Developer's Notes

Siconos Development Team

November 4, 2009

Chapter 1

First Order Nonlinear Relation

author	0. Bonnefon
date	July, 1 2009
version	Kernel 3.0.0

Chapter 2

OneStepNSProblem formalisation for several interactions

author	F. Pérignon
date	May 16, 2006
version	?

2.1 LinearDS - Linear Time Invariant Relations

2.1.1 General notations

We consider n dynamical systems of the form:

$$\dot{x}_i = A_i x_i + R_i \tag{2.1}$$

Each system if of dimension n_i, and we denote $N = \sum_{i=1}^{n} n_i$.
An interaction, I_α is composed with a non smooth law, $nslaw_\alpha$ and a relation:

$$y_\alpha = C_\alpha X_\alpha + D_\alpha \lambda_\alpha \tag{2.2}$$

The "dimension" of the interaction, ie the size of vector y_α, is denoted m_α and we set:

$$M = \sum_{\alpha=1}^{m} m_\alpha$$

m being the number of interactions in the Non Smooth Dynamical System.
X_α is a vector that represents the DS concerned by the interaction. Its dimension is noted N_α, this for n_α systems in the interaction.
C_α is a $m_\alpha \times N_\alpha$ row-blocks matrix and D_α a $m_\alpha \times m_\alpha$ square matrix.

$$C_\alpha = \begin{bmatrix} C_\alpha^i & C_\alpha^j & \dots \end{bmatrix} \tag{2.3}$$

with $i, j, \dots \in \mathcal{DS}_\alpha$ which is the set of DS belonging to interaction α.
We also have the following relation:

$$\begin{bmatrix} R_\alpha^i \\ R_\alpha^j \\ \dots \end{bmatrix} = B_\alpha \lambda_\alpha = \begin{bmatrix} B_\alpha^i \\ B_\alpha^j \\ \dots \end{bmatrix} \lambda_\alpha \tag{2.4}$$

R^i_α represents the contribution of interaction α on the reaction of the dynamical system i, and B^i_α is a $n_i \times m_\alpha$ block matrix.
And so:

$$R_i = \sum_{\beta \in \mathcal{I}_i} R^i_\beta = \sum_{\beta \in \mathcal{I}_i} B^i_\beta \lambda_\beta \tag{2.5}$$

with $\mathcal{I}_i$ the set of interactions in which dynamical system number i is involved.
Introducing the time dicretisation, we get:

$$x^{k+1}_i - x^k_i = hA_i x^{k+1}_i + hR^{k+1}_i \tag{2.6}$$

$$y^{k+1}_\alpha = C_\alpha X^{k+1}_\alpha + D_\alpha \lambda^{k+1}_\alpha \tag{2.7}$$

$$R^{k+1}_i = \sum_{\beta \in \mathcal{I}_i} B^i_\beta \lambda^{k+1}_\beta \tag{2.8}$$

ie, with $W_i = (I - hA_i)^{-1}$:

$$x^{k+1}_i = W_i x^k_i + hW_i R^{k+1}_i \tag{2.9}$$

$$y^{k+1}_\alpha = C_\alpha W_\alpha X^k_\alpha + C_\alpha h W_\alpha \sum_{\beta \in \mathcal{I}_i} B^i_\beta \lambda^{k+1}_\beta + D_\alpha \lambda^{k+1}_\alpha \tag{2.10}$$

$$= C_\alpha W_\alpha X^k_\alpha + (C_\alpha h W_\alpha B_\alpha + D_\alpha)\lambda^{k+1}_\alpha + \sum_{\beta \neq \alpha} \Big(\sum_{i \in \mathcal{DS}_\alpha \cap \in \mathcal{DS}_\beta} h C^i_\alpha W_i B^i_\beta \lambda^{k+1}_\beta \Big) \tag{2.11}$$

with

$$W_\alpha = \begin{bmatrix} W_i & 0 & \dots \\ 0 & W_j & \dots \\ 0 & \dots & \dots \end{bmatrix} \tag{2.12}$$

the block-diagonal matrix of all the W for the dynamical systems involved in interaction α.
The global-assembled Y vector, of dimension M, composed by m y_α subvectors, is given by:

$$Y_{k+1} = q_{OSNSP} + M_{OSNSP} \Lambda_{k+1} \tag{2.13}$$

or,

$$\begin{aligned} Y_{k+1} = \begin{bmatrix} y_1 \\ \dots \\ y_m \end{bmatrix}_{k+1} &= \begin{bmatrix} C^1_1 & \dots & C^n_1 \\ \vdots & \dots & \vdots \\ C^1_m & \dots & C^n_m \end{bmatrix} \begin{bmatrix} W_1 & 0 & \dots & 0 \\ 0 & W_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \vdots \\ & & 0 & W_n \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ \vdots \\ x_n \end{bmatrix}_k \\ &+ \begin{bmatrix} D_1 + h\sum_{j \in \mathcal{DS}_1} C^j_1 W_j B^j_1 & h \sum_{j \in \mathcal{DS}_1 \cap \mathcal{DS}_2} C^j_1 W_j B^j_2 & \dots \\ \vdots & \ddots & \\ & h \sum_{j \in \mathcal{DS}_m} C^j_m W_j B^j_{m-1} & D_m + h \sum_{j \in \mathcal{DS}_m \cap \mathcal{DS}_{m-1}} C^j_m W_j B^j_m \end{bmatrix} \begin{bmatrix} \lambda_1 \\ \vdots \\ \lambda_m \end{bmatrix}_{k+1} \end{aligned} \tag{2.14}$$

To sum it up, the block-diagonal term of matrix M_{OSNSP}, for block-row α is:

$$D_\alpha + h \sum_{j \in \mathcal{DS}_\alpha} C^j_\alpha W_j B^j_\alpha \tag{2.15}$$

This is an $m_\alpha \times m_\alpha$ square matrix. The extra-diagonal block term, in position (α, β) is:

$$h \sum_{j \in \mathcal{DS}_\alpha \cap \mathcal{DS}_\beta} C^j_\alpha W_j B^j_\beta \tag{2.16}$$

and is a $m_\alpha \times m_\beta$ matrix. This matrix differs from 0 when interactions α and β are coupled, ie have common DS.

Or, for the relation l of interaction α, we get:

$$D_{\alpha,l} + h \sum_{j \in \mathcal{DS}_\alpha} C^j_{\alpha,l} W_j B^j_\alpha \tag{2.17}$$

for the diagonal, and

$$h \sum_{j \in \mathcal{DS}_\alpha \cap \mathcal{DS}_\beta} C^j_{\alpha,l} W_j B^j_\beta \tag{2.18}$$

for extra-diagonal terms.
$D_{\alpha,l}$, row number l of D_α, the same for $C_{\alpha,l}$

Finally, the linked-Interaction map provides, for each interaction (named "current interaction"), the list of all the interactions (named "linked interaction") that have common dynamical system with the "current interaction".

2.1.2 A simple example

We consider $n = 3$ dynamical systems and $m = 2$ interactions:

$$\begin{aligned} I_\mu &\rightarrow \mathcal{DS}_\mu = \{DS_1, DS_3\}, m_\mu = 3 \\ I_\theta &\rightarrow \mathcal{DS}_\theta = \{DS_2, DS_3\}, m_\theta = 1 \end{aligned}$$

The linked-interaction map is :

$$\begin{aligned} I_\mu &\rightarrow I_\theta, commonDS = DS_3 \\ I_\theta &\rightarrow I_\mu, commonDS = DS_3 \end{aligned}$$

And:

$$\begin{aligned} M &= 4, N = \sum_{i=1}^{3} n_i \\ \mathcal{I}_1 &= \{I_\mu\} \\ \mathcal{I}_2 &= \{I_\theta\} \\ \mathcal{I}_3 &= \{I_\mu, I_\theta\} \end{aligned}$$

$$y_1 = \begin{bmatrix} C^1_1 & C^3_1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_3 \end{bmatrix} + D_1 \lambda_1 \tag{2.19}$$

$$y_2 = \begin{bmatrix} C^2_2 & C^3_2 \end{bmatrix} \begin{bmatrix} x_2 \\ x_3 \end{bmatrix} + D_2 \lambda_2 \tag{2.20}$$

$$\begin{bmatrix} R_1 \\ R_2 \\ R_3 \end{bmatrix} = \begin{bmatrix} B^1_1 \lambda_1 \\ B^2_2 \lambda_2 \\ B^3_1 \lambda_1 + B^3_2 \lambda_2 \end{bmatrix} \tag{2.21}$$

$$M_{OSNSP} = \begin{bmatrix} D_1 + hC^1_1 W_1 B^1_1 + hC^3_1 W_3 B^3_1 & hC^3_1 W_3 B^3_2 \\ hC^3_2 W_3 B^3_1 & D_2 + hC^2_2 W_2 B^2_2 + hC^3_2 W_3 B^3_2 \end{bmatrix} \begin{bmatrix} \lambda_1 \\ \lambda_2 \end{bmatrix}_{k+1} \tag{2.22}$$

2.1.3 relative degree

Let us consider the global vector

$$Y = \begin{bmatrix} y_1 \\ ... \\ y_M \end{bmatrix} = CX + D\Lambda \tag{2.23}$$

We denot r_j the relative degree of equation j, $j \in [1..M]$. We have:

$$y_j = \sum_{i=1}^{n} C_j^i x_i + D_{j,j}\lambda_j + \sum_{i\neq j, i=1}^{m} D_{j,i}\lambda_i \tag{2.24}$$

$D_{j,i}$ a scalar and C_j^i a $1 \times n_i$ line-vector.
If $D_{jj} \neq 0$, then $r_j = 0$. Else, we should consider the first derivative of y_j.
Before that, recall that:

$$R_i = \sum_{k=1}^{M} B_k^i \lambda_j \tag{2.25}$$

Through many of the B_j^i are equal to zero, we keep them all in the following lines.
Then:

$$\begin{aligned} \dot{y}_j &= \sum_{i=1}^{n} C_j^i (A_i x_i + \sum_{k=1}^{M} B_k^i \lambda_k) + f(\lambda_k)_{k\neq j} && (2.26) \\ &= \sum_{i=1}^{n} C_j^i (A_i x_i + B_j^i \lambda_j + \sum_{k=1, k\neq j}^{M} B_k^i \lambda_k) + \dots && (2.27) \end{aligned}$$

So, if $\sum_{i=1}^{n} C_j^i B_j^i \neq 0$ (note that this corresponds to the product between line j of C and column j of B) then $r_j = 1$ else we consider the next derivative, and so on.
In derivative r, the coefficient of λ_j will be:

$$coeff_j = \sum_{i=1}^{n} C_j^i (A_i)^{r-1} B_j^i \tag{2.28}$$

if $coeff_j \neq 0$ then $r_j = r$.

2.2 LagrangianDS - Lagrangian Linear Relations

2.2.1 General notations

We consider n dynamical systems, lagrangian and non linear, of the form:

$$M_i(q_i)\ddot{q}_i + N_i(\dot{q}_i, q_i) = F_{Int,i}(\dot{q}_i, q_i, t) + F_{Ext,i}(t) + p_i \tag{2.29}$$

Each system if of dimension n_i, and we denote $N = \sum_{i=1}^{n} n_i$.
An interaction, I_α is composed with a non smooth law, $nslaw_\alpha$ and a relation:

$$y_\alpha = H_\alpha Q_\alpha + b_\alpha \tag{2.30}$$

The "dimension" of the interaction, ie the size of vector y_α, is denoted m_α and we set:

$$M_y = \sum_{\alpha=1}^{m} m_\alpha$$

m being the number of interactions in the Non Smooth Dynamical System.
Q_α is a vector that represents the DS concerned by the interaction. Its dimension is noted N_α, this for n_α systems in the interaction.
H_α is a $m_\alpha \times N_\alpha$ row-blocks matrix and b_α a m_α vector.

$$H_\alpha = \left[\begin{array}{cccc} H_\alpha^i & H_\alpha^j & \dots \end{array} \right] \tag{2.31}$$

with $i, j, \dots \in \mathcal{DS}_\alpha$ which is the set of DS belonging to interaction α.
We also have the following relation:

$$\left[\begin{array}{c} R_\alpha^i \\ R_\alpha^j \\ \dots \end{array} \right] = {}^tH_\alpha \lambda_\alpha = \left[\begin{array}{c} {}^tH_\alpha^i \\ {}^tH_\alpha^j \\ \dots \end{array} \right] \lambda_\alpha \tag{2.32}$$

R_α^i represents the contribution of interaction α on the reaction of the dynamical system i, and ${}^tH_\alpha^i$ is a $n_i \times m_\alpha$ block matrix.
And so:

$$R_i = \sum_{\beta \in \mathcal{I}_i} R_\beta^i = \sum_{\beta \in \mathcal{I}_i} H_\beta^i \lambda_\beta \tag{2.33}$$

with $\mathcal{I}_i$ the set of interactions in which dynamical system number i is involved.
Introducing the time dicretisation, we get:

$$\begin{aligned} \dot{q}_i^{k+1} &= \dot{q}_{free,i} + W_i R_i^{k+1} \\ \dot{y}_\alpha^{k+1} &= H_\alpha \dot{Q}_\alpha^{k+1} \end{aligned} \tag{2.34}$$

$$R_i^{k+1} = \sum_{\beta \in \mathcal{I}_i} H_\beta^i \lambda_\beta^{k+1} \tag{2.35}$$

ie,

$$y_\alpha^{k+1} = H_\alpha Q_\alpha^{free} + H_\alpha W_\alpha {}^tH_\alpha \lambda_\alpha + \sum_{i \in \mathcal{DS}_\alpha} \sum_{\beta \in \mathcal{I}_i, \alpha \neq \beta} H_\alpha^i W_i H_\beta^j \lambda_\beta \tag{2.36}$$

with W_α given by (2.12).

The global-assembled Y vector, of dimension M, composed by m y_α subvectors, is given by:

$$Y_{k+1} = q_{OSNSP} + M_{OSNSP} \Lambda_{k+1} \tag{2.37}$$

with:

$$q_{OSNSP}^\alpha = H_\alpha Q_\alpha^{free} \tag{2.38}$$

and for M_{OSNSP}, the block-diagonal term for block-row α is

$$\sum_{j \in \mathcal{DS}_\alpha} H_\alpha^j W_j {}^tH_\alpha^j \tag{2.39}$$

an $m_\alpha \times m_\alpha$ square matrix. The extra-diagonal block term, in position (α, β) is:

$$\sum_{j \in \mathcal{DS}_\alpha \cap \mathcal{DS}_\beta} H_\alpha^j W_j {}^tH_\beta^j \tag{2.40}$$

and is a $m_\alpha \times m_\beta$ matrix. This matrix differs from 0 when interactions α and β are coupled, ie have common DS.

Or, for the relation l of interaction α, we get:

$$\sum_{j \in \mathcal{DS}_\alpha} H^j_{\alpha,l} {W_j}^t H^j_\alpha \tag{2.41}$$

for the diagonal, and

$$\sum_{j \in \mathcal{DS}_\alpha \cap \mathcal{DS}_\beta} H^j_{\alpha,l} {W_j}^t H^j_\beta \tag{2.42}$$

for extra-diagonal terms.
$H_{\alpha,l}$, row number l of H_α.

WARNING: depending on linear and non linear case for the DS, there should be a factor h ahead W. See Bouncing Ball template.

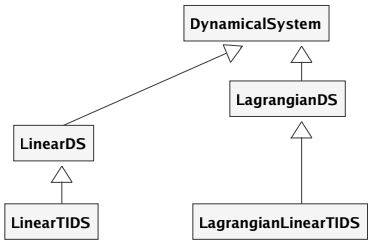
Chapter 3

Dynamical Systems formulations in Siconos.

author	F. Pérignon
date	March 22, 2006
version	Kernel 1.1.4

3.1 Class Diagram

There are four possible formulation for dynamical systems in Siconos, two for first order systems and two for second order Lagrangian systems. The main class is DynamicalSystem, all other derived from this one, as shown in the following diagram:



3.2 General non linear first order dynamical systems $\rightarrow$ **class** *DynamicalSystem*

This is the top class for dynamical systems. All other systems classes derived from this one.

A general dynamical systems is described by the following set of n equations, completed with initial conditions:

$$\dot{x} = f(x,t) + T(x)u(x,\dot{x},t) + r \tag{3.1}$$
$$x(t_0) = x_0 \tag{3.2}$$

- x: state of the system - Vector of size n.
- $f(x,t)$: vector field - Vector of size n.

- $u(x, \dot{x}, t)$: control term - Vector of size $uSize$.
- $T(x)$: $n \times uSize$ matrix, related to control term.
- r: input due to non-smooth behavior - Vector of size n.

The Jacobian matrix, $\nabla_x f(x,t)$, of f according to x, $n \times n$ square matrix, is also a member of the class.

Initial conditions are given by the member x_0, vector of size n. This corresponds to x value when simulation is starting, ie after a call to strategy->initialize().

There are plug-in functions in this class for f (vectorField), $jacobianX$, u and T. All of them can handle a vector of user-defined parameters.

3.3 First order linear dynamical systems → class *LinearDS*

Derived from DynamicalSystem, described by the set of n equations and initial conditions:

$$\dot{x} = A(t)x(t) + Tu(t) + b(t) + r \tag{3.3}$$
$$x(t_0) = x_0 \tag{3.4}$$

With:

- $A(t)$: $n \times n$ matrix, state independent but possibly time-dependent.
- $b(t)$: Vector of size n, possibly time-dependent.

Other variables are those of DynamicalSystem class.
A and B have corresponding plug-in functions.

Warning: time dependence for A and b is not available at the time in the simulation part for this kind of dynamical systems.

Links with vectorField and its Jacobian are:

$$f(x,t) = A(t)x(t) + b(t) \tag{3.5}$$
$$jacobianX = \nabla_x f(x,t) = A(t) \tag{3.6}$$

3.4 Second order non linear Lagrangian dynamical systems → class *LagrangianDS*

Lagrangian second order non linear systems are described by the following set of$nDof$ equations + initial conditions:

$$M(q)\ddot{q} + NNL(\dot{q}, q) + F_{Int}(\dot{q}, q, t) = F_{Ext}(t) + p \tag{3.7}$$
$$q(t_0) = q0 \tag{3.8}$$
$$\dot{q}(t_0) = velocity0 \tag{3.9}$$

With:

- $M(q)$: $nDof \times nDof$ matrix of inertia.
- q: state of the system - Vector of size $nDof$.
- $\dot{q}$ or $velocity$: derivative of the state according to time - Vector of size $nDof$.

- $NNL(\dot{q}, q)$: non linear terms, time-independent - Vector of size $nDof$.
- $F_{Int}(\dot{q}, q, t)$: time-dependent linear terms - Vector of size $nDof$.
- $F_{Ext}(t)$: external forces, time-dependent BUT do not depend on state - Vector of size $nDof$.
- p: input due to non-smooth behavior - Vector of size $nDof$.

The following Jacobian are also member of this class:

- jacobianQFInt = $\nabla_q F_{Int}(t, q, \dot{q})$ - $nDof \times nDof$ matrix.
- jacobianVelocityFInt = $\nabla_{\dot{q}} F_{Int}(t, q, \dot{q})$ - $nDof \times nDof$ matrix.
- jacobianQNNL = $\nabla_q NNL(q, \dot{q})$ - $nDof \times nDof$ matrix.
- jacobianVelocityNNL = $\nabla_{\dot{q}} NNL(q, \dot{q})$ - $nDof \times nDof$ matrix.

There are plug-in functions in this class for F_{int}, F_{Ext}, M, NNL and the four Jacobian matrices. All of them can handle a vector of user-defined parameters.

Links with first order dynamical system are:

$$n = 2nDof \tag{3.10}$$

$$x = \begin{bmatrix} q \\ \dot{q} \end{bmatrix} \tag{3.11}$$

$$f(x,t) = \begin{bmatrix} \dot{q} \\ M^{-1}(F_{Ext} - F_{Int} - NNL) \end{bmatrix} \tag{3.12}$$

$$\tag{3.13}$$

$$\nabla_x f(x,t) = \begin{bmatrix} 0_{nDof \times nDof} & I_{nDof \times nDof} \\ \nabla_q(M^{-1})(F_{Ext} - F_{Int} - NNL) - M^{-1}\nabla_q(F_{Int} + NNL) & -M^{-1}\nabla_{\dot{q}}(F_{Int} + NNL) \end{bmatrix} \tag{3.14}$$

$$r = \begin{bmatrix} 0_{nDof} \\ p \end{bmatrix} \tag{3.15}$$

$$u(x, \dot{x}, t) = u_L(\dot{q}, q, t) \text{ (not yet implemented)} \tag{3.16}$$

$$T(x) = \begin{bmatrix} 0_{nDof} \\ T_L(q) \end{bmatrix} \text{ (not yet implemented)} \tag{3.17}$$

$$\tag{3.18}$$

With 0_n a vector of zero of size n, $0_{n \times m}$ a $n \times m$ zero matrix and $I_{n \times n}$, identity $n \times n$ matrix.

Warning: control terms (Tu) are not fully implemented in Lagrangian systems. This will be part of future version.

3.5 Second order linear and time-invariant Lagrangian dynamical systems → class *LagrangianLinearTIDS*

$$M\ddot{q} + C\dot{q} + Kq = F_{Ext}(t) + p \tag{3.19}$$

With:

- C: constant viscosity $nDof \times nDof$ matrix
- K: constant rigidity $nDof \times nDof$ matrix

And:

$$F_{Int} = C\dot{q} + Kq \tag{3.20}$$

$$NNL = 0_{nDof} \tag{3.21}$$

Chapter 4

Dynamical Systems implementation in Siconos.

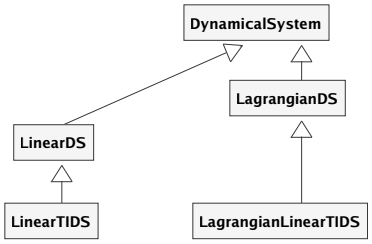
author	F. Pérignon
date	November 7, 2006
version	Kernel 1.3.0

4.1 Introduction

This document is only a sequel of notes and remarks on the way Dynamical Systems are implemented in Siconos.
It has to be completed, reviewed, reorganized etc etc for a future Developpers'Guide.
See also documentation in Doc/User/DynamicalSystemsInSiconos for a description of various dynamical systems types.

4.2 Class Diagram

There are four possible formulation for dynamical systems in Siconos, two for first order systems and two for second order Lagrangian systems. The main class is DynamicalSystem, all other derived from this one, as shown in the following diagram:



4.3 Construction

Each constructor must:

- initialize all the members of the class and of the top-class if it exists

- allocate memory and set value for all required inputs
- allocate memory and set value for optional input if they are given as argument (in xml for example)
- check that given data are coherent and that the system is complete (for example, in the LagrangianDS if the internal forces are given as a plug-in, their Jacobian are also required. If they are not given, this leads to an exception).

No memory allocation is made for unused members ⇒ requires if statements in simulation. (if!=NULL ...).

4.3.1 DynamicalSystem

Required data:
n, x0, f, jacobianXF
Optional:
T,u

Always allocated in constructor:
x, x0, xFree, r, rhs, jacobianXF
Warning: default constructor is always private or protected and apart from the others and previous rules or remarks do not always apply to it. This for DS class and any of the derived ones.

4.3.2 LagrangianDS

Required data:
ndof, q0, velocity0, mass
Optional:
fInt and its Jacobian, fExt, NNL and its Jacobian.

Always allocated in constructor:
mass, q, q0, qFree, velocity, velocity0, velocityFree, p.
All other pointers to vectors/matrices are set to NULL by default.
Memory vectors are required but allocated during call to initMemory function.
Various rules:

- **fInt (NNL) given as a plug-in ⇒ check that JacobianQ/Velocity are present (matrices or plug-in)**
- **any of the four Jacobian present ⇒ allocate memory for block-matrix jacobianX (connectToDS function)**
-

check: end of constructor or in initialize?
computeF and JacobianF + corresponding set functions: virtual or not?

4.4 Specific flags or members

- **isAllocatedIn: to check inside-class memory allocation**
- **isPlugin: to check if operators are computed with plug-in or just directly set as a matrix or vector**
- **workMatrix: used to save some specific matrices in order to avoid recomputation if possible (inverse of mass ...)**

4.5 plug-in management

DynamicalSystem class has a member named parameterList which is a $map < string, SimpleVector* >$**, ie a list of pointers to SimpleVector*, with a string as a key to identified them. For example,** $parametersList["mass"]$ **is a SimpleVector*, which corresponds to the last argument given in mass plug-in function.**
By default, each parameters vectors must be initialized with a SimpleVector of size 1, as soon as the plug-in is declared. Moreover, to each vector corresponds a flag in isAllocatedIn map, to check if the corresponding vector has been allocated inside the class or not.
For example, in DynamicalSystem, if $isPlugin["vectorField"] == true$**, then, during call to constructor or set function, it is necessary to defined the corresponding parameter:**
$parametersList["vectorField"] = newSimpleVector(1)$
and to complete the $isAllocatedIn$ **flag:**
$isAllocatedIn["parameter_for_vectorField"] = true$**.**

Chapter 5

Interactions

author	F. Pérignon
date	November 7, 2006
version	Kernel 1.3.0

5.1 Introduction

This document is only a sequel of notes and remarks on the way Interactions are implemented in Siconos.
It has to be completed, reviewed, reorganized etc etc for a future Developpers'Guide.
See also documentation in Doc/User/Interaction.

5.2 Class Diagram

5.3 Description

5.3.1 Redaction note F. PERIGNON

review of interactions (for EventDriven implementation) 17th May 2006.

- **variable *nInter* renamed in *interactionSize*: represents the size of y and λ. NOT the number of relations !!**

- **add a variable *nsLawSize* that depends on the non-smooth law type.**
  Examples:
  - **NewtonImpact -> *nsLawSize* = 1**
  - **Friction 2D -> *nsLawSize* = 2**
  - **Friction 3D -> *nsLawSize* = 3**
  - **...**
  - ***nsLawSize* = n with n dim of matrix D in : $y = Cx + D\lambda$, D supposed to be a full-ranked matrix.**
    Warning: this case is represented by only one relation of size n.

- ***numberOfRelations*: number of relations in the interaction, *numberOfRelations* = $\frac{interactionSize}{nsLawSize}$.**

Chapter 6

Notes on the Non Smooth Dynamical System construction

author	F. Pérignon
date	November 7, 2006
version	Kernel 1.3.0

6.1 Introduction

6.2 Class Diagram

6.3 Description

Objects must be constructed in the following order:

- **DynamicalSystems**
- **NonSmoothLaw: depends on nothing**
- **Relation: no link with an interaction during construction, this will be done during initialization.**
- **Interaction: default constructor is private and copy is forbidden. Two constructors: xml and from data. Required data are a DSSet, a NonSmoothLaw and a Relation (+ dim of the Interaction and a number).**
  Interaction has an initialize function which allocates memory for y and lambda, links correctly the relation and initializes it .... This function is called at the end of the constructor. That may be better to call it in simulation->initialize? Pb: xml constructor needs memory allocation for y and lambda if they are provided in the input xml file.
- **NonSmoothDynamicalSystem: default is private, copy fobidden. Two constructors: xml and from data. Required data are the DSSet and the InteractionsSet. The topology is declared and constructed (but empty) during constructor call of the nsds, but initialize in the Simulation, this because it can not be initialize until the nsds has been fully described (ie this to allow user to add DS, Inter ...) at any time in the model, but before simulation initialization).**

6.4 misc

- **no need to keep a number for Interactions? Only used in xml for OSI, to know which Interactions it holds.**
- **pb: the number of saved derivatives for y and lambda in Interactions is set to 2. This must depends on the relative degree which is computes during Simulation initialize and thus too late. It is so not available when memory is allocated (Interaction construction). Problem-> to be reviewed.**

Chapter 7

OneStepIntegrator and derived classes.

author	F. Pérignon
date	November 7, 2006
version	Kernel 1.3.0

7.1 Introduction

This document is only a sequel of notes and remarks on the way OneStepIntegrators are implemented in Siconos.
It has to be completed, reviewed, reorganized etc etc for a future Developpers'Guide.
See also documentation in Doc/User/OneStepIntegrator for a description of various OSI.

7.2 Class Diagram

7.3 Misc

OSI review for consistency between Lsodar and Moreau:

- **add set of DynamicalSystem***
- **add set of Interaction***
- **add link to strategy that owns the OSI**
- **remove td object in OSI -> future: replace it by a set of td (one per ds)**
- **add strat in constructors arg list**

osi -> strat -> Model -> nsds -> topology
osi -> strat -> timeDiscretisation

let a timeDiscretisation object in the OSI? set of td (one per ds)?
create a class of object that corresponds to DS on the simulation side ?
will contain the DS, its discretization, theta for Moreau ... ?
Allow setStrategyPtr operation? Warning: need reinitialisation.

Required input by user:

- **list of DS or list of Interactions ?**
- **pointer to strategy**
- **...**

7.4 Construction

Each constructor must:

-

7.4.1 Moreau

Two maps: one for W, and one for theta. To each DS corresponds a theta and a W.
Strategy arg in each constructor.

Required data:

Optional:

Always allocated in constructor:

Warning: default constructor is always private or protected and apart from the others and previous rules or remarks do not always apply to it.

7.4.2 Lsodar

Required data:

Optional:

Always allocated in constructor:

Chapter 8

Newton's linearization

Last update : 2009/07/07 This section is devoted to the implementation and the study of the algorithm. The interval of integration is $[0, T]$, $T > 0$, and a grid $t_0 = 0$, $t_{k3+1} = t_k + h$, $k \geq 0$, $t_N = T$ is constructed. The approximation of a function $f(\cdot)$ on $[0, T]$ is denoted as $f^N(\cdot)$, and is a piecewise constant function, constant on the intervals $[t_k, t_{k+1})$. We denote $f^N(t_k)$ as f_k. The time-step is $h > 0$.

8.0.3 Time–discretization of the general case (??)

This fully general case is not yet implemented in Siconos.

Starting from (??), let us introduce anew notation,

$$\begin{aligned} \dot{x}(t) &= f(x(t), t) + r(t) \\ y(t) &= h(t, x(t), \lambda(t)) \\ r(t) &= g(t, x(t), \lambda(t)) \end{aligned} \tag{8.1}$$

where $s(t) \in \mathbb{R}^m$ and $y(t) \in \mathbb{R}^m$ are complementary variables related trough the Sgn multi-valued mapping. According to the class of systems (??), (??) or (??) , we are studying, the function f and g are defined by a fully nonlinear framework or by affine functions. We have decided to present the time-discretization in its full generality and specialize the algorithms for each cases in Section 8.0.5.

Let us now proceed with the time discretization of (8.49) by a fully implicit scheme :

$$\begin{aligned} x_{k+1} &= x_k + h\theta f(x_{k+1}, t_{k+1}) + h(1-\theta) f(x_k, t_k) + h\theta_r r(t_{k+1}) + h(1-\theta_r) r(t_k) \\ y_{k+1} &= h(t+1, x_{k+1}, \lambda_{k+1}) \\ r_{k+1} &= g(x_{k+\theta}, \lambda_{k+1}, t+1) \end{aligned} \tag{8.2}$$

where $\theta = [0, 1]$ and $\theta_r \in [0, 1]$. As in ?, we call the problem (8.50) the "one–step nonsmooth problem".

This time-discretization is slightly more general than a standard implicit Euler scheme. The main discrepancy lies in the choice of a θ-method to integrate the nonlinear term. For $\theta = 0$, we retrieve the explicit integration of the smooth and single valued term f. Moreover for $\gamma = 0$, the term g is explicitly evaluated. The flexibility in the choice of θ and γ allows the user to improve and control the accuracy, the stability and the numerical damping of the proposed method. For instance, if the smooth dynamics given by f is stiff, or if we have to use big step sizes for practical reasons, the choice of $\theta > 1/2$ offers better stability with the respect to h.

8.0.4 Newton's linearization

Due to the fact that two of the studied classes of systems that are studied in this paper are affine functions in terms of f and g, we propose to solve the "one–step nonsmooth problem" (8.50) by performing an external Newton linearization, which yields a Mixed Linear Complementarity Problems (MLCP).

Newton's linearization The first line of the problem (8.50) can be written under the form of a residue $\mathcal{R}$ **depending only on** x_{k+1} **and** r_{k+1} **such that**

$$\mathcal{R}(x_{k+1}, r_{k+1}) = 0 \tag{8.3}$$

with $\mathcal{R}(x,r) = x - x_k - h\theta f(x, t_{k+1}) - h(1-\theta)f(x_k, t_k) - h\theta_r r - h(1-\theta_r)r_k$**. The solution of this system of nonlinear equations is sought as a limit of the sequence** $\{x^\alpha_{k+1}, r^\alpha_{k+1}\}_{\alpha\in\mathbb{N}}$ **such that**

$$\begin{cases} x^0_{k+1} = x_k \\ \\ \mathcal{R}_L(x^{\alpha+1}_{k+1}, r^{\alpha+1}_{k+1}) = \mathcal{R}(x^\alpha_{k+1}, r^\alpha_{k+1}) + \left[\nabla_x \mathcal{R}(x^\alpha_{k+1}, r^\alpha_{k+1})\right](x^{\alpha+1}_{k+1} - x^\alpha_{k+1}) + \left[\nabla_x \mathcal{R}(x^\alpha_{k+1}, r^\alpha_{k+1})\right](r^{\alpha+1}_{k+1} - r^\alpha_{k+1}) = 0 \end{cases} \tag{8.4}$$

The residu free is also defined (useful for implementation only):

$$\mathcal{R}_{free}(x) = x - x_k - h\theta f(x, t_{k+1}) - h(1-\theta)f(x_k, t_k)$$

$$\mathcal{R}(x,r) = \mathcal{R}_{free}(x) - h\theta_r r - h(1-\theta_r)r_k$$

$$\mathcal{R}^\alpha_{k+1} = \mathcal{R}(x^\alpha_{k+1}, r^\alpha_{k+1}) = \mathcal{R}_{free}(x^\alpha_{k+1}, r^\alpha_{k+1}) - h\theta_r r^\alpha_{k+1} - h(1-\theta_r)r_k$$

$$\mathcal{R}^\alpha_{freek+1} = \mathcal{R}_{free}(x^\alpha_{k+1}, r^\alpha_{k+1}) = x^\alpha_{k+1} - x_k - h\theta f(x^\alpha_{k+1}, t_{k+1}) - h(1-\theta)f(x_k, t_k)$$

The computation of the Jacobian of $\mathcal{R}$ **with respect to** x**, denoted by** $M(x,\lambda)$ **leads to**

$$M^\alpha_{k+1} = \nabla_x \mathcal{R}(x^\alpha_{k+1}, r^\alpha_{k+1}) = I - h\theta \nabla_x f(x^\alpha_{k+1}, t_{k+1}). \tag{8.5}$$

At each time–step, we have to solve the following linearized problem,

$$\mathcal{R}^\alpha_{k+1} + M^\alpha_{k+1}(x^{\alpha+1}_{k+1} - x^\alpha_{k+1}) - h\theta_r(r^{\alpha+1}_{k+1} - r^\alpha_{k+1}) = 0, \tag{8.6}$$

that is

$$h\theta_r r^{\alpha+1}_{k+1} = r_c + M^\alpha_{k+1} x^{\alpha+1}_{k+1}. \tag{8.7}$$

with

$$\begin{aligned} r_c &= h\theta_r r^\alpha_{k+1} - M^\alpha_{k+1} x^\alpha_{k+1} + \mathcal{R}^\alpha_{k+1} = -M^\alpha_{k+1} x^\alpha_{k+1} + \mathcal{R}^\alpha_{freek+1} - h(1-\theta_r)r_k \\ \mathcal{R}^\alpha_{k+1} &= x^\alpha_{k+1} - x_k - h\theta f(x^\alpha_{k+1}) - h(1-\theta)f(x_k) - h\theta_r r^\alpha_{k+1} - h(1-\theta_r)r_k \end{aligned} \tag{8.8}$$

The matrix M **is clearly non singular for small** h**. The same operation is performed with the second equation of (8.50)**

$$\begin{aligned} &\mathcal{R}_y(x, y, \lambda) = y - h(t_{k+1}, x, \lambda) = 0 \\ &\mathcal{R}_{Ly}(x^{\alpha+1}_{k+1}, y^{\alpha+1}_{k+1}, \lambda^{\alpha+1}_{k+1}) = \mathcal{R}_y(x^\alpha_{k+1}, y^\alpha_{k+1}, \lambda^\alpha_{k+1}) + (y^{\alpha+1}_{k+1} - y^\alpha_{k+1}) - \\ &\qquad C^\alpha_{k+1}(x^{\alpha+1}_{k+1} - x^\alpha_{k+1}) - D^\alpha_{k+1}(\lambda^{\alpha+1}_{k+1} - \lambda^\alpha_{k+1}) = 0 \end{aligned} \tag{8.9}$$

$\mathcal{R}_y(x^{\alpha+1}_{k+1}, y^{\alpha+1}_k + 1, \lambda^{\alpha+1}_{k+1})$ **leading to the following linearized equation**

$$y^{\alpha+1}_{k+1} = y^\alpha_{k+1} - \mathcal{R}^\alpha_{yk+1} + C^\alpha_{k+1}(x^{\alpha+1}_{k+1} - x^\alpha_{k+1}) + D^\alpha_{k+1}(\lambda^{\alpha+1}_{k+1} - \lambda^\alpha_{k+1}). \tag{8.10}$$

with,

$$\begin{aligned} C^\alpha_{k+1} &= \nabla_x h(t_{k+1}, x^\alpha_{k+1}, \lambda^\alpha_{k+1}) \\ D^\alpha_{k+1} &= \nabla_\lambda h(t_{k+1}, x^\alpha_{k+1}, \lambda^\alpha_{k+1}) \end{aligned} \tag{8.11}$$

and

$$\mathcal{R}^\alpha_{yk+1} = y^\alpha_{k+1} - h(x^\alpha_{k+1}, \lambda^\alpha_{k+1}) \tag{8.12}$$

The same operation is performed with the thirdtheta equation of (8.50)

$$\begin{aligned}
&\mathcal{R}_r(r,x,\lambda) = r - g(x,\lambda,t_{k+1}) = 0\\
&\mathcal{R}_{L\lambda}(r^{\alpha+1}_{k+1},x^{\alpha+1}_{k+1},\lambda^{\alpha+1}_{k+1}) = \mathcal{R}^{\alpha}_{rk+1} + (r^{\alpha+1}_{k+1} - r^{\alpha}_{k+1}) - K^{\alpha}_{k+1}(x^{\alpha+1}_{k+1} - x^{\alpha}_{k+1}) - B^{\alpha}_{k+1}(\lambda^{\alpha+1}_{k+1} - \lambda^{\alpha}_{k+1}) = 0\\
&r^{\alpha+1}_{k+1} = r_1 + K^{\alpha}_{k+1}x^{\alpha+1}_{k+1} + B^{\alpha}_{k+1}\lambda^{\alpha+1}_{k+1}
\end{aligned} \tag{8.13}$$

with,

$$r_1 = g(x^{\alpha}_{k+1}, r^{\alpha}_{k+1}, t_{k+1}) - K^{\alpha}_{k+1}x^{\alpha}_{k+1} - B^{\alpha}_{k+1}\lambda^{\alpha}_{k+1} \tag{8.14}$$

and,

$$\begin{aligned}
K^{\alpha}_{k+1} &= \nabla_x g(x^{\alpha}_{k+1}, r^{\alpha}_{k+1}, t_{k+1})\\
B^{\alpha}_{k+1} &= \nabla_\lambda g(x^{\alpha}_{k+1}, r^{\alpha}_{k+1}, t_{k+1})
\end{aligned} \tag{8.15}$$

and the r residue:

$$\mathcal{R}^{\alpha}_{rk+1} = r^{\alpha}_{k+1} - g(x^{\alpha}_{k+1}, r^{\alpha}_{k+1}, t_{k+1}) \tag{8.16}$$

Inserting (8.31), we get the following linear relation between $x^{\alpha+1}_{k+1}$ and $\lambda^{\alpha+1}_{k+1}$,

$$\begin{aligned}
&(M^{\alpha}_{k+1} - h\theta_r K^{\alpha}_{k+1})x^{\alpha+1}_{k+1} = h\theta_r\left[-\mathcal{R}^{\alpha}_{rk+1} + r^{\alpha}_{k+1} - K^{\alpha}_{k+1}x^{\alpha}_{k+1} + B^{\alpha}_{k+1}(\lambda^{\alpha+1}_{k+1} - \lambda^{\alpha}_{k+1})\right] - r_c\\
&x^{\alpha+1}_{k+1} = x_p + h\theta_r(M^{\alpha}_{k+1} - h\theta_r K^{\alpha}_{k+1})^{-1}B^{\alpha}_{k+1}\lambda^{\alpha+1}_{k+1}
\end{aligned} \tag{8.17}$$

with

$$x_p = (M^{\alpha}_{k+1} - h\theta_r K^{\alpha}_{k+1})^{-1}\left[-r_c + h\theta_r(-\mathcal{R}^{\alpha}_{rk+1} + r^{\alpha}_{k+1} - K^{\alpha}_{k+1}x^{\alpha}_{k+1} - B^{\alpha}_{k+1}\lambda^{\alpha}_{k+1})\right] \tag{8.18}$$

Case K=0 leads to :

$$\begin{aligned}
x_p &= x_{free} + h\theta_r(-\mathcal{R}^{\alpha}_{rk+1} + r^{\alpha}_{k+1} - K^{\alpha}_{k+1}x^{\alpha}_{k+1} - B^{\alpha}_{k+1}\lambda^{\alpha}_{k+1})\\
x_{free} &= x^{\alpha}_{k+1} + (M^{\alpha}_{k+1})^{-1}(\mathcal{R}^{\alpha}_{freek+1} + h(1-\theta_r)r_k)\\
x_p &= (M^{\alpha}_{k+1} - h\theta_r K^{\alpha}_{k+1})^{-1}\left[-r_c + h\theta_r(-\mathcal{R}^{\alpha}_{rk+1} + r^{\alpha}_{k+1} - K^{\alpha}_{k+1}x^{\alpha}_{k+1} - B^{\alpha}_{k+1}\lambda^{\alpha}_{k+1})\right]
\end{aligned} \tag{8.19}$$

Inserting (8.34), we get the following linear relation between $y^{\alpha+1}_{k+1}$ and $\lambda^{\alpha+1}_{k+1}$,

$$y^{\alpha+1}_{k+1} = y_p + \left[h\theta_r C^{\alpha}_{k+1}(M^{\alpha}_{k+1} - h\theta_r K^{\alpha}_{k+1})^{-1}B^{\alpha}_{k+1} + D^{\alpha}_{k+1}\right]\lambda^{\alpha+1}_{k+1} \tag{8.20}$$

with

$$y_p = y^{\alpha}_{k+1} - \mathcal{R}^{\alpha}_{yk+1} + C^{\alpha}_{k+1}(x_p - x^{\alpha}_{k+1}) - D^{\alpha}_{k+1}\lambda^{\alpha}_{k+1} \tag{8.21}$$

Mixed linear complementarity problem (MLCP) To summarize, the problem to be solved in each Newton iteration is:

$$\begin{cases} y^{\alpha+1}_{k+1} = W^{\alpha}_{k+1}\lambda^{\alpha+1}_{k+1} + b^{\alpha}_{k+1} \\ \\ -y^{\alpha+1}_{k+1} \in N_{[l,u]}(\lambda^{\alpha+1}_{k+1}). \end{cases} \tag{8.22}$$

with $W \in I\!R^{m\times m}$ and $b \in I\!R^{m}$ defined by

$$\begin{aligned}
W^{\alpha}_{k+1} &= h\theta_r C^{\alpha}_{k+1}(M^{\alpha}_{k+1} - h\theta K^{\alpha}_{k+1})^{-1}B^{\alpha}_{k+1} + D^{\alpha}_{k+1}\\
b^{\alpha}_{k+1} &= y_p
\end{aligned} \tag{8.23}$$

The problem (8.46) is equivalent to a Mixed Linear Complementarity Problem (MLCP) which can be solved under suitable assumptions by many linear complementarity solvers such as pivoting

techniques, interior point techniques and splitting/projection strategies. The reformulation into a standard MLCP follows the same line as for the MCP in the previous section. One obtains,

$$\begin{aligned} & y_{k+1}^{\alpha+1} = -W_{k+1}^{\alpha}\lambda_{k+1}^{\alpha+1} + b_{k+1}^{\alpha} \\ & (y_{k+1}^{\alpha+1})_i = 0 \qquad \textbf{for } i \in \{1..n\} \\ & 0 \leq (\lambda_{k+1}^{\alpha+1})_i \perp (y_{k+1}^{\alpha+1})_i \geq 0 \qquad \textbf{for } i \in \{n..n+m\} \end{aligned} \tag{8.24}$$

MLCP solvers. As for MCP, there exists numerous methods to numerically solve MLCP. In the worst case when the matrix $W_{k+1}^{\alpha+1}$ has no special properties, the MCLP can be always solved by enumerative solvers for which various implementation can be found. With positivity, P-matrix or co-positivity properties, standard methods for LCP? can be straightforwardly extended. Among these methods, we can cite the family of projection/splitting methods, interior point methods and semi-smooth Newton methods (see ? for an overview of various types of methods.).

8.0.5 The special cases of the affine systems (??) and (??)

In this section, we specify the time–discretization of the fully nonlinear case for the two other classes of systems (??) and (??) and for particular value of θ and γ.

8.0.6 Time–discretization of the the case (??)

This case is implemented in Siconos with the relation FirstOrderType2R.

Starting from (??), let us introduce a new notation,

$$\begin{aligned} M\dot{x}(t) &= f(x(t),t) + r(t) \\ y(t) &= h(t,x(t),\lambda(t)) \\ r(t) &= g(t,\lambda(t)) \end{aligned} \tag{8.25}$$

Let us now proceed with the time discretization of (8.49) by a fully implicit scheme :

$$\begin{aligned} Mx_{k+1} &= Mx_k + h\theta f(x_{k+1},t_{k+1}) + h(1-\theta)f(x_k,t_k) + h\theta_r r(t_{k+1}) + h(1-\theta_r)r(t_k) \\ y_{k+1} &= h(t+1,x_{k+1},\lambda_{k+1}) \\ r_{k+1} &= g(\lambda_{k+1},t+1) \end{aligned} \tag{8.26}$$

8.0.7 Newton's linearization

Newton's linearization **The first line of the problem (8.50) can be written under the form of a residue $\mathcal{R}$ depending only on x_{k+1} and r_{k+1} such that**

$$\mathcal{R}(x_{k+1},r_{k+1}) = 0 \tag{8.27}$$

with $\mathcal{R}(x,r) = M(x-x_k) - h\theta f(x,t_{k+1}) - h(1-\theta)f(x_k,t_k) - h\theta_r r - h(1-\theta_r)r_k$. The solution of this system of nonlinear equations is sought as a limit of the sequence $\{x^\alpha_{k+1}, r^\alpha_{k+1}\}_{\alpha\in N}$ such that

$$\begin{cases} x^0_{k+1} = x_k \\ \\ \mathcal{R}_L(x^{\alpha+1}_{k+1}, r^{\alpha+1}_{k+1}) = \mathcal{R}(x^\alpha_{k+1}, r^\alpha_{k+1}) + \left[\nabla_x \mathcal{R}(x^\alpha_{k+1}, r^\alpha_{k+1})\right](x^{\alpha+1}_{k+1} - x^\alpha_{k+1}) + \left[\nabla_x \mathcal{R}(x^\alpha_{k+1}, r^\alpha_{k+1})\right](r^{\alpha+1}_{k+1} - r^\alpha_{k+1}) = 0 \end{cases} \tag{8.28}$$

The residu free is also defined (useful for implementation only):

$$\mathcal{R}_{free}(x) = M(x-x_k) - h\theta f(x,t_{k+1}) - h(1-\theta)f(x_k,t_k)$$

$$\mathcal{R}(x,r) = \mathcal{R}_{free}(x) - h\theta_r r - h(1-\theta_r)r_k$$

$$\boxed{\mathcal{R}^\alpha_{k+1} = \mathcal{R}(x^\alpha_{k+1}, r^\alpha_{k+1}) = \mathcal{R}_{free}(x^\alpha_{k+1}, r^\alpha_{k+1}) - h\theta_r r^\alpha_{k+1} - h(1-\theta_r)r_k}$$

$$\mathcal{R}^\alpha_{freek+1} = \mathcal{R}_{free}(x^\alpha_{k+1}, r^\alpha_{k+1}) = \boxed{M(x^\alpha_{k+1} - x_k) - h\theta f(x^\alpha_{k+1}, t_{k+1}) - h(1-\theta)f(x_k,t_k)}$$

The computation of the Jacobian of $\mathcal{R}$ with respect to x, denoted by $M(x,\lambda)$ leads to

$$W^\alpha_{k+1} = \nabla_x \mathcal{R}(x^\alpha_{k+1}, r^\alpha_{k+1}) = M - h\theta\nabla_x f(x^\alpha_{k+1}, t_{k+1}). \tag{8.29}$$

At each time–step, we have to solve the following linearized problem,

$$\mathcal{R}^\alpha_{k+1} + W^\alpha_{k+1}(x^{\alpha+1}_{k+1} - x^\alpha_{k+1}) - h\theta_r(r^{\alpha+1}_{k+1} - r^\alpha_{k+1}) = 0, \tag{8.30}$$

that is

$$h\theta_r r^{\alpha+1}_{k+1} = r_c + W^\alpha_{k+1}x^{\alpha+1}_{k+1}. \tag{8.31}$$

with

$$r_c = h\theta_r r^\alpha_{k+1} - W^\alpha_{k+1}x^\alpha_{k+1} + \mathcal{R}^\alpha_{k+1} = -W^\alpha_{k+1}x^\alpha_{k+1} + \mathcal{R}^\alpha_{freek+1} - h(1-\theta_r)r_k \tag{8.32}$$

so:

$$x^{\alpha+1}_{k+1} = h(W^\alpha_{k+1})^{-1}r^{\alpha+1}_{k+1} + (W^\alpha_{k+1})^{-1}(\mathcal{R}^\alpha_{freek+1}) + x^\alpha_{k+1}$$

$$\boxed{x_{k+1}^{\alpha+1} = h(W_{k+1}^{\alpha})^{-1} r_{k+1}^{\alpha+1} + x_{free}}$$

The matrix W is clearly non singular for small h. The same operation is performed with the second equation of (8.50)

$$\begin{aligned} &\mathcal{R}_y(x, y, \lambda) = y - h(t_{k+1}, x, \lambda) = 0 \\ &\mathcal{R}_{Ly}(x_{k+1}^{\alpha+1}, y_{k+1}^{\alpha+1}, \lambda_{k+1}^{\alpha+1}) = \mathcal{R}_y(x_{k+1}^{\alpha}, y_{k+1}^{\alpha}, \lambda_{k+1}^{\alpha}) + (y_{k+1}^{\alpha+1} - y_{k+1}^{\alpha}) - \\ &\qquad C_{k+1}^{\alpha}(x_{k+1}^{\alpha+1} - x_{k+1}^{\alpha}) - D_{k+1}^{\alpha}(\lambda_{k+1}^{\alpha+1} - \lambda_{k+1}^{\alpha}) = 0 \end{aligned} \tag{8.33}$$

$\mathcal{R}_y(x_{k+1}^{\alpha+1}, y_{k+1}^{\alpha+1}, \lambda_{k+1}^{\alpha+1})$ **leading to the following linearized equation**

$$y_{k+1}^{\alpha+1} = y_{k+1}^{\alpha} - \mathcal{R}_{yk+1}^{\alpha} + C_{k+1}^{\alpha}(x_{k+1}^{\alpha+1} - x_{k+1}^{\alpha}) + D_{k+1}^{\alpha}(\lambda_{k+1}^{\alpha+1} - \lambda_{k+1}^{\alpha}). \tag{8.34}$$

with,

$$\begin{aligned} C_{k+1}^{\alpha} &= \nabla_x h(t_{k+1}, x_{k+1}^{\alpha}, \lambda_{k+1}^{\alpha}) \\ D_{k+1}^{\alpha} &= \nabla_\lambda h(t_{k+1}, x_{k+1}^{\alpha}, \lambda_{k+1}^{\alpha}) \end{aligned} \tag{8.35}$$

and

$$\boxed{\mathcal{R}_{yk+1}^{\alpha} = y_{k+1}^{\alpha} - h(x_{k+1}^{\alpha}, \lambda_{k+1}^{\alpha}, t_{k+1})} \tag{8.36}$$

The same operation is performed with the thirdtheta equation of (8.50)

$$\begin{aligned} &\mathcal{R}_r(r, \lambda) = r - g(\lambda, t_{k+1}) = 0 \\ &\mathcal{R}_{L\lambda}(r_{k+1}^{\alpha+1}, \lambda_{k+1}^{\alpha+1}) = \mathcal{R}_{rk+1}^{\alpha} + (r_{k+1}^{\alpha+1} - r_{k+1}^{\alpha}) - B_{k+1}^{\alpha}(\lambda_{k+1}^{\alpha+1} - \lambda_{k+1}^{\alpha}) = 0 \\ &r_{k+1}^{\alpha+1} = -\mathcal{R}_{rk+1}^{\alpha} + r_{k+1}^{\alpha} + B_{k+1}^{\alpha}(\lambda_{k+1}^{\alpha+1} - \lambda_{k+1}^{\alpha}) \\ &\boxed{r_{k+1}^{\alpha+1} = g(\lambda_{k+1}^{\alpha}, t_{k+1}) - B_{k+1}^{\alpha}\lambda_{k+1}^{\alpha} + B_{k+1}^{\alpha}\lambda_{k+1}^{\alpha+1}} \end{aligned} \tag{8.37}$$

because the r residue is:

$$\mathcal{R}_{rk+1}^{\alpha} = r_{k+1}^{\alpha} - g(r_{k+1}^{\alpha}, t_{k+1}) \tag{8.38}$$

with,

$$B_{k+1}^{\alpha} = \nabla_\lambda g(r_{k+1}^{\alpha}, t_{k+1}) \tag{8.39}$$

Inserting (8.31), we get the following linear relation between $x_{k+1}^{\alpha+1}$ and $\lambda_{k+1}^{\alpha+1}$,

$$\begin{aligned} &(W_{k+1}^{\alpha})x_{k+1}^{\alpha+1} = h\theta_r \left[g(r_{k+1}^{\alpha}, t_{k+1}) + B_{k+1}^{\alpha}(\lambda_{k+1}^{\alpha+1} - \lambda_{k+1}^{\alpha}) \right] - r_c \\ &x_{k+1}^{\alpha+1} = h\theta_r (W_{k+1}^{\alpha})^{-1} \left[g(r_{k+1}^{\alpha}, t_{k+1}) + B_{k+1}^{\alpha}(\lambda_{k+1}^{\alpha+1} - \lambda_{k+1}^{\alpha}) \right] - (W_{k+1}^{\alpha})^{-1}(-W_{k+1}^{\alpha}x_{k+1}^{\alpha} + \mathcal{R}_{freek+1}^{\alpha} - h(1-\theta_r)r_k) \\ &x_{k+1}^{\alpha+1} = x_p + h\theta_r (W_{k+1}^{\alpha})^{-1} B_{k+1}^{\alpha}\lambda_{k+1}^{\alpha+1} \end{aligned} \tag{8.40}$$

with $\theta_r = 1$:

$$\begin{aligned} (W_{k+1}^{\alpha})x_{k+1}^{\alpha+1} &= hr_{k+1}^{\alpha+1} - \mathcal{R}_{freek+1}^{\alpha} + W_{k+1}^{\alpha})x_{k+1}^{\alpha} \\ x_{k+1}^{\alpha+1} &= h(W_{k+1}^{\alpha})^{-1} r_{k+1}^{\alpha+1} - (W_{k+1}^{\alpha})^{-1}\mathcal{R}_{freek+1}^{\alpha} + x_{k+1}^{\alpha} \\ x_{k+1}^{\alpha+1} &= h(W_{k+1}^{\alpha})^{-1} r_{k+1}^{\alpha+1} + x_{free} \end{aligned}$$

with

$$x_p - x_{k+1}^\alpha = (W_{k+1}^\alpha)^{-1}\left[-r_c + h\theta_r(g(r_{k+1}^\alpha, t_{k+1}) - B_{k+1}^\alpha\lambda_{k+1}^\alpha)\right] \tag{8.41}$$

$$x_p - x_{k+1}^\alpha = \tilde{x}_{free} + h\theta_r(W_{k+1}^\alpha)^{-1}(g(r_{k+1}^\alpha, t_{k+1}) - B_{k+1}^\alpha\lambda_{k+1}^\alpha)$$

$$\tilde{x}_{free} = (W_{k+1}^\alpha)^{-1}(-\mathcal{R}_{freek+1}^\alpha + h(1-\theta_r)r_k)$$

$$x_{free} = \tilde{x}_{free} + x_{k+1}^\alpha = \boxed{-W^{-1}R_{freek+1}^\alpha + x_{k+1}^\alpha}$$

$$\boxed{x_p = x_{free} + h\theta_r(W_{k+1}^\alpha)^{-1}(g(\lambda_{k+1}^\alpha, t_{k+1}) - B_{k+1}^\alpha\lambda_{k+1}^\alpha)}$$

Inserting (8.34), we get the following linear relation between $y_{k+1}^{\alpha+1}$ and $\lambda_{k+1}^{\alpha+1}$,

$$y_{k+1}^{\alpha+1} = y_p + \left[h\theta_r C_{k+1}^\alpha (W_{k+1}^\alpha)^{-1}B_{k+1}^\alpha + D_{k+1}^\alpha\right]\lambda_{k+1}^{\alpha+1} \tag{8.42}$$

with

$$\boxed{y_p = y_{k+1}^\alpha - \mathcal{R}_{yk+1}^\alpha + C_{k+1}^\alpha(x_p - x_{k+1}^\alpha) - D_{k+1}^\alpha\lambda_{k+1}^\alpha} \tag{8.43}$$

$$y_p = h(x_{k+1}^\alpha, \lambda_{k+1}^\alpha) + C_{k+1}^\alpha(x_p - x_{k+1}^\alpha) - D_{k+1}^\alpha\lambda_{k+1}^\alpha \tag{8.44}$$

$$y_p = h(x_{k+1}^\alpha, \lambda_{k+1}^\alpha) - C_{k+1}^\alpha x_{k+1}^\alpha + C_{k+1}^\alpha x_{free} + hC(W_{k+1}^\alpha)^{-1}(g(\lambda_{k+1}^\alpha, t_{k+1}) - B_{k+1}^\alpha\lambda_{k+1}^\alpha) - D_{k+1}^\alpha\lambda_{k+1}^\alpha \tag{8.45}$$

Mixed linear complementarity problem (MLCP) To summarize, the problem to be solved in each Newton iteration is:

$$\begin{cases} y_{k+1}^{\alpha+1} = \tilde{W}_{k+1}^\alpha\lambda_{k+1}^{\alpha+1} + b_{k+1}^\alpha \\ \\ -y_{k+1}^{\alpha+1} \in N_{[l,u]}(\lambda_{k+1}^{\alpha+1}). \end{cases} \tag{8.46}$$

with $W \in I\!R^{m\times m}$ and $b \in I\!R^m$ defined by

$$\begin{array}{l} \tilde{W}_{k+1}^\alpha = h\theta_r C_{k+1}^\alpha (W_{k+1}^\alpha)^{-1}B_{k+1}^\alpha + D_{k+1}^\alpha \\ b_{k+1}^\alpha = y_p \end{array} \tag{8.47}$$

The problem (8.46) is equivalent to a Mixed Linear Complementarity Problem (MLCP) which can be solved under suitable assumptions by many linear complementarity solvers such as pivoting techniques, interior point techniques and splitting/projection strategies. The reformulation into a standard MLCP follows the same line as for the MCP in the previous section. One obtains,

$$\begin{array}{l} y_{k+1}^{\alpha+1} = -W_{k+1}^\alpha\lambda_{k+1}^{\alpha+1} + b_{k+1}^\alpha \\ \\ (y_{k+1}^{\alpha+1})_i = 0 \qquad \textbf{for } i \in \{1..n\} \\ 0 \le (\lambda_{k+1}^{\alpha+1})_i \perp (y_{k+1}^{\alpha+1})_i \ge 0 \qquad \textbf{for } i \in \{n..n+m\} \end{array} \tag{8.48}$$

MLCP solvers. As for MCP, there exists numerous methods to numerically solve MLCP. In the worst case when the matrix $W_{k+1}^{\alpha+1}$ has no special properties, the MCLP can be always solved by enumerative solvers for which various implementation can be found. With positivity, P-matrix or co-positivity properties, standard methods for LCP? can be straightforwardly extended. Among these methods, we can cite the family of projection/splitting methods, interior point methods and semi-smooth Newton methods (see ? for an overview of various types of methods.).

8.0.8 Implementation In siconos

The initial residu x:

$$\mathcal{R}^0_{k+1} = \mathcal{R}^0_{freek+1} - hr^0_{k+1}$$

$$\mathcal{R}^0_{k+1} = M(x^0_{k+1} - x_k) - h\theta f(x_k, t_{k+1}) - h(1-\theta) f(x_k, t_k) - hr^0_{k+1}$$

$$\mathcal{R}^0_{k+1} = -h\theta f(x_k, t_{k+1}) - h(1-\theta) f(x_k, t_k) - hr^0_{k+1}$$

The initial residu y:

$$\mathcal{R}^0_{yk+1} = y_k - h(x_k, \lambda_k, t_{k+1})$$

The initial residu r:

$$\mathcal{R}^0_{rk+1} = r_k - g(\lambda_k, t_{k+1})$$

In the case of the system (??) with a affine function f or $\theta = 0$, the the MLCP matrix W can be computed before the beginning of the time loop saving a lot of computing effort. In the case of the system (??) with $\theta = \gamma = 0$, the MLCP matrix W can be computed before the beginning of the Newton loop.

8.0.9 Time–discretization of the linear case (??)

Starting from (??), let us introduce anew notation,

$$\begin{aligned} M\dot{x}(t) &= Ax(t) + r(t) + b(t) \\ y(t) &= h(x(t), \lambda(t), z) = Cx + Fz + D\lambda \\ r(t) &= g(t, \lambda(t)) = B\lambda \end{aligned} \tag{8.49}$$

Let us now proceed with the time discretization of (8.49) by a fully implicit scheme :

$$\begin{aligned} Mx_{k+1}^{\alpha+1} &= Mx_k + h\theta Ax_{k+1}^{\alpha+1} + h(1-\theta)Ax_k + h\theta_r r_{k+1}^{\alpha+1} + h(1-\theta_r)r(t_k) + b \\ y_{k+1}^{\alpha+1} &= Cx_{k+1}^{\alpha+1} + D\lambda_{k+1}^{\alpha+1} + Fz + e \\ r_{k+1}^{\alpha+1} &= B\lambda_{k+1}^{\alpha+1} \end{aligned} \tag{8.50}$$

$$R_{free} = M(x_{k+1}^{\alpha} - x_k) - h\theta Ax_{k+1}^{\alpha} - h(1-\theta)Ax_k - hb_{k+1}$$

$$R_{free} = W(x_{k+1}^{\alpha} - x_k) - hAx_k - hb_{k+1}$$

8.0.10 Resulting Newton step (only one step)

suppose:$\theta_r = 1$

$$\begin{aligned} (M - h\theta A)x_{k+1}^{\alpha+1} &= Mx_k + h(1-\theta)Ax_k + r_{k+1}^{\alpha+1} + hb \\ y_{k+1}^{\alpha+1} &= Cx_{k+1}^{\alpha+1} + D\lambda_{k+1}^{\alpha+1} + Fz + e \\ r_{k+1}^{\alpha+1} &= B\lambda_{k+1}^{\alpha+1} \end{aligned} \tag{8.51}$$

that lead to with: $(M - h\theta A) = W$

$$\begin{aligned} x_{k+1}^{\alpha+1} &= W^{-1}(Mx_k + h(1-\theta)Ax_k + r_{k+1}^{\alpha+1} + hb) = xfree + W^{-1}(r_{k+1}^{\alpha+1}) \\ y_{k+1}^{\alpha+1} &= (D + CW^{-1}B)\lambda_{k+1}^{\alpha+1} + Fz + CW^{-1}(Mx_k + h(1-\theta)Ax_k) + e \\ r_{k+1}^{\alpha+1} &= B\lambda_{k+1}^{\alpha+1} \end{aligned} \tag{8.52}$$

with $x_{free} = x_{k+1}^{\alpha} + W^{-1}(-R_{free}) = x_{k+1}^{\alpha} - W^{-1}(W(x_{k+1}^{\alpha} - x_k) - hAx_k - hb_{k+1}) = W^{-1}(Mx_k + h(1-\theta)Ax_k + hb_{k+1})$

$$\begin{aligned} y_{k+1}^{\alpha+1} &= (D + CW^{-1}B)\lambda_{k+1}^{\alpha+1} + Fz + Cx_{free} + e \\ r_{k+1}^{\alpha+1} &= B\lambda_{k+1}^{\alpha+1} \end{aligned} \tag{8.53}$$

8.0.11 coherence with previous formulation

$$y_p = y_{k+1}^{\alpha} - \mathcal{R}_{yk+1}^{\alpha} + C_{k+1}^{\alpha}(x_p - x_{k+1}^{\alpha}) - D_{k+1}^{\alpha}\lambda_{k+1}^{\alpha}$$

$$y_p = Cx_k + D\lambda_k + C(\tilde{x}_{free}) - D\lambda_k + Fz + e$$

$$y_p = Cx_k + C(\tilde{x}_{free}) + Fz + e$$

$$y_p = Cx_k + C(\tilde{x}_{free}) + Fz + e$$

$$y_p = C(x_{free}) + Fz + e$$

Chapter 9

NewtonEuler

This section is about the Newton-Euler equations:

9.0.12 The dynamical system

$$\begin{aligned} m\dot{V}(t) &= F_{ext} \\ I\dot{\Omega} + \Omega \wedge I\Omega &= M_{ext} \end{aligned} \tag{9.1}$$

Where V is the velocity of the center of mass and Ω is the angular speed in the referential attached to the object.

9.0.13 The relation

X is the position and the orientation, we don't focus on its representation.

$$\begin{aligned} Y &= H(X) \\ R &= G(X, \lambda) \end{aligned} \tag{9.2}$$

The first equation is derived:

$$\dot{Y} = C\dot{X} + \dot{H}$$

It exists an operator T :

$$T : \begin{pmatrix} V \\ \Omega \end{pmatrix} \rightarrow \dot{X}$$

Using T leads to :

$$\dot{Y} = CT \begin{pmatrix} V \\ \Omega \end{pmatrix} + \dot{H}$$

9.0.14 discretization $t_k \rightarrow t_{k+1}$, and implementation in Siconos

The goal of this section is to describe the computation done in Siconos. The unknown are rename using the Siconos convention.

9.0.14.a the unknowns

About the DS:

$$\_v_k = \begin{pmatrix} V_k \\ \Omega_k \end{pmatrix}$$

And $\_q_k$ represent the system, usually it could be the coordinate of the center of mass and a representation of the orientation.

9.0.14.b explicit case

The dynamical system 9.1 leads to the system:

$$\begin{pmatrix} m & 0 \\ 0 & I \end{pmatrix} (\_v_{k+1} - \_v_k) = h\_Fl_k +^t (CT) h\lambda_{k+1} \tag{9.3}$$

With

$$\_Fl_k = \begin{pmatrix} Fext_k \\ Mext_k - \Omega_k \wedge I\Omega_k \end{pmatrix}$$

Using $W = \begin{pmatrix} m & 0 \\ 0 & I \end{pmatrix}^{-1}$

$$\boxed{\_v_{k+1} = W(h\_Fl_k) + W^t(CT)h\lambda_{k+1} + \_v_k} \tag{9.4}$$

This computation is done in Moreau::updateState, using:

$$\_ResiduFree_k = -h\_Fl_k$$

$$Xfree_k = -W\_ResiduFree_k + \_v_k$$

The relation 9.2 leads to the system:

$$\dot{Y}_{k+1} = CT\_v_{k+1} + \dot{H}_k$$

Substitute $\_v_{k+1}$ **using 9.4 leads:**

$$\dot{Y}_{k+1} = CT[Wh\_Fl_k + hW^t(CT)\lambda_{k+1} + \_v_k] + \dot{H}_k = CT[hW^t(CT)\lambda_{k+1} + Xfree_k] + \dot{H}_k$$

Ones gets:

$$\boxed{\dot{Y}_{k+1} = CTW^t(CT)(h\lambda_{k+1}) + CTXfree_k + \dot{H}_k}$$

Solving the one step problem gives $h\lambda_{k+1}$**, from 9.4 we get** $\_v_{k+1}$**.** V **and** Ω **are used to compute** $\dot{\_}q$**, integrated to get** $\_q_{k+1}$**.**

9.0.15 Quaternion case

Working in 3D, we chose $\_q = \begin{pmatrix} X_g \\ q \end{pmatrix}$**.** X_g **are the 3 coordinates of the center of mass, and** q **is a quaternion represented the orientation of solid. It means :**

$$q_k(0, GM_0)q_k^c = (0, GM_k)$$

Where G is the center of mass, and M any point of the solid.
This section describes the T **operator in this case. Computation using quaternion leads to the relation:**

$$\dot{q} = \frac{1}{2} q(0, \Omega)$$

So using the matrix formulation:

$$\dot{q} = \frac{1}{2} \begin{pmatrix} q_0 & -q_1 & -q_2 & -q_3 \\ q_1 & q_0 & -q_3 & q_2 \\ q_2 & q_3 & q_0 & -q_1 \\ q_3 & -q_2 & -q_1 & q_0 \end{pmatrix} \begin{pmatrix} 0 \\ \Omega \end{pmatrix} = T_q \Omega$$

That lead to :

$$\dot{\_}q = \begin{pmatrix} I_3 & 0 \\ 0 & T_q \end{pmatrix} \begin{pmatrix} V \\ \Omega \end{pmatrix} = T \begin{pmatrix} V \\ \Omega \end{pmatrix} = T\_v$$

It is noteworthy that T **must be updated at each step.**

Chapter 10

Simulation of a Cam Follower System

Main Contributors: *Mario di Bernardo, Gustavo Osorio, Stefania Santini*
University of Naples Federico II, Italy.

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 non linear 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

$$\mu\frac{d^2u(t)}{dt^2}+\zeta\frac{du(t)}{dt}+\kappa u(t)=f_v(t), \qquad \textbf{if} \quad u(t)>c(t). \tag{10.1}$$

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

$$v(t^+)=\frac{dc}{dt}-r\left(v(t^-)-\frac{dc}{dt}\right)=(1+r)\frac{dc}{dt}-rv(t^-), \quad \textbf{if} \quad u(t)=c(t). \tag{10.2}$$

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 10.1 is presented the schematic diagram of the physical cam-follower system. In Figure 10.1.a for $t=0$, 10.1.b for $t=\beta$, and 10.1.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.

10.0.16 The cam-follower as a Lagrangian NSDS.

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 (10.1) and (10.2) can be expressed as (the argument t will not be explicitly written)

$$\mu\frac{d^2\hat{u}}{dt^2}+\zeta\frac{d\hat{u}}{dt}+\kappa\hat{u}=f_v-\left(\mu\frac{d^2c}{dt^2}+\zeta\frac{dc}{dt}+\kappa c\right) \equiv \hat{f}, \qquad \textbf{if} \quad \hat{u}>0. \tag{10.3}$$

$$\hat{v}^+ = -r\hat{v}^-, \quad \textbf{if} \quad \hat{u}=0. \tag{10.4}$$

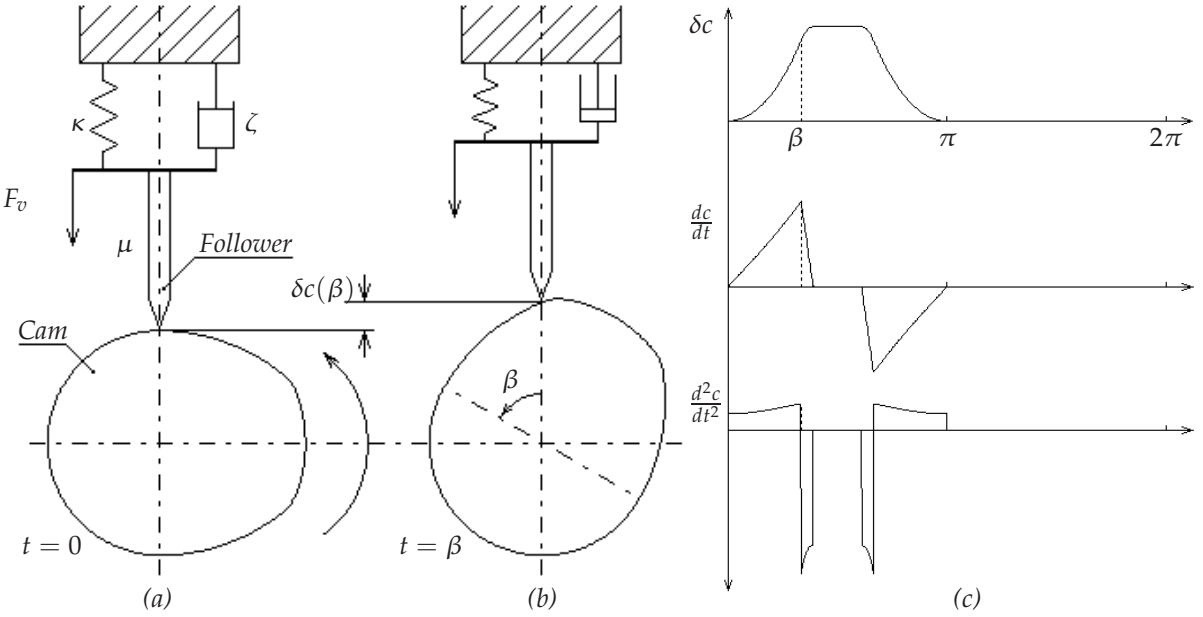


Figure 10.1: Cam-Shaft's schematics. *(a)* t=0. *(b)* t=β. *(c)* Constraint position $\delta c(t)$, velocity $\frac{dc}{dt}(t)$ and acceleration $\frac{d^2c}{dt}(t^2)$.

Using the framework presented in [2] we have that the equation of motion of a Lagrangian system may be stated as follows :

$$M(q)\ddot{q} + Q(q,\dot{q}) + F(\dot{q},q,t) = F_{ext}(t) + R \tag{10.5}$$

From the (10.3) we can derive all of the terms which define a Lagrangian NSDS. In our case the model is completely linear:

$$\begin{aligned} q &= \begin{bmatrix} \hat{u} \end{bmatrix} \\ M(q) &= \begin{bmatrix} \mu \end{bmatrix} \\ Q(q,\dot{q}) &= \begin{bmatrix} 0 \end{bmatrix} \\ F(q,\dot{q}) &= \begin{bmatrix} \zeta \end{bmatrix}\dot{q} + \begin{bmatrix} \kappa \end{bmatrix} q \\ F_{ext} &= \begin{bmatrix} \hat{f} \end{bmatrix} \end{aligned} \tag{10.6}$$

The unilateral constraint requires that:

$$\hat{u} \geq 0$$

so we can obtain

$$\begin{aligned} y &= H^T q + b \\ H^T &= \begin{bmatrix} 1 \end{bmatrix} \\ b &= 0 \end{aligned} \tag{10.7}$$

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

$$R = H\lambda, \qquad \textbf{with } H = \begin{bmatrix} 1 \end{bmatrix}$$

The unilataral contact law may be formulated as follow:

$$0 \leq y \perp \lambda \geq 0 \tag{10.8}$$

and the Newton's impact law:

$$\textbf{If } y = 0, \dot{y}^+ = -r\dot{y}^- \tag{10.9}$$

10.0.17 Implementation in the platform

For the simulation of the cam follower system follow the steps

1. **Move to the working directory** `sample/CamFollower`

   ```
   $cd sample/CamFollower
   ```

2. **Clean the directory form binary files using the** `siconos` **command**

   ```
   $siconos -c
   ```

3. **Compile the file** `CamFollowerNoXml.cpp` **in the sample folder (*See* the code at the end of the section)**

   ```
   $siconos CamFollowerNoXml.cpp
   ```

4. **Change the simulation parameters (*i.e.* Follower initial position and velocity, cam initial angle, simulations time, cam rotational speed in rpm, etc.) in the file** `CamFollowerNoXml.cpp`**.**

Next we present the sample code for the `CamFollowerNoXml.cpp` **file:**

```
int main(int argc, char* argv[]) {
  {
    // ======== Creation of the model =============
    // User-defined main parameters
    double rpm=358;
    double phi_0=0;
    unsigned int dsNumber = 1;     // the Follower and the ground
    unsigned int nDof = 1;         // degrees of freedom for the Follower
    double t0 = 0;                 // initial computation time
    double T = 5;                  // final computation time
    double h = 0.0001;             // time step
    int Kplot;
    Kplot=(int)(Tplot/h);
    double position_init = 0.4;    // initial position for lowest bead.
    double velocity_init = 0.4;    // initial velocity for lowest bead.

    // ======= Dynamical systems =========

    vector<DynamicalSystem *> vectorDS; // the list of DS
    vectorDS.resize(dsNumber,NULL);

    SiconosMatrix *Mass, *K, *C; // mass/rigidity/viscosity
    Mass = new SiconosMatrix(nDof,nDof);
    (*Mass)(0,0) = 1.221;
    K = new SiconosMatrix(nDof,nDof);
    (*K)(0,0) = 1430.8;
    C = new SiconosMatrix(nDof,nDof);
    (*C)(0,0) = 0;

    // Initial positions and velocities
    vector<SimpleVector *> position_0;
    vector<SimpleVector *> velocity_0;
    position_0.resize(dsNumber,NULL);
    velocity_0.resize(dsNumber,NULL);
    position_0[0] = new SimpleVector(nDof);
    velocity_0[0] = new SimpleVector(nDof);
```

```
(*(position_0[0]))(0) = position_init;
(*(velocity_0[0]))(0) = velocity_init;

vectorDS[0] =
new LagrangianLinearTIDS(0,nDof,*(position_0[0]),*(velocity_0[0]),*Mass,*K,*C);

static_cast<LagrangianDS*>(vectorDS[0])
                    ->setComputeFExtFunction("FollowerPlugin.so", "FollowerFExt");

// Example to set a list of parameters in FExt function.
// 1 - Create a simple vector that contains the required parameters.

// Here we set two parameters, the DS number.
SimpleVector * param = new SimpleVector(2);

(*param)(0)=rpm;
(*param)(1)=phi_0;
// 2 - Assign this param to the function FExt
static_cast<LagrangianDS*>(vectorDS[0])->setParametersListPtr(param,2);
// 2 corresponds to the position of FExt in the stl vector of possible parameters.
// 0 is mass, 1 FInt.
// Now the cam rotational velocity in rpms will be available in FExt plugin.

// ===== Interactions =====

vector<Interaction*> interactionVector;
interactionVector.resize(1,NULL);
vector<DynamicalSystem*> *dsConcerned =
                    new vector<DynamicalSystem*>(dsNumber);

// ===== Non Smooth Law =====
double e = 0.8;
// Interaction Follower-floor
SiconosMatrix *H = new SiconosMatrix(1,nDof);
(*H)(0,0) = 1.0;
NonSmoothLaw * nslaw = new NewtonImpactLawNSL(e);
Relation * relation = new LagrangianLinearR(*H);
(*dsConcerned)[0] = vectorDS[0];
interactionVector[0] = new Interaction("Follower-Ground",0,1, dsConcerned);
interactionVector[0]->setRelationPtr(relation);
interactionVector[0]->setNonSmoothLawPtr(nslaw);
// ===== Interactions =====

// ===== NonSmoothDynamicalSystem =====
bool isBVP =0;
NonSmoothDynamicalSystem * nsds =
                        new NonSmoothDynamicalSystem(isBVP);

// Set DS of this NonSmoothDynamicalSystem
nsds->setDynamicalSystems(vectorDS);
// Set interactions of the NonSmoothDynamicalSystem
nsds->setInteractions(interactionVector);

// ===== Model =====
```

```
Model * Follower = new Model(t0,T);
// set NonSmoothDynamicalSystem of this model
Follower->setNonSmoothDynamicalSystemPtr(nsds);

// ====== Strategy ======

double theta = 0.5;          // theta for Moreau integrator
string solverName = "QP" ;

Strategy* S = new TimeStepping(Follower);

// – Time discretisation –
TimeDiscretisation * t = new TimeDiscretisation(h,S);

// – OneStepIntegrators –
vector<OneStepIntegrator *> vOSI;
vOSI.resize(dsNumber,NULL);
vOSI[0] = new Moreau(t,vectorDS[0],theta);
S->setOneStepIntegrators(vOSI);

// – OneStepNsProblem –
OneStepNSProblem * osnspb = new LCP(S,solverName,101, 0.0001,"max",0.6);
S->setOneStepNSProblemPtr(osnspb); // set OneStepNSProblem of the strategy
cout « "=== End of model loading === " « endl;
// ==== End of model definition======



// ========= Computation============

// — Strategy initialization —
S->initialize();
cout «"End of strategy initialisation" « endl;

int k = t->getK();                  // Current step
int N = t->getNSteps();             // Number of time steps

// — Get the values to be plotted —
// -> saved in a matrix dataPlot
unsigned int outputSize = 8;

SiconosMatrix DataPlot(Kplot+1,outputSize );
// For the initial time step:

// time
DataPlot(k,0) = k*t->getH();

DataPlot(k,1) = static_cast<LagrangianDS*>(vectorDS[0])->getQ()(0);
DataPlot(k,2) = static_cast<LagrangianDS*>(vectorDS[0])->getVelocity()(0);
DataPlot(k,3) = (Follower->getNonSmoothDynamicalSystemPtr()->
                 getInteractionPtr(0)->getLambda(1))(0);
DataPlot(k,4) = static_cast<LagrangianDS*>(vectorDS[0])->getFExt()(0);
```

```
// State of the Cam
double CamEqForce,CamPosition,CamVelocity,CamAcceleration;
CamEqForce=
        CamState(k*t->getH(),rpm,CamPosition,CamVelocity,CamAcceleration);
// Position of the Cam
DataPlot(k, 5) = CamPosition;
// Velocity of the Cam
DataPlot(k, 6) = CamVelocity;
// Acceleration of the Cam
DataPlot(k, 7) =
        CamPosition+static_cast<LagrangianDS*>(vectorDS[0])->getQ()(0);

// --- Time loop ---
cout << "Start computation ... " << endl;
while(k < N)
{

  // --- Get values to be plotted ---
  DataPlot(k,0) = k*t->getH();

  DataPlot(k,1) =
        static_cast<LagrangianDS*>(vectorDS[0])->getQ()(0);
  DataPlot(k,2) =
        static_cast<LagrangianDS*>(vectorDS[0])->getVelocity()(0);
  DataPlot(k,3) =
        (Follower->getNonSmoothDynamicalSystemPtr()->
        getInteractionPtr(0)->getLambda(1))(0);
  DataPlot(k,4) = static_cast<LagrangianDS*>(vectorDS[0])->getFExt()(0);

  CamEqForce=
  CamState(k*t->getH(),rpm,CamPosition,CamVelocity,CamAcceleration);

  DataPlot(k, 5) = CamPosition;
  DataPlot(k, 6) = CamVelocity;
  DataPlot(k, 7) = CamPosition+
        static_cast<LagrangianDS*>(vectorDS[0])->getQ()(0);
  // transfer of state i+1 into state i and time incrementation
  S->nextStep();
  // get current time step
  k = t->getK();
  // solve ...
  S->computeFreeState();
  S->computeOneStepNSProblem();
  // update
  S->update();
}
// --- Output files ---
DataPlot.rawWrite("result.dat", "ascii");
// --- Free memory ---
delete osnspb;
delete vOSI[0];
delete t;
delete S;
delete Follower;
```

```
      delete nsds;
      delete interactionVector[0];
      delete relation;
      delete nslaw;
      delete H;
      delete dsConcerned;
      delete vectorDS[0];
      delete position_0[0];
      delete velocity_0[0];
      delete C;
      delete K;
      delete Mass;

}
```

10.0.18 Simulation

We have perform the simulation of the cam follower system for different values of the cam rotational speed with the SICONOS software package using a time-stepping numerical scheme with step size ($h = 1e^{-4}$) and an event-driven scheme with minimum step size ($h_{min} = 1e^{-12}$). Fig. 10.2 and 10.3 show the time simulations for different values of the cam rotational speed and Fig. 10.4 show the chaotic attractor at $rpm = 660$ for impact and stroboscopic Poincarè sections.

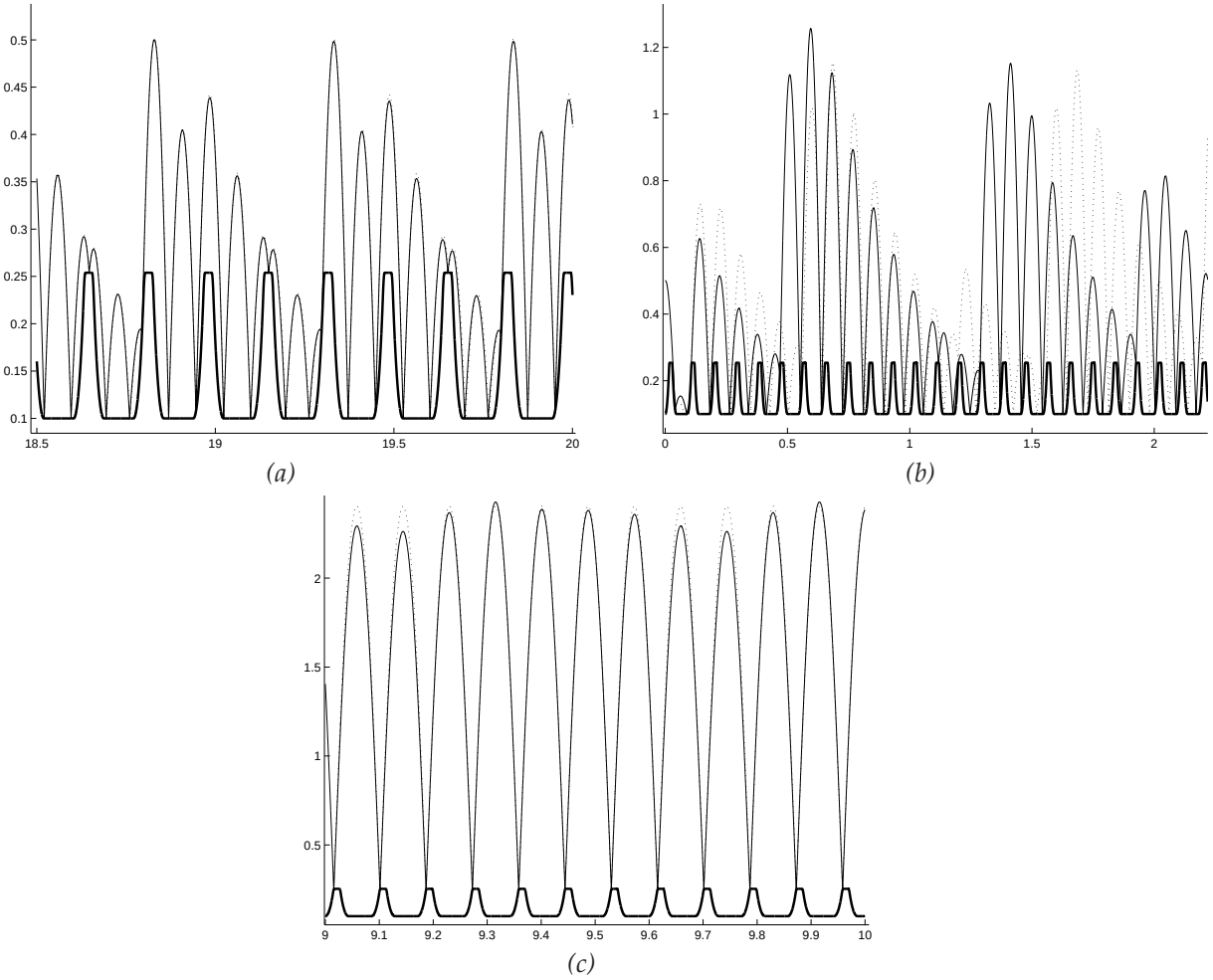


Figure 10.2: Time series using SICONOS platform. Time-stepping scheme (continuous line). Event-driven scheme (dashed line) *(a)* rpm=358. *(b)* rpm=660. *(c)* rpm=700.

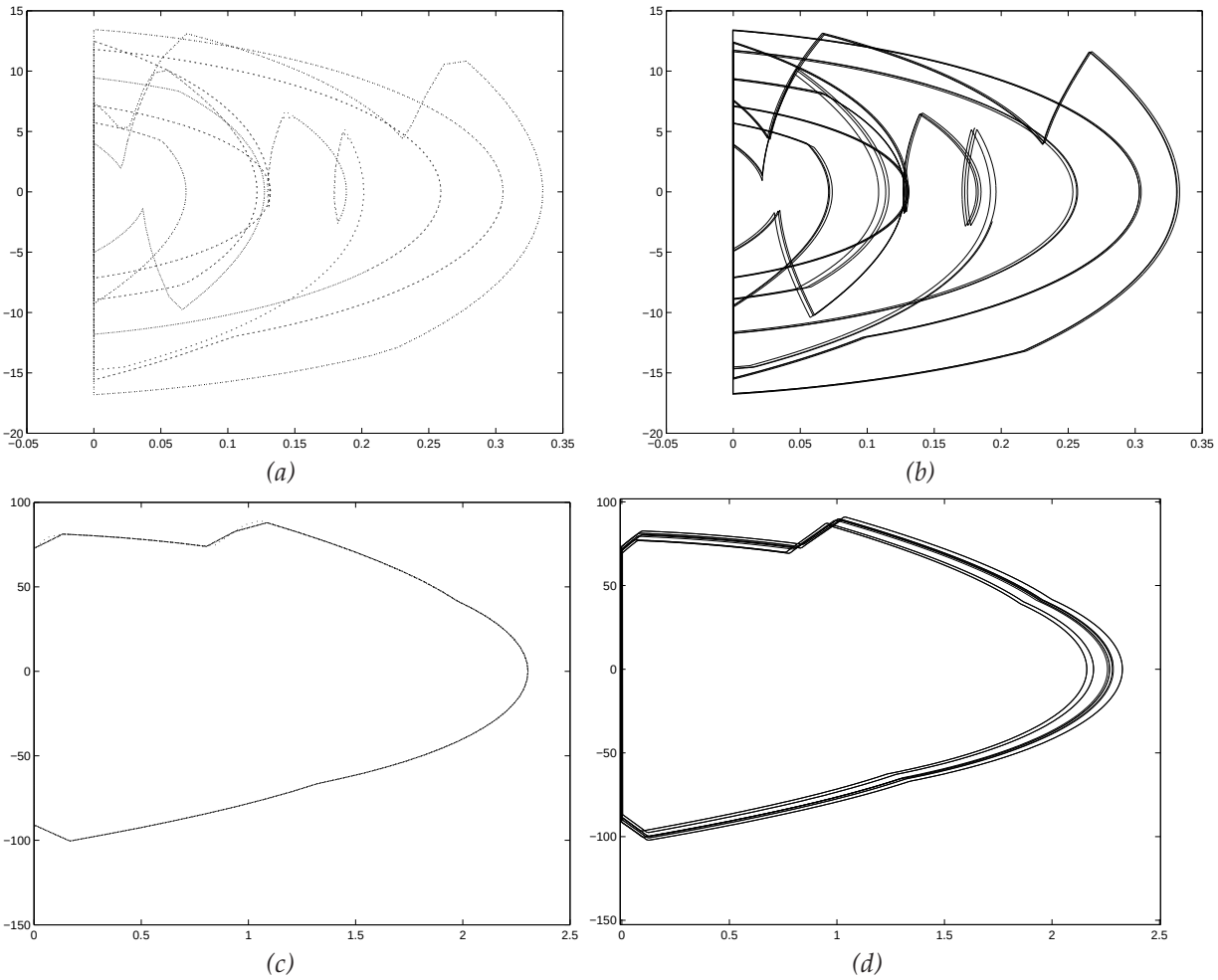


Figure 10.3: State space comparison using SICONOS platform. *(a)* rpm=358. Event Driven *(b)* rpm=358. Time Stepping ($h = 1e^{-4}$)*(c)* rpm=700. Event Driven *(d)* rpm=700. Time Stepping ($h = 1e^{-4}$)

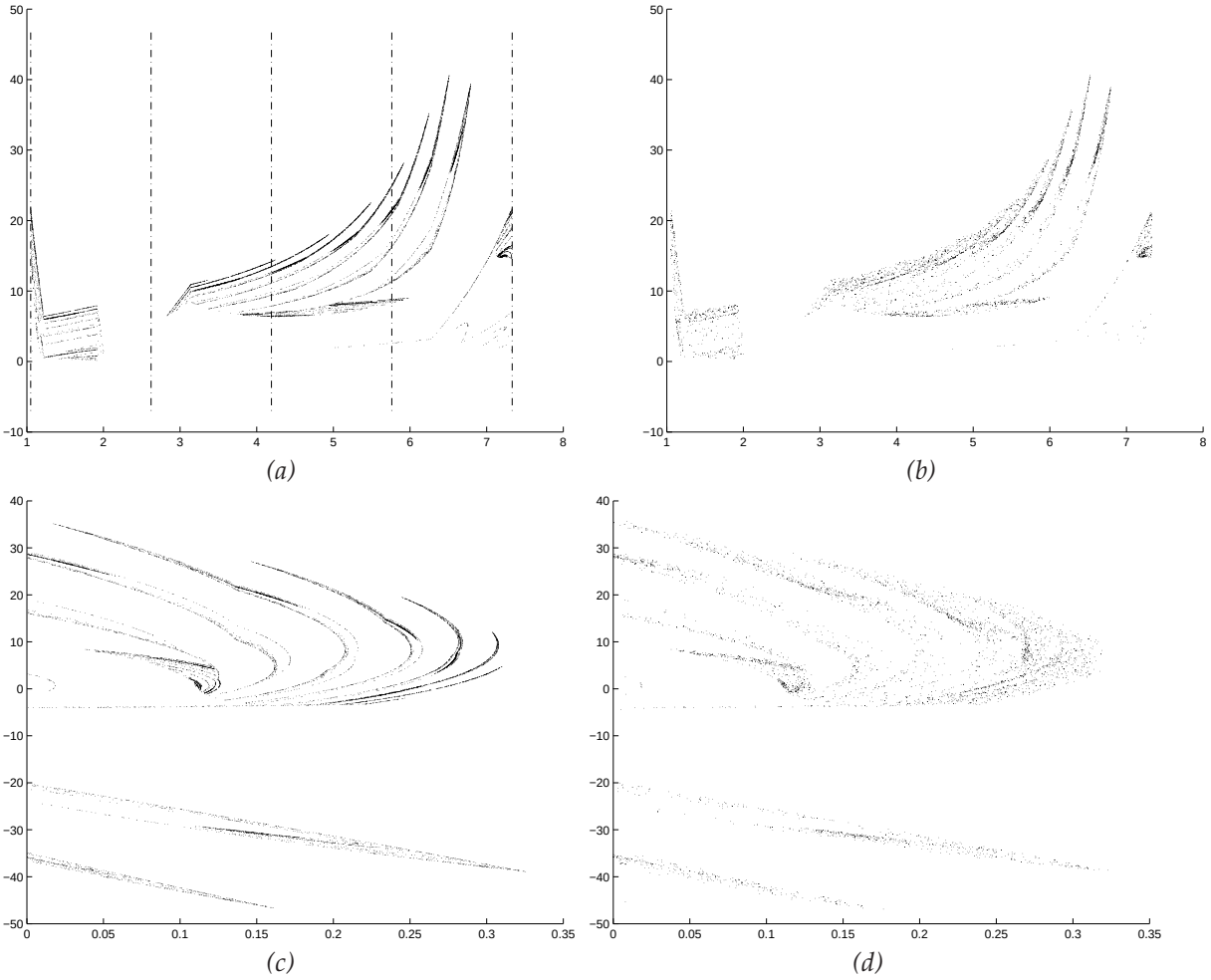


Figure 10.4: Attractors comparison using SICONOS platform at rpm=660. *(a)* Impact map. (Event Driven) *(b)* Impact Map. Time Stepping ($h = 1e^{-4}$)*(a)* Stroboscopic map. (Event Driven) *(b)* Stroboscopic Map. Time Stepping ($h = 1e^{-4}$)