Faculté des Sciences et Techniques de Limoges **Année 2018-2019**

Master ACSYON
Practical session: Nonsmooth Dynamical Systems Simulation[1]

Simulation of the half–wave rectifier.

Vincent Acary, INRIA `vincent.acary@inria.fr`

1 Description of the physical problem

Let us consider the circuit depicted in Figure 1 involving one diode (supposed to be ideal), a resistor with the resistance $R > 0$, a capacitor with the capacitance $C > 0$ and an inductor with the inductance $L > 0$. This circuit is known as the half wave rectifier and allows unidirectional current through the load during the one–half input cycle. The full-wave rectifier lets the positive signal through and cancels the negative signal (see Figure 2).

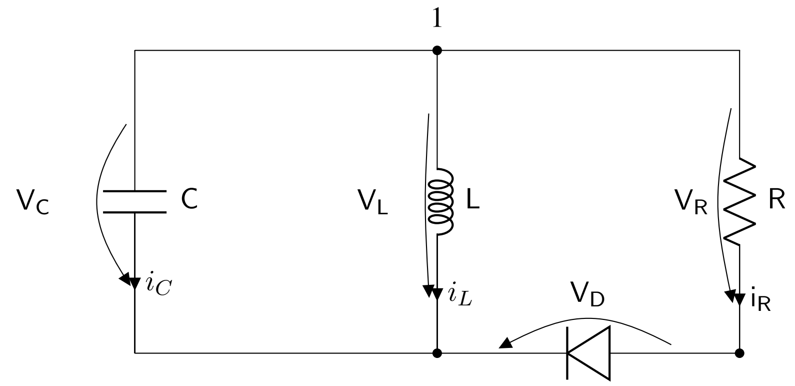

Figure 1: The half-wave rectifier. LC oscillator with a load resistor

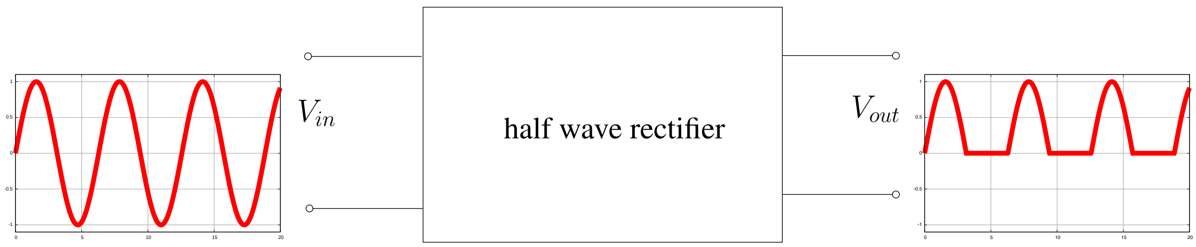

Figure 2: Input and output of half-wave rectification

2 Linear Complementarity Systems (LCS)

Given matrices $A \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{n\times m}$, $C \in \mathbb{R}^{m\times n}$ and $D \in \mathbb{R}^{m\times m}$, a Linear Complementarity System, denoted by $LCS(A, B, C, D)$, is a problem of finding a state trajectory $t \mapsto \mathbf{x}(t) \in \mathbb{R}^n$ and a

[1] contact: vincent.acary@inria.fr

input $t \mapsto \lambda(t) \in \mathbb{R}^m$ such that

$$LCS(A,B,C,D) \begin{cases} \dot{\mathbf{x}}(t) = A\mathbf{x}(t) + B\lambda(t), \\ \mathbf{y}(t) = C\mathbf{x}(t) + D\lambda(t), \\ 0 \le \mathbf{y}(t) \perp \lambda(t) \ge 0, \\ \mathbf{x}(0) = \mathbf{x}_0. \end{cases}$$

This kind of problems appear in many applied fields and particularly in circuit analysis and mechanical systems with unilateral constraints. We will show that the circuit depicted in Figure 1 can be formulated as a LCS.

3 Modeling the circuit as a LCS

We recall the following Kirchhoff's laws and I-V characteristic for an ideal diode:

- Kirchhoff's voltage law (KVL): the algebraic sum of the voltages between successive nodes in all meshes in the circuit is zero.
- Kirchhoff's current law (KCL): the algebraic sum of the currents in all branches which converge to a common node equals to zero.
- Each electrical device, is characterized by its ampere-volt characteristic. For example, Figure 3 illustrates the I-V characteristic of an ideal diode. This is a model in which the diode is a simple switch. If $V < 0$ then $i = 0$ and the diode is blocking. If $i > 0$ then $V = 0$ and the diode is conducting. It is easy to see that the ideal diode is described by the complementarity relation

$$V \le 0, \quad i \ge 0, \quad \text{and} \quad Vi = 0.$$

- The constitutive equations for linear devices like capacitors, inductors and resistors are given by:

$$i_c = C\frac{dv_C}{dt}, \quad v_L = L\frac{di_L}{dt}, \quad \text{and} \quad v_R = Ri_R. \tag{1}$$

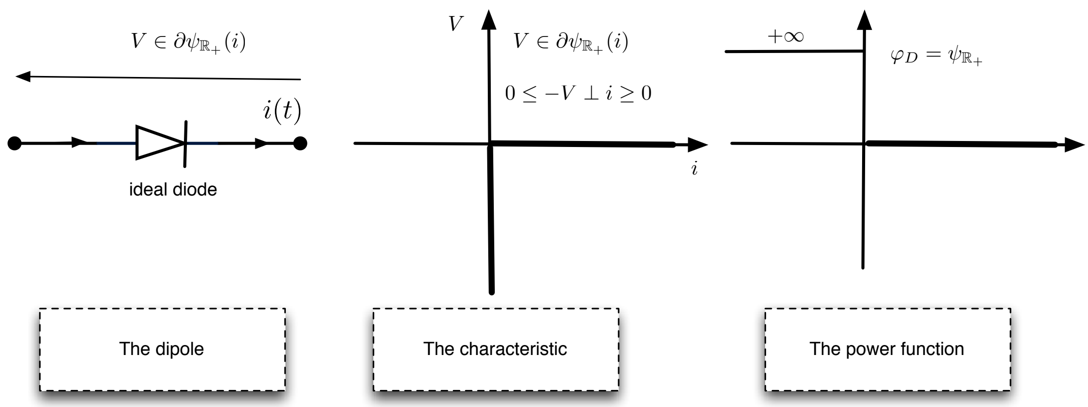

Figure 3: I-V characteristic of an ideal diode

Show that the Kirchhoff's laws can be written as:

$$\begin{cases} v_L = v_C = V_D + V_R \\ i_R + i_C + i_L = 0 \end{cases}$$

We set $x = \begin{pmatrix} v_C \\ i_L \end{pmatrix}$.

Question 1 : Show that the Kirchhoff's laws together with (1) can be rewritten as a LCS with

$$A = \begin{pmatrix} -1/(RC) & -1/C \\ 1/L & 0 \end{pmatrix},\ B = \begin{pmatrix} -1/(RC) \\ 0 \end{pmatrix},\ \lambda = \begin{pmatrix} -v_D \end{pmatrix},\ C = \begin{pmatrix} 1/R & 0 \end{pmatrix},$$

$$D = \begin{pmatrix} 1/R \end{pmatrix}.$$

4 Description of the numerical method: Moreau's time-stepping scheme

Let $t_0 = 0 < t_1 < t_2 < \ldots < t_N = T$ be a finite subdivision of the time interval $[0, T]$ with $T > 0$. We supposed that the length of the time step is $h = t_{k+1} - t_k = \frac{T}{N}$. Given $k = 0, 1, \ldots, N$, we define $\mathbf{x}_k = \mathbf{x}(t_k)$, $\mathbf{y}_k = \mathbf{y}(t_k)$, and $\mathbf{u}_k = \mathbf{u}(t_k)$. The following time-stepping scheme is used to solve LCS(A, B, C, D):

$$\begin{cases} \dfrac{\mathbf{x}_{k+1} - \mathbf{x}_k}{h} = A\mathbf{x}_{k+\theta} + B\lambda_{k+1}, \\ \mathbf{y}_{k+1} = C\mathbf{x}_{k+1} + D\lambda_{k+1} \\ 0 \leq \mathbf{y}_{k+1} \perp \lambda_{k+1} \geq 0, \end{cases}$$

where $\theta \in [0, 1]$ and $\mathbf{x}_{k+\theta} = \theta\mathbf{x}_{k+1} + (1-\theta)\mathbf{x}_k$.

We will suppose that the matrix $(I_n - h\theta A)$ is non-singular and we set

$$W = (I_n - h\theta A)^{-1} \text{ and } \tilde{\mathbf{x}}_k = W\Big(I_n + h(1-\theta)A\Big)\mathbf{x}_k.$$

Show that the scheme below is equivalent to solve, for each $k = 0, 1, \ldots, N$, the following linear complementarity $[\mathbf{y}_{k+1}, \lambda_{k+1}] = LCP(M, \mathbf{q}_k)$ with

$$M = hCWB + D \text{ and } \mathbf{q}_k = C\tilde{\mathbf{x}}_k.$$

We compute the new state

$$\mathbf{x}_{k+1} = \tilde{\mathbf{x}}_k + hWB\lambda_{k+1}.$$

5 Numerical simulation using SICONOS

The general documentation of Siconos can be found here:
https://nonsmooth.gricad-pages.univ-grenoble-alpes.fr/siconos/getting_started/index.html
In this tutorial class, the Python Front-End to Siconos will be used. For a tutorial on Python, we refer to http://docs.python.org/2/tutorial/. Only the syntax in Python differs from the C++ interface but the signature (name of the functions and arguments) remains the same.
For the classes and the methods of Siconos/kernel pyhton API, the documentation can be found here:
https://nonsmooth.gricad-pages.univ-grenoble-alpes.fr/siconos/reference/python_api.html

5.1 Python environment and interpreter.

Open with your favorite text editor (vi, emacs, gedit, ...) a file named `CircuitRLCD.py`. In order to load some Siconos modules into your Python script file, the header of `CircuitRLCD.py` must contain for this class.

```
from siconos.kernel import FirstOrderLinearDS, FirstOrderLinearTIR,  \
                           ComplementarityConditionNSL, Interaction, \
                           NonSmoothDynamicalSystem, EulerMoreauOSI, TimeDiscretisation, LCP,  \
                           TimeStepping
```

Enter `ipython` to launch the enhanced Interactive Python interpreter, add the path to the installation of Siconos and execute your script file :

```
[acary@saturne] scl enable python27 bash
[acary@saturne]$ ipython2
Python 2.7.13 (default, Apr 12 2017, 06:53:51)
Type "copyright", "credits" or "license" for more information.

IPython 5.1.0 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.

In [1]: import sys

In [2]: sys.path.append('/usr/local/lib/python2.7/site-packages/')

In [3]: %run 'CircuitRLCD.py'
```

5.2 Dynamical System

The class FirstOrderLinearDS that inherits from DynamicalSystem represents first order linear systems of the form:

$$M\dot{x}(t) = A(t)x(t) + b(t) + r$$

$$x(t_0) = x_0$$

where

1. x is the state, x_0 the initial condition
2. r the input due to the nonsmooth Interaction .
3. M is an optional constant matrix (not necessarily full rank). By default, the matrix M is the identity matrix.

The complete description of the class FirstOrderLinearDS can be obtained by invoking the help command

```
In [4]: help(FirstOrderLinearDS)
Help on class FirstOrderLinearDS in module siconos.kernel:

class FirstOrderLinearDS(FirstOrderNonLinearDS)
 |  Proxy of C++ FirstOrderLinearDS class
 |
 |  Method resolution order:
 |      FirstOrderLinearDS
 |      FirstOrderNonLinearDS
 |      DynamicalSystem
 |      __builtin__.object
 |
 |  Methods defined here:
 |
...
```

The following code creates an instance of FirstOrderLinearDS .

```
# initial voltage
Vinit = 10.0

# Define the initial state
init_state = [Vinit, 0]

# Define the matrix A
Lvalue = 1e-2     # inductance
Cvalue = 1e-6     # capacitance
Rvalue = 5e+2      # resistance

A = [[-1.0/(Rvalue*Cvalue), -1.0/Cvalue],
     [1.0/Lvalue, 0           ]]

# call of the constructor method of FirstOrderLinearDS
LSCircuitRLCD = FirstOrderLinearDS(init_state, A)
```

5.3 Interactions

5.3.1 Relations

The linear time invariant relation is defined by the class FirstOrderLinearTIR that inherits from Relation and defines a linear relation for first order dynamical systems:

$$y = Cx + Fz + D\lambda + e$$

$$r = B\lambda$$

```
C = [[1./Rvalue,   0.]]
B = [[ -1./(Rvalue*Cvalue)], [0.]]
D = [[1.0/Rvalue]]
# call of the constructor method of FirstOrderLinearTIR
LTIRCircuitRLCD = FirstOrderLinearTIR(C, B)
# set the matrix of the relation by invoking the setDPtr method
LTIRCircuitRLCD.setDPtr(D)
```

5.3.2 ComplementarityConditionNSL

```
# dimension of the nonsmooth law
m = 1
# call of the constructor method of ComplementarityConditionNSL
nslaw=ComplementarityConditionNSL(m)
```

The class ComplementarityConditionNSL defines a complementarity condition between y and λ of $\mathbb{R}^n$

$$0 \leq y \perp \lambda \geq 0$$

5.3.3 Interaction

The class Interaction collects the nonsmooth law and the relation. An Interaction describes the non-smooth interactions between some Dynamical Systems.

```
# call of the constructor method of ComplementarityConditionNSL
InterCircuitRLCD = Interaction(nslaw, LTIRCircuitRLCD)
```

It represents the link between a set of Dynamical Systems that interact through some relations (between state variables (x, r) and local variables (y, λ) completed by a nonsmooth law. By invoking the help on the Interaction class:

```
In [5]: help(Interaction)
Help on class Interaction in module siconos.kernel:

class Interaction(__builtin__.object)
 |  Proxy of C++ Interaction class
 |
 |  Methods defined here:
...
 |  __init__(self, *args)
 |      __init__(Interaction self) -> Interaction
 |      __init__(Interaction self, SP::InteractionXML arg2) -> Interaction
 |      __init__(Interaction self, unsigned int interactionSize, SP::NonSmoothLaw NSL,
SP::Relation rel, unsigned int number=0) -> Interaction
 |      __init__(Interaction self, unsigned int interactionSize, SP::NonSmoothLaw NSL,
 SP::Relation rel) -> Interaction
```

we get some constructor where the arguments are :

- SP::NonSmoothLaw NSL : a pointer to the non smooth law
- SP::Relation ref : a pointer to the Relation
- int (optional) : the number of this Interaction (default 0)

5.4 NonSmoothDynamicalSystem

In this section, we build the complete model by creating an instance of the NonSmoothDynamicalSystem and by inserting the dynamical systems that have been previously created into a nonSmoothDynamicalSystem and by linking Interaction instances with the DynamicalSystem instances.

```
t0 = 0.0          # initial time
T = 5.0e-3        # Total simulation time
#   call the constructor method of the model
Modeltitle = 'CircuitRLCD' # optional name of the model
CircuitRLCD = NonSmoothDynamicalSystem(t0, T)
CircuitRLCD.setTitle(Modeltitle)

#   add the dynamical system in the non smooth dynamical system
CircuitRLCD.nonSmoothDynamicalSystem().insertDynamicalSystem(LSCircuitRLCD)
#   link the interaction and the dynamical system
CircuitRLCD.nonSmoothDynamicalSystem().link(InterCircuitRLCD, LSCircuitRLCD)
```

5.5 Simulation

In this section, we create a Simulation instance, which is composed of a TimeDiscretisation , a OneStepIntegrator and a OneStepNSProblem .

5.5.1 Time discretisation

The TimeDiscretisation defines the Time–discretization.

```
h_step = 1.0e-5   # Time step
aTiDisc = TimeDiscretisation(t0,h_step)
```

5.5.2 One step integrator

The OneStepIntegrator defines the time integration scheme. In our example, we choose the Moreau scheme.

```
theta = 0.5
aOSI = EulerMoreauOSI(theta)
```

5.5.3 One step Non smooth problem

The OneStepNSProblem defines the one step nonsmooth problem that is formulated at each time–step. In our example, we choose a LCP.

```
solver_number = 200
aLCP = LCP(solver_number)
```

For a list of solvers available in Siconos/Numerics, we can have a look to the documentation of the C++ code :

```
enum LCP_SOLVER {
  SICONOS_LCP_LEMKE=200,          SICONOS_LCP_NSGS_SBM=201,   SICONOS_LCP_PGS=202,
  SICONOS_LCP_CPG =203,           SICONOS_LCP_LATIN =204,     SICONOS_LCP_LATIN_W =205,
  SICONOS_LCP_QP =206,            SICONOS_LCP_NSQP =207,      SICONOS_LCP_NEWTONMIN =208,
  SICONOS_LCP_NEWTONFB =209,      SICONOS_LCP_PSOR =210,      SICONOS_LCP_RPGS =211,
  SICONOS_LCP_PATH=212,           SICONOS_LCP_ENUM =213
};
```

5.5.4 Simulation

Finally, we complete the Simulation instance by invoking its constructor with the previous objects. In our case, we choose a TimeStepping type of simulation.

```
# call the constructor method of the class TimeStepping
aTS = TimeStepping(CircuitRLCD, aTiDisc,aOSI,aLCP)
```

The parameters of the constructor are

- pointer to a timeDiscretisation
- one step integrator (default none)
- one step non smooth problem (default none)

5.6 Launch a computation

The following step are necessary to launch a simulation

1. **Compute one step**

```
# call the computeOneStep method of the simulation
aTS.computeOneStep()
```

2. **Advance to the next time step**

```
# call the nextStep method of the simulation
aTS.nextStep()
```

   This method will increment the model current time according to user TimeDiscretisation and call SaveInMemory.

The time–step can be obtained by

```
#get the current time step size ("next time"-"current time")
h = aTS.timeStep();
```

At the end of this first time step, you can have access to the state of the Dynamical system and the variable of the interaction.

```
# get a pointer on the state of the system
x = LSCircuitRLCD.x()
print 'state =', x
# get a pointer on the ith derivative of y
i = 0
y = InterCircuitRLCD.y(i)
print 'y =', y
# get a pointer on the ith level of lambda
lambda_ = InterCircuitRLCD.lambda_(i)
print 'lambda =', lambda_
```

Question 2 : Build a loop that will run the simulation for N steps from t_0 to T.

6 Post-processing and displaying the result

Question 3 : In the previous loop, store at each time the result of interest into a matrix of the form

```
from numpy import zeros
dataPlot = zeros((N, number_of_ouput))
```

Question 4 : Draw a figure with the inductor voltage, inductor current and the current and voltage in the diode w.r.t time.

```
from matplotlib.pyplot import subplot, title, plot, grid, show

subplot(411)
title('inductor voltage')
plot(dataPlot[0:k-1,0], dataPlot[0:k-1,1])
grid()
show()
```

7 Study of the order of the method.

Question 5 : Give an analytic solution of the state of the system and the current trough the diode in the half wave rectifier ?

Hint : positive part of an damped oscillator

Question 6 : Perform the simulation with various time–step ranging from 10^{-3} to 10^{-6} and compute the error we respect to the analytic solution. Plot in log scale the error w.r.t the time step.