a complementarity condition and a Newton's impact law.

Simulation The simulation of the cam follower system has been performed for different values of the cam rotational speed with the SICONOS software package using a time-stepping numerical scheme with step size ($h = 1.10^{-4}$) and an event-driven scheme with minimum step size ($h_{min} = 1.10^{-12}$). Figure 24 and 25 show the time simulations for different values of the cam rotational speed and Figure 26 shows the chaotic attractor at $rpm = 660$ for impact and stroboscopic Poincaré sections.

10 Electrical Examples

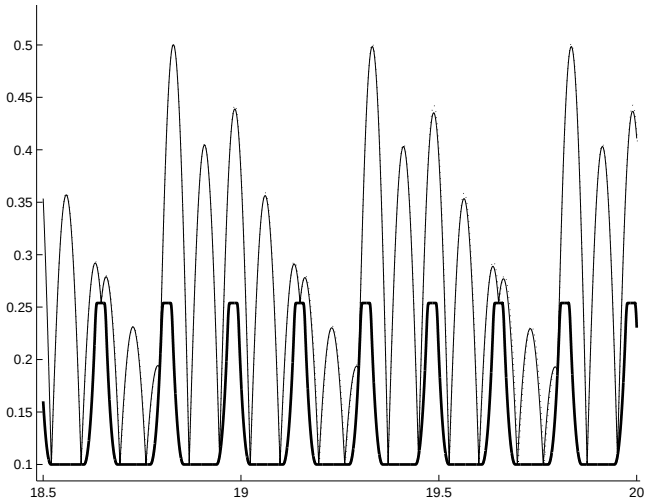
10.1 MOS Transistors and Inverters

Most of this work has been done by P. Denoyelle. More details can be found in [5]

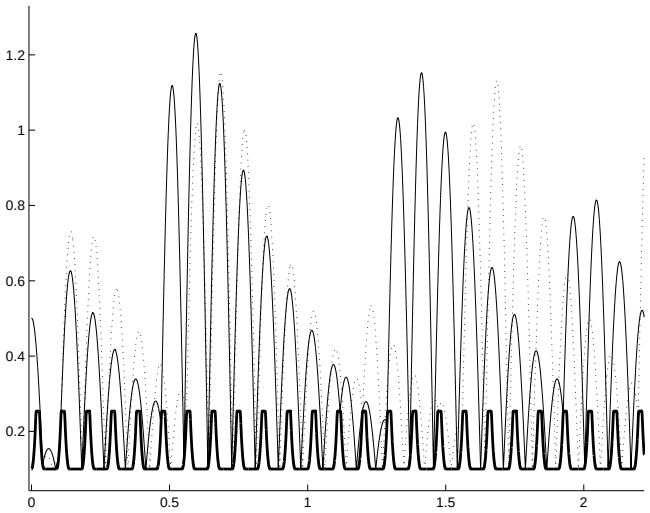
10.1.1 Piecewise Linear Model of a MOS Transistor

People could benefit from a simplification of devices models (e.g MOS models) in the form of a piecewise linear representation instead of the complicated formula implemented in SPICE simulators. For instance, in [9], the authors considered the Sah model of the NMOS static characteristic:

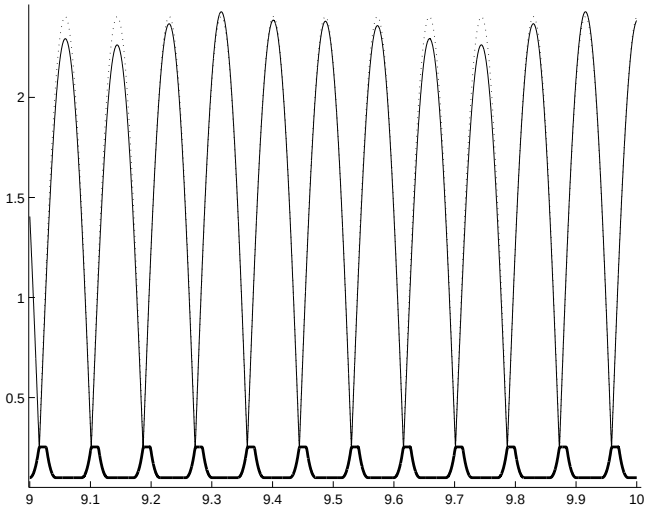
$$I_{DS} = \frac{K}{2} \cdot (f(V_G - V_S - V_T) - f(V_G - V_D - V_T))$$



(a)



(b)



(c)

Figure 24: The Cam Follower System. Time series using SICONOS platform. Time-stepping scheme (continuous line). Event-driven scheme (dashed line) (a) rpm=358. (b) rpm=660. (c) rpm=700.

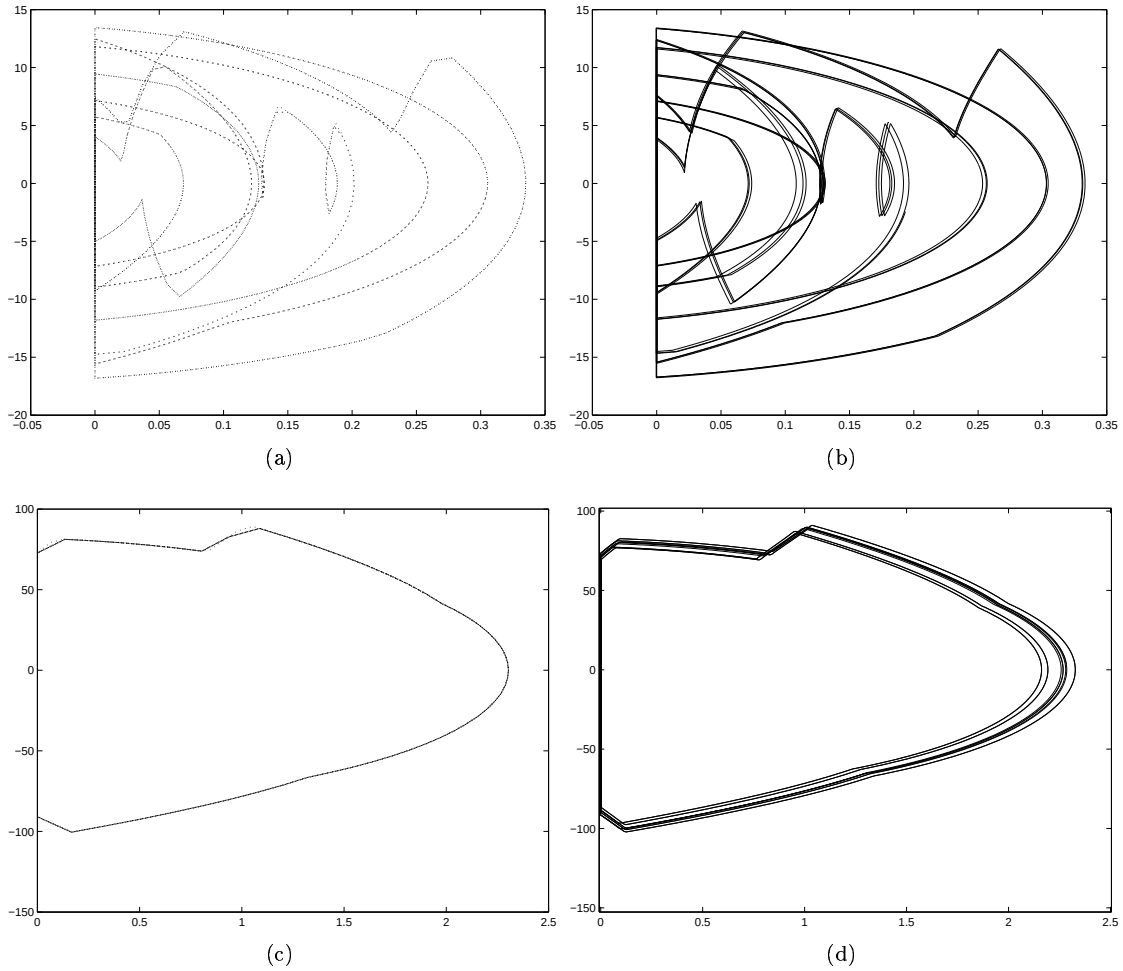


Figure 25: The Cam Follower System. State space comparison using SICONOS platform. (a) rpm=358. Event Driven (b) rpm=358. Time Stepping (h=1e-4) (c) rpm=700. Event Driven (d) rpm=700. Time Stepping (h=1e-4).

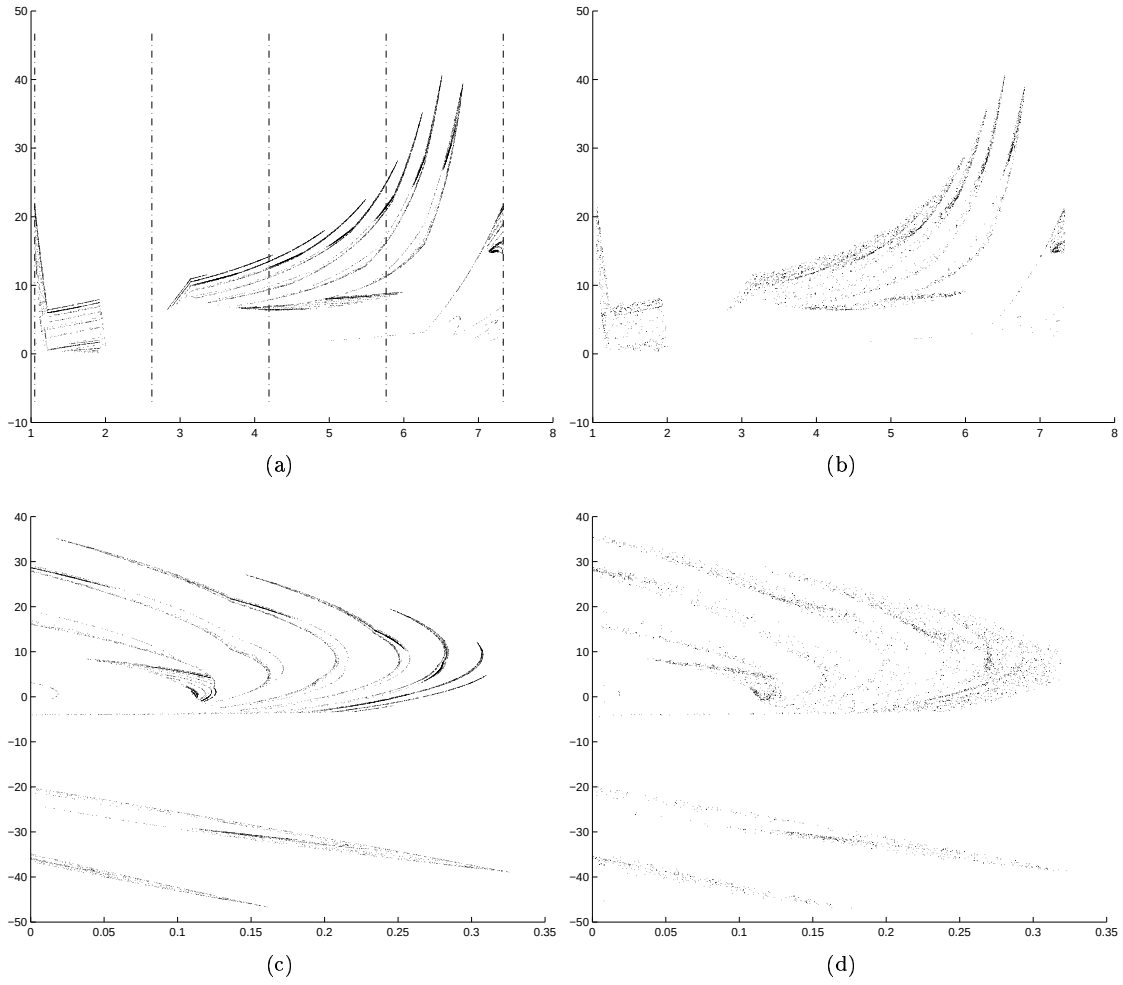


Figure 26: The Cam Follower System. Attractors comparison using SICONOS platform at rpm=660. (a) Impact map. (Event Driven) (b) Impact Map. Time Stepping (h=1e-4)(a) Stroboscopic map. (Event Driven) (b) Stroboscopic Map. Time Stepping (h=1e-4).

with:

$K = \dfrac{\mu C_{OX} W}{L}$

μ mobility of majority carriers

(sample values of 750 $cm^2.V^{-1}.s^{-1}$ for a NMOS, 250 $cm^2.V^{-1}.s^{-1}$ for a PMOS)

$C_{OX} = \dfrac{\epsilon_{SiO_2}}{t_{OX}}$

$\epsilon_{SiO_2} = \epsilon_{r\ SiO_2} \cdot \epsilon_0$ ($\epsilon_{r\ SiO_2} \approx 3.9$)

t_{OX} oxide thickness $\approx 4nm$ in a recent $180nm$ technology

W channel width

L channel length $\approx 130nm$ in a recent $180nm$ technology

V_T threshold voltage depending on technology, V_{BS} , temperature ≈ 0.25 to 1 V

The function $f : \mathbb{R} \longrightarrow \mathbb{R}$ is defined as:

$$f(x) = \begin{cases} 0 & \text{if } x < 0 \\ x^2 & \text{if } x \geqslant 0 \end{cases}$$

The piecewise and quadratic nature of this function was approximated by the following 6 segments piecewise linear function by the authors of [9] (see Figure 27):

$$f_{PWL}(x) = \begin{cases} 0 & \text{if } x < 0 \\ 0.09 \cdot x & \text{if } 0 \leqslant x < 0.1 \\ 0.314055 \cdot x - 0.0224055 & \text{if } 0.1 \leqslant x < 0.2487 \\ 0.780422 \cdot x - 0.138391 & \text{if } 0.2487 \leqslant x < 0.6185 \\ 1.94107 \cdot x - 0.856254 & \text{if } 0.6185 \leqslant x < 1.5383 \\ 4.82766 \cdot x - 5.29668 & \text{if } 1.5383 \leqslant x \end{cases}$$

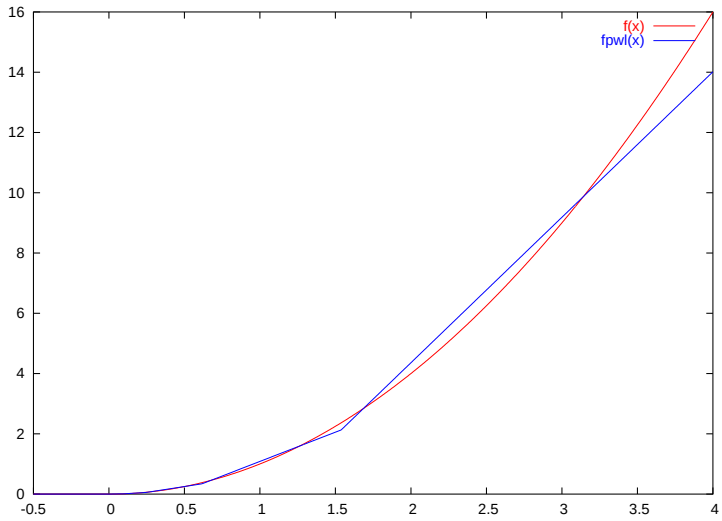


Figure 27: Piecewise linear approximation of f

The relative error between f and f_{PWL} is kept below 0.1 for $0.1 \leqslant x < 3.82$. The absolute error is less than $2 \cdot 10^{-3}$ for $0 \leqslant x < 0.1$ and 0 for negative x. In practice, the values of V_G, V_S, V_D, V_T in logic integrated circuits allow a good approximation of f by f_{PWL}.

The figure 28 displays the static characteristic $I_{DS}(V_{GS}, V_{DS})$ of an NMOS obtained with the SPICE level 1 model and the piecewise linear approximation of the Sah model. The following parameter values were used:

$$\epsilon_{r\ SiO_2} = 3.9$$
$$t_{OX} = 20\ nm$$
$$\mu = 750\ cm^2.V^{-1}.s^{-1}$$
$$W = 1\ \mu m$$
$$L = 1\ \mu m$$
$$V_T = 1\ V$$

Bottom figures include both models results with two different viewpoints to display the regions where differences appear.

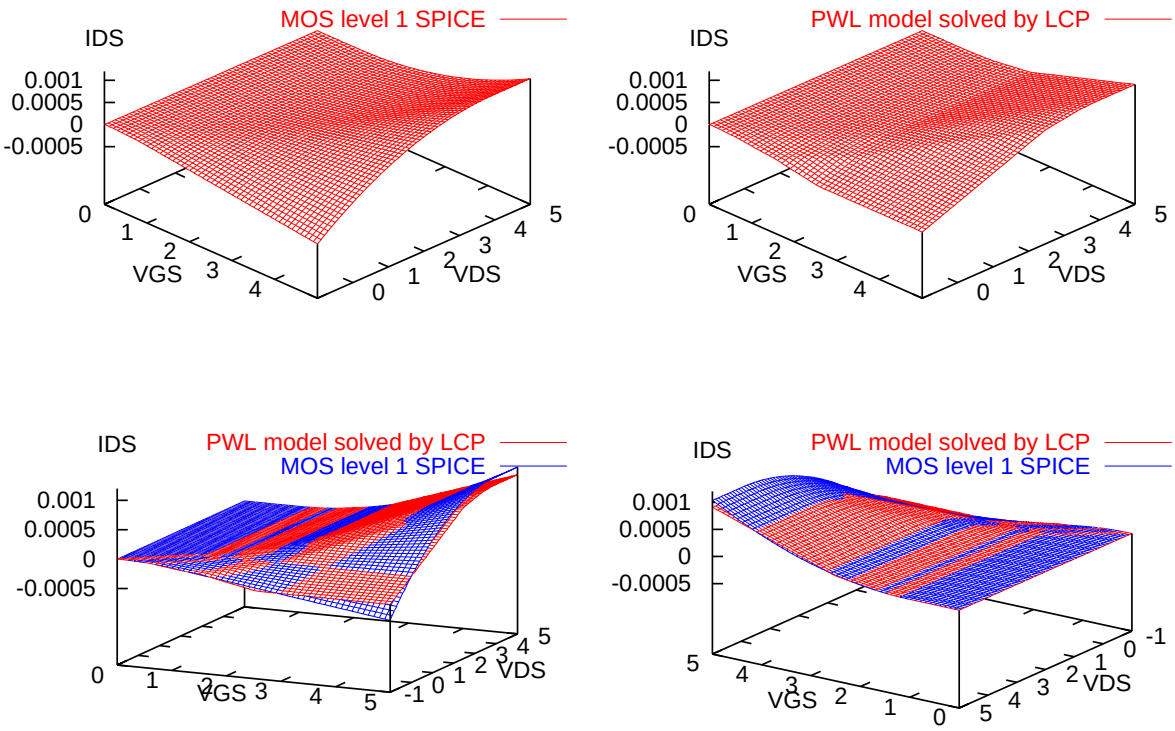


Figure 28: Static characteristic of an NMOS transistor with a simple PWL model and SPICE level 1 model

10.1.2 Inverter Chain

This simple model of a NMOS transistor was adapted to the PMOS transistor and both models were used to simulate an inverter chain (see Figure 29). The output of each inverter is loaded by the intrinsic capacitances of transistors (with values of a few fF) and a load capacitor of 50 fF representing the wiring between successive inverters.

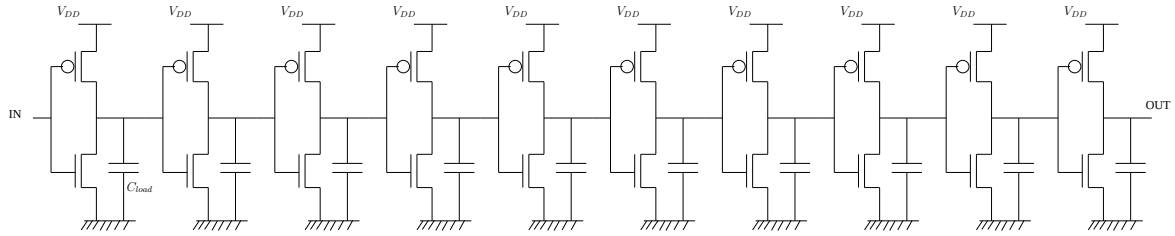


Figure 29: Inverter chain in CMOS

In these early simulations, the dynamical behavior of the MOS transistor was simplified by keeping the intrinsic capacitances C_{GS} and C_{GD} independent from voltages. Of course, this differs from the Meyer nonlinear capacitances implemented in the SPICE level 1 model. Figure 30 shows the comparison between simulation results with SPICE (dotted lines) and SICONOS for a selection of inverters output voltage and MOS currents.

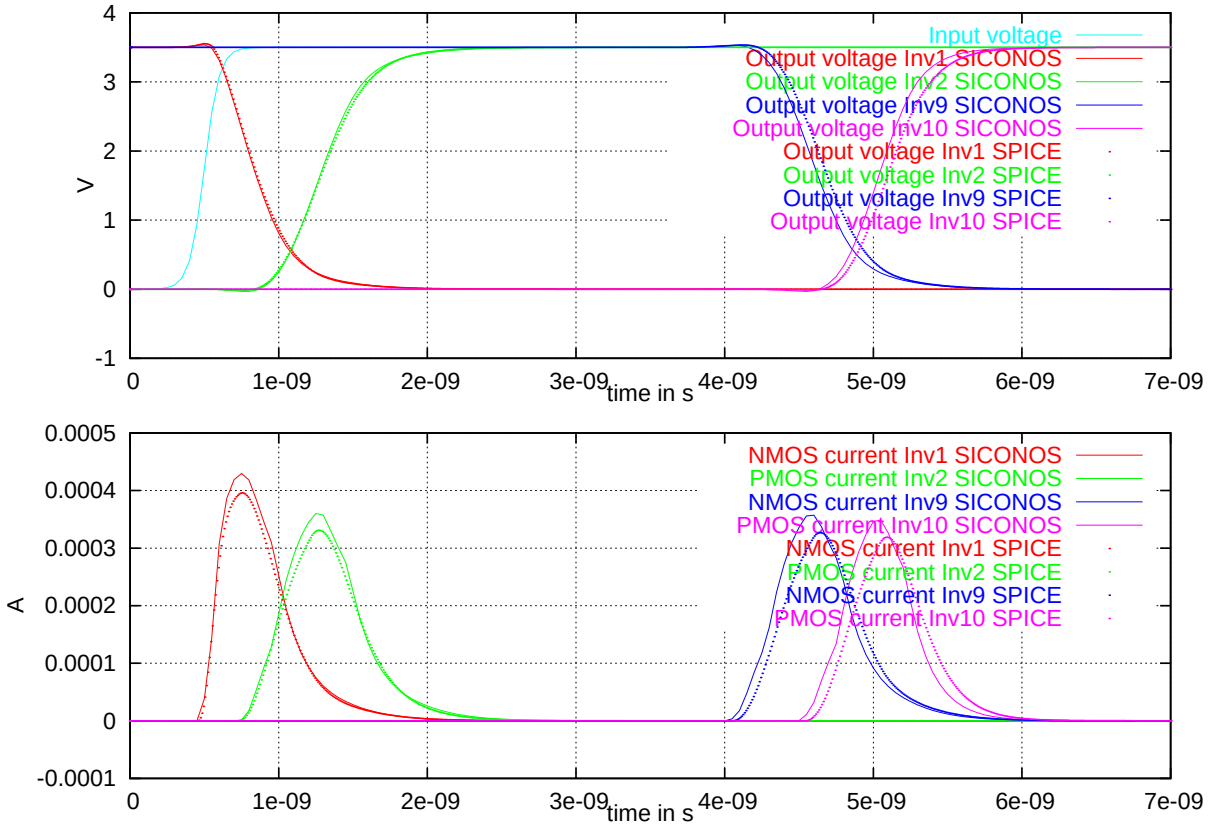


Figure 30: Simulation of a 10 inverters chain with SICONOS and SPICE level 1 model

10.2 Power Converters

The functioning of switch mode power supplies is based on flyback diodes and switched transistors with a regulation on output current or voltage. The article [1] presents some simulation results of a Cuk converter [8] which have been reproduced in Siconos.

This example has been proposed and developed by I. Merillas, E. Fossas and Carles Battle from Universitat Politècnica de Catalunya (UPC), Barcelona [11]. The system consists in a Parallel Resonant Converter (PRC) which is basically a dc-dc power converter. The schematic of the PRC is shown in Figure 31. The system is composed of four parts: an inverter block, a resonant tank in series, a rectifier block and an output filter. In our case, the inverter block is a full-bridge inverter. It is called parallel resonant converter because the load is in parallel with the resonant capacitor.

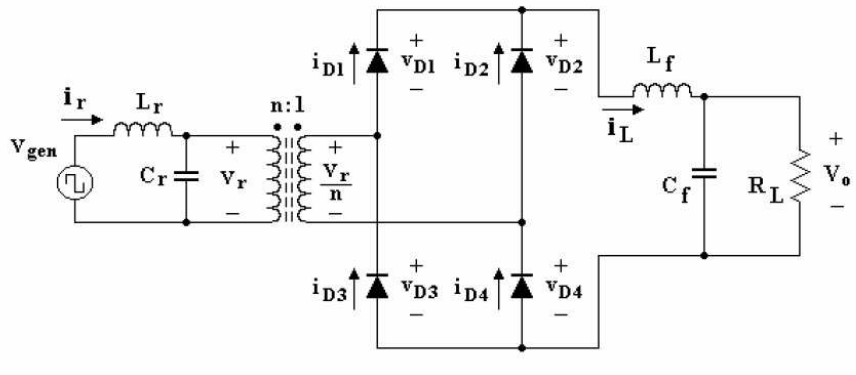


Figure 31: A Parallel Resonant Converter diagram

To represent the system in Siconos, we use a `FirstOrderLinearDS`, a `FirstOrderLinearTIR` and a `ComplementarityConditionNSL`, as given in (24), (34) and (51) respectively.

An thus we obtain a LCS (Linear Complementarity System) of the form:

$$\dot{x}(t) = Ax(t) + b(t) + r, \tag{72}$$

$$y(t) = Cx(t) + D\lambda(t), \tag{73}$$

$$0 \leqslant y \perp \lambda \geqslant 0 \tag{74}$$

with the state $x = [i_r, v_r, i_L, v_0]$, and the complementarity variables $y = [v_{D1}, v_{D3}, i_{D2}, i_{D4}]$ and $\lambda = [i_{D1}, i_{D3}, v_{D2}, v_{D4}]$. And:

$$A = \begin{pmatrix} 0 & -\frac{1}{L_r} & 0 & 0 \\ \frac{1}{C_r} & 0 & 0 & 0 \\ 0 & 0 & 0 & -\frac{1}{L_f} \\ 0 & 0 & \frac{1}{C_f} & -\frac{1}{R_L C_f} \end{pmatrix}, \quad B = \begin{pmatrix} 0 & 0 & 0 & 0 \\ -\frac{1}{nC_r} & \frac{1}{nC_r} & 0 & 0 \\ 0 & 0 & \frac{1}{L_f} & \frac{1}{L_f} \\ 0 & 0 & 0 & 0 \end{pmatrix}, \tag{75}$$

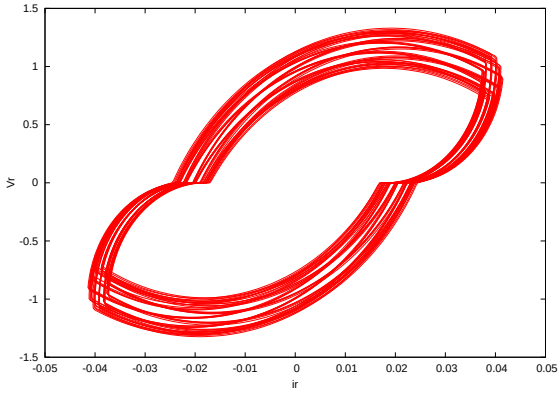
$$C = \begin{pmatrix} 0 & -\frac{1}{n} & 0 & 0 \\ 0 & \frac{1}{n} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}, D = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \end{pmatrix}, \tag{76}$$

[8] This work was supported by the European project SICONOS IST-2001-37172.

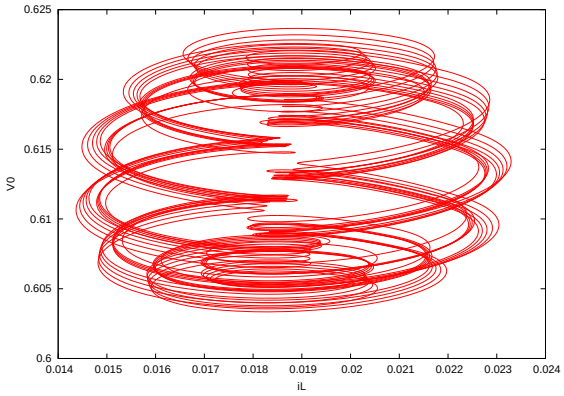
And for the external source:

$$b(t) = E\,\mathrm{Sign}(sin(wt)), E = \begin{pmatrix} \frac{1}{L_r} \\ 0 \\ 0 \\ 0 \end{pmatrix} \tag{77}$$

The numerical simulations are performed with the following parameter values: $L_r = 150\mu H$, $L_f = 0.4mH$, $C_r = 68nF$, $C_f = 2.2\mu F$, $R_L = 33\Omega$ and the frequency of the input voltage $F_0 = 55000Hz$. The results are given on Figure 32.



(a) V_r according to i_R



(b) V_0 according to i_L

Figure 32: Siconos simulation of the parallel resonant converter, with $R_L = 33\Omega$ and $F0 = 55KHz$.

Acknowledgments

This work is supported by the European Community through the Information Society Technologies thematic programme under the project SICONOS (IST-2001-37172).

The authors acknowledge the Siconos partners who have contributed the examples Section of this document, especially, Mathias Moeller for his contribution to the woodpecker toy, Mario di Bernardo, Gustavo Osorio and Stefania Santini for their contribution to the Cam-follower system, P. denoyelle for the Mos transistor modelling and I. Merillas for the power converter simulation. Finally, The author want also to thank F. Dubois and R. Pissard-Gibollet for their great help in the design and the implementation of the Siconos platform.

References

[1] C. Batlle, E. Fossas, I. Merillas, and A. Miralles. Generalized discontinuous conduction modes in the complementarity formalism. *Circuits and Systems II: Express Briefs, IEEE Transactions on*, 52, no.8:447–451, Aug. 2005.

[2] B. Brogliato. *Nonsmooth Mechanics.* Springer Verlag London, second edition, 1999.

[3] F.H. Clarke. *Optimization and Nonsmooth analysis.* Wiley, New York, 1983.

[4] R. W. Cottle, J. Pang, and R. E. Stone. *The linear complementarity problem.* Academic Press, Inc., Boston, MA, 1992.

[5] P. Denoyelle and V. Acary. The non-smooth approach applied to simulating integrated circuits and power electronics. evolution of electronic circuit simulators towards fast-SPICE performance. *INRIA Research Report 0321, http://hal.inria.fr/docs/00/08/09/20/PDF/RT-0321.pdf*, 2006.

[6] H. Elmqvist, S.E. Mattsson, and M. and Otter. Object-oriented and hybrid modeling in modelica. *Journal Européen des systèmes automatisés*, 35(4):395 – 404, 2001.

[7] C. Glocker. *Set-Valued Force Laws: Dynamics of Non-Smooth systems*, volume 1 of *Lecture notes in applied mechanics.* Spring Verlag, 2001.

[8] W.P.M.H. Heemels, J.M. Schumacher, and S. Weiland. Linear complementarity problems. *S.I.A.M. Journal of applied mathematics*, 60(4):1234–1269, 2000.

[9] Domine M. W. Leenaerts and Wim M. Van Bokhoven. *Piecewise Linear Modeling and Analysis.* Kluwer Academic Publishers, Norwell, MA, USA, 1998.

[10] Ch. Leine, R. Glocker and D.H. Van Campen. Nonlinear dynamics and modelling of various wooden toys with impact and friction. *Journal of Vibration and Control,*, 9:25–78, 2001.

[11] I. Merillas Santos. *Modeling and Numerical Study of Nonsmooth Dynamical systems.* PhD thesis, Universitat Politècnica de Catalunya, December 2006.

[12] M. D. P. Monteiro Marques. *Differential Inclusions in NonSmooth Mechanical Problems : Shocks and Dry Friction.* Birkhauser, Verlag, 1993.

[13] J.J. Moreau. Unilateral contact and dry friction in finite freedom dynamics. In J.J. Moreau and Panagiotopoulos P.D., editors, *Nonsmooth mechanics and applications*, number 302 in CISM, Courses and lectures, pages 1–82. CISM 302, Spinger Verlag, 1988. Formulation mathematiques tire du livre Contacts mechanics.

[14] J.S. Pang and R. Chandrasekaran. Linear complementarity problems solvable by a polynomially bounded pivoting algorithms. *Math. Prog. Study*, 25:13–27, 1985.

[15] F. Pfeiffer and C. Glocker. *Multibody Dynamics with Unilateral Contacts.* Non-linear Dynamics. John Wiley & Sons, 1996.

[16] S.M. Robinson. Generalized equations and their solutions. I. Basic theory. *Mathematical programming study*, 10:128–141, 1979.

[17] M. Schatzman. A class of nonlinear differential equations of second order in time. *Nonlinear Analysis, Theory, Methods & Applications*, 2(3):355–373, 1978.

[18] A. J. van der Schaft and J.M. Schumacher. *An Introduction to Hybrid Dynamical Systems.* Springer Verlag London, 2000.

Unité de recherche INRIA Rhône-Alpes
655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier (France)

Unité de recherche INRIA Futurs : Parc Club Orsay Université - ZAC des Vignes
4, rue Jacques Monod - 91893 ORSAY Cedex (France)
Unité de recherche INRIA Lorraine : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)
Unité de recherche INRIA Sophia Antipolis : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-0803